Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Engenharia Elétrica

Tese de Doutorado

Detecção de Intrusos em Redes de Computadores com uso de Códigos Corretores de Erros e Medidas de Informação

Christiane Ferreira Lemos Lima

Francisco Marcos de Assis
Orientador
Cleonilson Protásio de Souza
Orientador

Campina Grande – PB

2013

Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Programa de Pós-Graduação em Engenharia Elétrica

Detecção de Intrusos em Redes de Computadores com uso de Códigos Corretores de Erros e Medidas de Informação

Christiane Ferreira Lemos Lima

Tese de Doutorado submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Doutor em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Processamento da Informação.

Francisco Marcos de Assis
Orientador
Cleonilson Protásio de Souza
Orientador

Campina Grande – PB
2013

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

L732d Lima, Christiane Ferreira Lemos.

Detecção de intrusos em redes de computadores com uso de códigos corretores de erros e medidas de informação / Christiane Ferreira Lemos Lima. -- Campina Grande, 2013.

183 f. : il. color.

Tese (Doutorado em Engenharia Elétrica) - Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática, 2013.

"Orientação: Prof. Dr. Francisco Marcos de Assis, Prof. Dr. Cleonilson Protásio de Souza".

Referências.

1. Detecção de Intrusos. 2. Classificação Multiclasses. 3.Codificação. 4. Decodificação. 5. Medidas de Informação. I. Assis, Francisco Marcos de. II. Souza, Cleonilson Protásio de. III. Título.

CDU 004.052.42(043)

"DETECÇÃO DE INTRUSOS EM REDES DE COMPUTADORES COM USO DE CÓDIGOS CORRETORES DE ERROS E MEDIDAS DE INFORMAÇÃO"

CHRISTIANE FERREIRA LEMOS LIMA

TESE APROVADA EM 19/04/2013

FRANCISCO MARCOS DE ASSIS, Dr., UFCG
Orientador(a)

CLEONILSON PROTÁSIO DE SOUZA, D.Sc., UFPB
Orientador(a)

JOSEANA MACEDO FECHINE RÉGIS DE ARAÚJO, D.Sc., UFCG
Examinador(a)

ANA CAROLINA LORENA, Dr., UFABC
Examinador(a)

CHARLES CASIMIRO CAVALCANTE, Dr., UFC
Examinador(a)

LUIZ FELIPE QUEIROZ SILVEIRA, D.Sc., UFRN
Examinador(a)

EDSON GUEDES DA COSTA, D.Sc., UFCG
Examinador(a)

CAMPINA GRANDE - PB

Aos meus pais, Raimundo e Marinalva.
Aos meus filhos, Arthur e Thaís.
A Orlando, meu esposo.

Agradecimentos

Muitas pessoas contribuíram de forma direta e indiretamente para a realização deste trabalho. Para relacioná-las todas, certamente muitas páginas a mais seriam necessárias. Portanto, cito algumas sob o risco de ser ingrata com muitas. Mas, sem dúvida, todas estarão presentes em minha lembrança.

Inicialmente, agradeço a Deus por todas as provisões que tornaram este dia uma realidade, permitindo que eu aprendesse e progredisse por meio do trabalho, perseverasse mesmo diante dos obstáculos e encontrasse suporte e colaboração de pessoas tão especiais ao longo da minha jornada.

Ao Prof. Dr. Francisco Marcos de Assis, com imensa gratidão por ter aceitado me orientar e por acreditar em meu potencial, bem como pela sua disponibilidade, paciência e atenção para comigo, formulando indagações e fornecendo sugestões as quais possibilitaram obter os resultados necessários para esta tese. Apesar de minhas "caminhadas em círculos", continuava acreditando em mim, fazendo com que atalhos fossem tomados sempre que possível. Certamente, o seu entusiasmo como pesquisador e a sua postura profissional servirão como exemplo para mim, por toda a vida.

Ao Prof. Dr. Cleonilson Protásio de Souza, que colaborou decisivamente para o sucesso deste trabalho, dando preciosas sugestões e tendo disponibilizando parte do seu tempo e atenção nas correções dos textos e artigos, assim como me concedeu autonomia para que eu encontrasse o meu caminho.

Aos membros da banca, pela apreciação do meu trabalho e pelas valiosas sugestões de melhoria.

À Profa. Dra. Valeska Martins de Souza e ao Prof. Dr. Josenildo de Souza Chaves, ambos da UFMA, que colaboram para o sucesso deste trabalho, tendo disponibilizando parte do seus tempos e atenção para tirar dúvidas e discutir indagações que surgiram ao longo deste trabalho. À Dra. Heloísa Musseti Ruivo, INPE, por seu interesse e indagações em relação a esta pesquisa. Aos Professores MSc. Fábio Rizental, UTFPR, Dr. Morgan Barbier, Universidade de Caen, e Dr. Li Chen, Universidade Sun Yat-Sen, pela atenção e disponibilidade em contribuir com este trabalho.

Ao meu esposo, Orlando, e nossos filhos, Thaís e Arthur, por me apoiarem em todos os momentos, e dedicarem parte de suas vidas nessa trajetória, que deixou de ser somente minha e passou a ser nossa, desde o início.

Aos meus pais, Raimundo e Marinalva, meus queridos irmãos, Bianca e Guélter, e sobrinhos, pois eles são partes de mim, acreditam e me apoiam incondicionalmente. Aos amigos e familiares, pelas palavras de incentivo e credibilidade no meu profissionalismo.

Aos colegas e amigos da pós-graduação, cujo convívio foi um grande aprendizado. Em especial a Cláudio Leão, Elloá, Gilson, Luís Hermano e Vinícius, apesar de estarem ocupados com seus trabalhos, sempre arrumavam tempo para me dar o apoio necessário, colaborando com inestimáveis sugestões, bem como me permitiam compartilhar ansiedades, conhecimentos e, principalmente, tinham sempre palavras de incentivo e perseverança.

A todo o pessoal administrativo da UFCG/COPELE, especialmente à Ângela e Pedro, pela atenção e pela eficiência no trato das questões burocráticas.

Ao IFMA, pelo apoio durante todo o período de doutorado.

À FAPEMA e à CAPES, que possibilitaram o desenvolvimento desta pesquisa por meio da concessão de apoio financeiro e ao IQuanta pelo suporte geral.

Jesus, o Código para a Vida Eterna.

DEUS → JESUS → HOMEM

"Quando o HOMEM entender a mensagem de DEUS para sua vida,
através de JESUS, ele entenderá o verdadeiro propósito da sua existência.
Este código é simples, só para quem quiser"

—ALONSO SEPÚLVEDA CASTELLANOS

Resumo

Este trabalho de tese tem como objetivo principal apresentar um novo esquema, direcionado à proteção de sistemas computacionais contra a ocorrência de invasões, fazendo uso de códigos corretores de erros e de medidas de informação. Para isto, considera-se que a identificação de diferentes tipos de ataques a uma rede de computadores pode ser vista como uma tarefa de classificação multiclasses, por envolver a discriminação de ataques em diversos tipos ou categorias. Com base nessa abordagem, o presente trabalho apresenta uma estratégia para classificação multiclasses, baseada nos princípios dos códigos corretores de erros, tendo em vista que para cada uma das M classes do problema proposto é associada a um conjunto de palavras códigos de comprimento igual a N, em que N é o número de atributos, selecionados por meio de medidas de informação, e que serão monitorados por dispositivos de software, aqui chamados de detectores de rede e detectores de *host*. Nesta abordagem, as palavras código que formam uma tabela são restritas a um sub-código de um código linear do tipo BCH (Bose-Chaudhuri-Hocquenghem), permitindo que a etapa de decodificação seja realizada, utilizando-se algoritmos de decodificação algébrica, o que não é possível para palavras código selecionadas aleatoriamente. Nesse contexto, o uso de uma variante de algoritmo genético é aplicado no projeto da tabela que será utilizada na identificação de ataques em redes de computadores. Dentre as contribuições efetivas desta tese, cujas comprovações são demonstradas em experimentos realizados por simulação computacional, tem-se a aplicação da teoria da codificação para a determinação de palavras código adequadas ao problema de detectar intrusões numa rede de computadores; a determinação dos atributos por meio do uso de árvores de decisão C4.5 baseadas nas medidas de informação de Rényi e Tsallis; a utilização de decodificação algébrica, baseado nos conceitos de decodificação tradicional e na decodificação por lista.

Palavras-chave: Detecção de Intrusão, Classificação Multiclasses, Codificação, Decodificação, Medidas de Informação, Algoritmos Genéticos, Árvore de Decisão.

Abstract

The thesis's main objective is to present a scheme to protect computer networks against the occurrence of invasions by making use of error correcting codes and information measures. For this, the identification of attacks in a network is viewed as a multiclass classification task because it involves attacks discrimination into various categories. Based on this approach, this work presents strategies for multiclass problems based on the error correcting codes principles, where each M class is associated with a codeword of length N, where N is the number of selected attributes, chosen by use of information measures. These attributes are monitored by software devices, here called network detectors and detectors host. In this approach, the codewords that form a codewords table are a sub-code of a BCH-type linear code. This approach allows decoding step to be performed using algebraic decoding algorithms, what is not possible with random selected codewords. In this context, the use of a variant of genetic algorithm are applied in the table-approach design to be used in attacks identification in networks. The effective contributions of this thesis, demonstrated on the cientific experiments, are: the application of coding theory to determine the appropriate codewords to network intrusion detection; the application of C4.5 decision tree based on information measurements of Rényi and Tsallis for attributes selection; the use of algebraic decoding, based on the concepts of the traditional decoding and list decoding techniques.

Keywords: Intrusion Detection, Multiclass Classification, Codification, Decoding, Information Measures, Genetic Algorithm, Decision Tree.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

Acc Acurácia do classificador

AD Árvore de Decisão

AG Algoritmo Genético

aiNet Rede imunológica

AIRS Sistema de reconhecimento imunológico artificial

AIS Sistema Imunológico Artificial

ARTIS *Artificial Immune System*

AUC Área abaixo da curva ROC

BCH Bose-Chaudhuri-Hocquenghem

BMA Algoritmo de *Berlekamp-Massey*

CGI *Common Gateway Interface*

CIDF *Common Intrusion Detection Framework*

CLONALG Algoritmo de seleção clonal

COFINS Contribuição para o Financiamento da Seguridade Social

CSCA Sistema de classificação clonal

DoS Negação de Serviço

err Taxa de erro do classificador

E/S Entrada e Saída de dados

FN Falso Negativo

FP Falso Positivo

FTP *File Transfer Protocol*

GSA Algoritmo de *Guruswami-Sudan*

HTTP *HyperText Transfer Protocol*

ICMS Imposto sobre operações relativas à circulação de mercadorias e sobre prestações de serviços de transporte interestadual, intermunicipal e de comunicação

ICMP *Internet Control Message Protocol*

IP *Internet Protocol*

IMMUNOS-99 Sistema Imunológico Artificial

KDD Descoberta de Conhecimento em Base de Dados

LISYS *Lightweight Intrusion Detection System*

MDS Separado por Máxima Distância

MDL *Minimum Description Length*

OBX *Order Based Crossover*

PBX *Position-Based Crossover*

PIS Programa de Integração Social

PMX *Partially Mapped Crossover*

PR Precisão de um classificador

R2L Remoto para Local

ROC *Receiver Operating Characteristic*

RS *Reed Solomon*

SANTA *Security Agents for Network Traffic Analysis*

SDI Sistema de Detecção de Intrusão

TDIDT *Top-Down Induction of Decision Tree*

TCP *Transmission Control Protocol*

TFP Taxa de Falso Positivo

TFN Taxa de Falso Negativo

TVP Taxa de Classificações Corretas

U2R Usuário para Superusuário

UCP Unidade Central de Processamento

UDP *User Datagram Protocol*

URL *Uniform Resource Locator*

VP Verdadeiros Positivos

VN Verdadeiros Negativos

WEKA *Waikato Environment for Knowledge Analysis*

Lista de Símbolos

α Coeficiente que pode ser usado para ajustar a sensibilidade em relação à distribuição de probabilidade

A Variável aleatória discreta definida num alfabeto com η símbolos

$ap(\imath)$ Variável que contém valor da aptidão do indivíduo $\imath$

C Variável aleatória discreta definida num alfabeto com M símbolos

$C_{M\times N}$ Matriz código

$\mathbf{C}_\ell$ Vetor que corresponde à linha da matriz código $C_{M\times N}$

c_ℓ Refere-se a uma classe do problema

D Conjunto de Teste

d Distância

d_{min} Distância mínima do código

$\mathbf{F}(\mathbf{x})$ Vetor de hipóteses gerado pelos classificadores

$GF(q)$ Corpo finito de q elementos

h Variável que contém as observações dos nós sensores

g Variável que controla o número de gerações no algoritmo genético

$\imath$ Refere-se ao $\imath$-ésimo indivíduo da população $P(g)$

i Refere-se à i-ésima instância do conjunto T

j Indica o j-ésimo atributo (ou classificador)

k Número de símbolos da palavra de informação

ℓ Indica a linha da matriz código $C_{M\times N}$ ou indica uma classe

l Refere-se à l-ésima instância do conjunto D

M Pode-se referir ao número de classes ou número de linhas de uma matriz código

m Número de exemplos na base de dados de teste

$\aleph$ Número de atributos utilizado no processo de indução de um classificador

N Pode-se referir ao número de subproblemas binários, ou número de detectores, ou número de colunas da matriz código.

n Número de símbolos da palavra código

η Número de valores distintos de um atributo

p_c Taxa de cruzamento

p_i Probabilidade de um indivíduo ser selecionado para reprodução

$P(g)$ População de indivíduos no AG

p_ℓ Probabilidade de uma instância pertencer à classe c_ℓ

p_m Taxa de mutação

T Conjunto de Treinamento

t Capacidade de correção de erro tradicional

τ Capacidade de correção de erro usando decodificação por lista

$T_{w \times n}$ Tabela de palavras código

v Limiar de partição

W Número de palavras código que define um conjunto de classes

$\vec{x}$ Objeto de treinamento

$\vec{z_l}$ Objeto de teste

y Variável que tem valores discretos e pertence a uma das M classes

CAPÍTULO 1

Introdução

Neste capítulo é apresentada uma descrição desta tese, com o objetivo de fornecer uma visão geral dos problemas tratados e dos objetivos principais do trabalho de pesquisa realizado. Em seguida, é apresentada a organização do documento, com uma descrição resumida do conteúdo abordado em cada capítulo.

1.1 Motivação

A tecnologia da informação tornou-se uma grande ferramenta estratégica na chamada Nova Economia, em que a moeda corrente é a informação e o uso da tecnologia é fundamental para o progresso das empresas. Esse novo panorama traz consigo muitos benefícios às organizações, tais como redução de custos e o fornecimento de serviços cada vez mais atraentes aos clientes, além de promover grandes oportunidades de negócios. Por conseguinte, os maiores ativos das corporações são constituídos por seus dados e dos dados de seus clientes, cuja confidencialidade, integridade e disponibilidade lhes garante poder competitivo no mercado e credibilidade perante seus usuários e um incidente de intrusão pode ocasionar grandes repercussões.

A exemplo do referido modelo de economia, tem-se o uso dos cartões de crédito e débito. Segundo um estudo encomendado pela administradora de cartões Visa do Brasil à Tendências Consultoria Integrada, o uso de cartões contribui significativamente para o aquecimento da economia e para a formalização da atividade comercial, uma vez que apenas estabelecimentos registrados podem se credenciar para aceitar cartões como meio de pagamento, gerando, assim, maior arrecadação de impostos. O levantamento aponta que, se 35% das vendas do comércio fossem pagas com cartões, a arrecadação de ICMS e PIS/Cofins aumentaria em aproximadamente 14%, o que representaria um incremento de R$ 5,2 bilhões na economia [1].

Além disso, a Rede Visa possui a capacidade de processar mais de 20.000 transações por segundo em todo o mundo. No dia 23 de dezembro de 2010, o sistema Visa atingiu uma média de quase 11 mil transações por segundo, registrando mais de 268,4 milhões de transações no minuto mais movimentado do ano. Considerando a média do ticket por transação no valor de

R$ 50,00, então se o sistema ficasse inoperante por um minuto, devido a um ataque de negação de serviço, mais de R$ 13,42 bilhões em transações poderiam ser perdidos [2].

Frente às demandas apresentadas, torna-se imprescindível a presença de mecanismos que visem dar apoio à segurança computacional, visto que o número de incidentes registrados aumentou ao longo dos últimos anos. De acordo com o Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil (CERT.br), só em 2011, foram quase 400 mil incidentes reportados, um aumento de mais de 32 vezes a partir de 2001, e até setembro de 2012 foram registrados aproximadamente 357 mil incidentes [3]. Esses números alarmantes podem ser ainda maiores, porque muitas corporações não divulgam ou desconhecem o fato de terem sido invadidas.

Um dos fatores que contribui para o aumento do número de incidentes de intrusão é que diariamente são divulgadas notícias de vulnerabilidades descobertas nos sistemas computacionais. Isso ocorre pois, como novas tecnologias e novos sistemas estão sendo continuamente criados, é razoável considerar que novas vulnerabilidades sempre existirão. Assim, uma vez que é improvável que um sistema não tenha vulnerabilidade, então é factível que intrusões possam ocorrer [4].

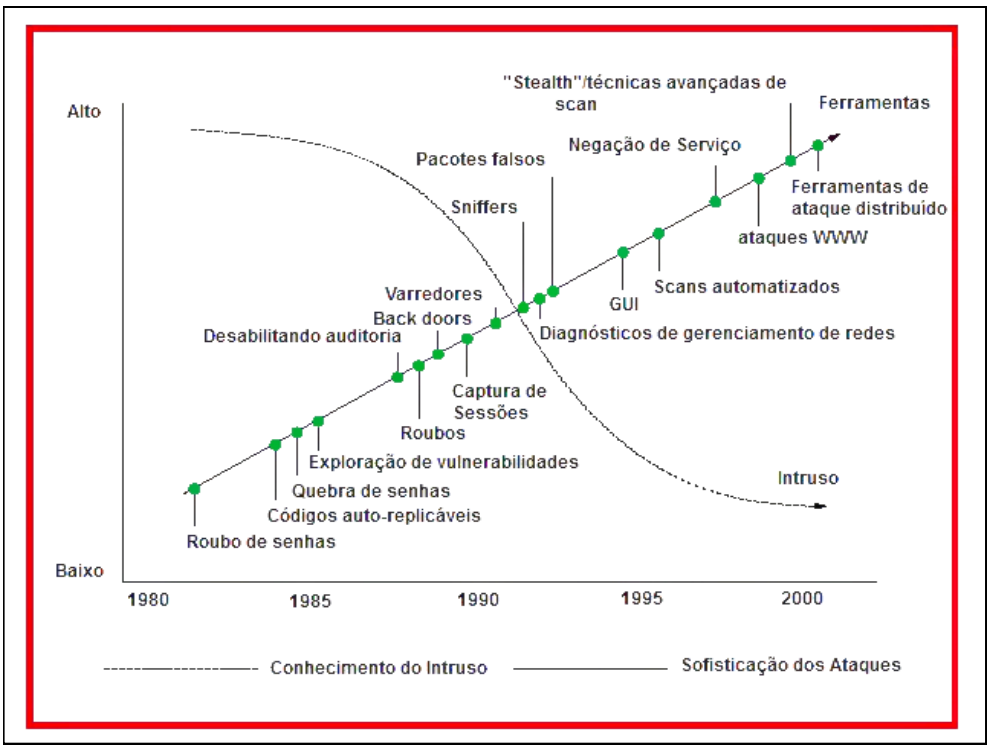
Verifica-se também que qualquer erro ou esquecimento na configuração de um sistema computacional é o suficiente para que se caracterize uma falha na segurança, que poderá ser explorada por pessoas maliciosas, colocando governos, negócios e usuários individuais sob risco constante. Outro problema está relacionado com os usuários legítimos com má intenção, ou negligentes com a segurança, que continuam sendo grandes responsáveis pelo número de incidentes ocorridos nas corporações [5].

Do mesmo modo, a própria história apresenta uma evolução constante das técnicas usadas para ataques, que estão cada vez mais acessíveis para os usuários em geral, permitindo que invasores, com conhecimentos técnicos limitados, mas dispondo de ferramentas sofisticadas, possam efetivar ataques bem sucedidos, conforme ilustrado na Figura 1.1 [6]. A mistura de diferentes técnicas, o uso de tecnologia para cobrir vestígios, a cooperação entre intrusos, a criatividade, dentre outros fatores, contribuem para aumentar o grau de dificuldade para se manter um sistema protegido contra invasões.

Nessa perspectiva, ferramentas que podem ajudar a elevar o patamar de segurança das instituições e empresas são os Sistemas de Detecção de Intrusão (SDI). Um SDI realiza o processo de monitorar, detectar e, em alguns casos, isolar atividades suspeitas ocorridas em um sistema ou rede de computadores em particular, por meio da análise dos pacotes que trafegam nos segmentos de rede e/ou dos registros de auditoria dos sistemas monitorados, baseando-se em semelhanças com assinaturas de intrusão previamente definidas ou desvio de padrão em perfis de comportamento [7–10].

O SDI é um componente importante em um sistema de segurança e complementa outras tecnologias. Ao fornecer informações à administração de segurança, permite a identificação não só de ataques já conhecidos, mas também de novos ataques imprevistos por outros compo-

Figura 1.1 Sofisticação de Ataques x Conhecimento Técnico dos Intrusos [6], adaptado pela autora.



nentes. Outrossim, podem fornecer informações forenses que potencialmente permitam que as organizações descubram as origens de um ataque [7].

Contudo, uma das dificuldades na detecção de intrusos reside no volume de tráfego que circula na rede e no tamanho elevado dos arquivos de auditoria, tornando inviável para especialistas supervisionarem todos os sinais de suspeitas de intrusão. Assim, faz-se necessário o uso de técnicas que possibilitem a descoberta de padrões, similaridades e conhecimento de forma automática, a partir de dados, facilitando, dessa forma, a análise das informações a serem utilizadas na tomada de decisão [11]. Dentre os principais métodos utilizados, têm-se [12]: a classificação, a regressão, o agrupamento e a associação. Nesta tese será dado enfoque ao método de classificação.

O método de classificação permite obter padrões por meio da construção de classificadores, em um processo de indução, tendo como base um conjunto de instâncias (exemplos) de treinamento em diferentes classes, levando em consideração as propriedades de seus atributos. O modelo criado deverá ter habilidade para classificar novas instâncias, ou seja, deve ser capaz de generalizar [11, 12].

Nesse contexto, como a detecção de intrusão envolve a discriminação de ataques de diversos tipos ou categorias, então pode ser vista como um problema de classificação, chamada de classificação multiclasses. Em geral, os modelos aplicados na construção de SDIs empregam um único classificador [13–17]. Contudo, a tarefa de classificar várias categorias de uma só vez possui algumas desvantagens, tais como ter maior predisposição a gerar falsos alarmes, uma vez que o classificador deve ser capaz de separar os dados em um número maior categorias,

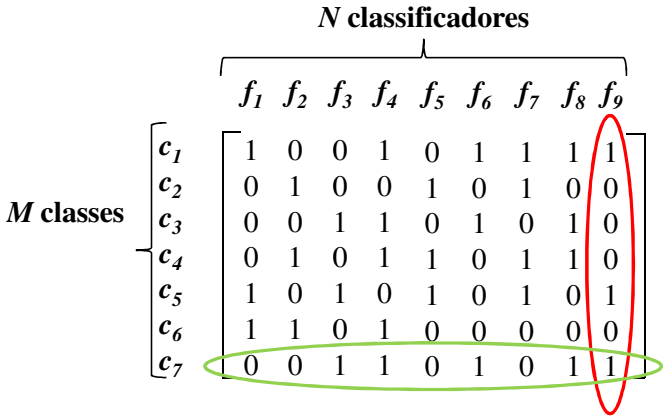
e ser mais complexa do que em sistemas que apenas classificam conexões como normais ou suspeitas (classificação binária).

Uma solução adotada em problemas multiclasses, conhecida como estratégia decomposicional, tem atraído a atenção de diversos pesquisadores, devido a fatores, tais como: permitir a redução da probabilidade de erro de classificação e possibilitar a diminuição da complexidade envolvida na separação das classes [18–23].

Assim sendo, dado um problema multiclasses, definido por um conjunto de τ instâncias de treinamento, $T = \{(\vec{x}_i, y_i)\}_{i=1}^{\tau}$, em que $\vec{x}_i$ é um vetor de atributos, $y_i \in \{c_1, c_2, ..., c_M\}$ e M é o número de classes do problema multiclasses ($M > 2$). O emprego de estratégias decomposicionais envolve basicamente a realização de dois procedimentos. No primeiro, o problema multiclasses é decomposto em N subproblemas, determinando, nesta fase, os classificadores (ou modelos de decisão) $f_1, f_2, ...f_N$ a serem utilizados na solução. Em seguida, é realizado o procedimento de decodificação ou reconstrução, que determina a forma como as saídas dos classificadores são combinadas para predizer a classe de uma instância.

Há diversas estratégias decomposicionais na literatura [20–23], sendo que algumas abordagens podem ser representadas por uma matriz $C_{M\times N}$, aqui referenciada como matriz código (não confundir com matriz geradora do código, definida no Apêndice D), em que M é o número de classes do problema e N o número de classificadores. Conforme ilustrado na Figura 1.2, cada linha dessa matriz representa um vetor de hipóteses que está associado a uma única classe. As suas colunas correspondem às hipóteses dadas a todas as classes por cada classificador $f_j, 1 \leq j \leq N$ gerado na fase de decomposição.

Figura 1.2 Exemplo ilustrativo de uma matriz código $C_{M\times N}$



Na fase de classificação, uma dada instância de teste $\vec{x}$ é aplicada a cada modelo de decisão criado na fase de decomposição, dando origem a N hipóteses distintas. O vetor de hipóteses gerado pelos classificadores é definido como $\mathbf{F}(\mathbf{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_N(\vec{x}))$. Supondo que as linhas da matriz código $C_{M\times N}$ sejam representadas por vetores $\mathbf{C}_\ell, 1 \leq \ell \leq M$, então o vetor de hipóteses $\mathbf{F}(\mathbf{x})$ será comparado com as linhas $\mathbf{C}_\ell$ da matriz $C_{M\times N}$ e aquela que for

mais próxima de $\mathbf{F}(\mathbf{x})$, de acordo com uma dada medida de distância d (por exemplo, a distância de *Hamming* [24], descrita no Apêndice D), será atribuída como a classe da instância $\vec{x}$. Em outras palavras, a classe predita é aquela que minimiza $d(\mathbf{C}_\ell, \mathbf{F}(\mathbf{x}))$.

Dentre as estratégias que utilizam uma matriz código $C_{M\times N}$, as mais comuns encontradas na literatura e que serão abordadas com maior detalhamento no Capítulo 4, são:

- Um-Contra-Todos: consiste na decomposição do problema original em $N = M$ subproblemas. Cada subproblema é definido por uma das M classes do problema comparando-a com as demais [20];
- Todos-Contra-Todos: consiste em criar $N = \frac{M(M-1)}{2}$ subuproblemas com todas as combinações possíveis das classes do problema original, juntado-as duas a duas [21, 22];
- Códigos Corretores de Erros: consiste em associar uma palavra código a uma classe do problema [23, 25–34].

Analisando as estratégias decomposicionais um-contra-todos, todos-contra-todos e baseadas em códigos corretores de erro, observa-se que a abordagem um-contra-todos utiliza somente os classificadores que identificam uma classe dentre as demais. A estratégia decomposicional todos-contra-todos, por sua vez, usa somente classificadores que distinguem pares de classes. Por fim, na estratégia baseada em códigos corretores de erros, que serve de inspiração para este trabalho de tese, tem-se maior flexibilidade na escolha dos classificadores, dentro dos limites permitidos ($\lceil \log_2 M \rceil \leq N \leq 2^{M-1} - 1$) [18], podendo envolver todas as classes simultaneamente.

Uma crítica comum às estratégias decomposicionais ora comentadas é que, na maioria das pesquisas realizadas, a decomposição do problema multiclasses é realizada a *priori*, sem levar em consideração as propriedades e as características de cada aplicação [19, 35]. Ademais, dependendo do domínio do problema e, em especial, no contexto da detecção de intrusão, essas estratégias tornam-se ineficientes, pois atualmente os ataques são polifórmicos [36], necessitando que mais de um vetor de hipóteses $\mathbf{C}_\ell$ seja associado a cada tipo de ataque. Em razão disso, utiliza-se neste trabalho, ao invés de uma matriz $C_{M\times N}$, uma tabela de palavras código $T_{w\times n}$, em que w é o número de palavras código que define um conjunto de classes, podendo ter mais de uma palavra código para a mesma classe e n é o número de símbolos da palavra código.

A ideia de usar códigos corretores de erros [24] para identificar intrusões em redes de computadores surgiu bem antes de ser realizada a pesquisa bibliográfica sobre o tema abordado, visto que dentro do limite $t \leq \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$ de correção de erros do código utilizado para esta finalidade, em que d_{min} é a distância mínima do código (ver Apêndice D), uma dada palavra recebida pode ser decodificada, por meio de um algoritmo de decodificação algébrica [24, 37], para uma palavra código que corresponde a uma assinatura de intrusão.

Entretanto, a análise das estratégias decomposicionais disponíveis na literatura científica reforça a motivação para este trabalho de tese que apresenta, como principal contribuição, uma

nova estratégia de classificação multiclasses baseada em códigos corretores de erros do tipo BCH (binário) e Reed-Solomon (BCH não binário) [24,37] aplicadas à detecção de intrusos em redes de computadores.

Como projetar a tabela de palavras código $T_{w\times n}$ é indubitavelmente uma questão importante, definir quais as regras de decisão para os classificadores, ou seja, as colunas da tabela e ao mesmo tempo garantir uma distância mínima entre as linhas suficiente para se obter a capacidade de correção de erros desejada, torna o projeto analítico da tabela bastante complexo.

Nesse contexto, o presente trabalho apresenta, como uma das contribuições, uma solução, baseada em uma variante de algoritmo genético (AG) [28], para realizar a busca por palavras código que possam ser utilizadas na identificação de tipos ataques e permitindo que variações deste padrão também possam ser identificados sem a necessidade imediata de atualização do sistema de detecção de intrusão.

Diferente das estratégias decomposicionais estudadas [18, 20–23, 25–34, 38], neste trabalho de tese, os classificadores $f_j, 1 \leq j \leq N$ serão substituídos por N detectores, aqui denominados de detectores de rede e de host (ver definição no Capítulo 2). Os detectores são responsáveis por monitorar informações (aqui denominados de atributos $A_j, 1 \leq j \leq N$) contidas nos pacotes que trafegam pela rede e/ou informações armazenadas nos *logs* de auditoria dos servidores e emitir um bit para cada atributo observado, com base em sua regra de decisão. Por exemplo, um dado detector sinaliza com *bit* '1' se o pacote é TCP (*Transmission Control Protocol*) [39] (ver Apêndice G) ou '0', caso contrário. Cada bit emitido por cada um dos N detectores, é utilizado para compor os símbolos da palavra recebida.

Ainda neste trabalho de tese, tenta-se minimizar também o número de atributos que serão observados pelos detectores, uma vez que existe uma relação direta entre o número de atributos e o tamanho da palavra código. Desta forma, definir qual subconjunto de todos os atributos possíveis de um tráfego de rede e *log* de servidores consegue predizer melhor atividades intrusivas é uma tarefa que deve ser feita com muito critério, pois, apesar de ser a situação ideal, observar todos os atributos não é tecnicamente viável.

Nesta perspectiva, a seleção do conjunto de atributos contribui fortemente para o sucesso do modelo de classificação, e, sendo mais uma contribuição desta tese, apresenta-se critérios de seleção dos atributos que serão monitorados pelos detectores, baseados no algoritmo de árvore de decisão C4.5, envolvendo as medidas de informação de Rényi [40] e Tsallis [41, 42]. Outrossim, por meio destes critérios de seleção, tem-se também a definição do tamanho da palavra código a ser utilizada no projeto da tabela $T_{w\times n}$.

1.2 Hipótese

Com base no que foi exposto anteriormente, o problema de detectar intrusão, visto aqui como um problema de classificação multiclasses, resolvido a partir da aplicação de estratégias

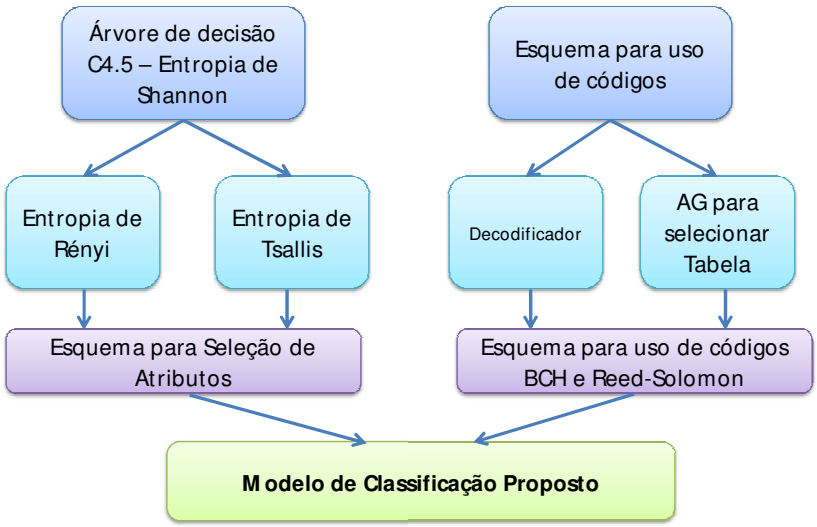
baseadas nos códigos corretores de erros [24], pode envolver três questões, de forma geral, as quais serão abordadas nesta tese:

- Como selecionar as palavras código que serão utilizadas como assinaturas de ataques?
- Quais atributos devem ser utilizados como pontos de observação pelos detectores?
- Qual estratégia de decodificação a ser adotada?

Observando essas questões de pesquisa, consideram-se nesta tese as seguintes hipóteses, ilustradas na Figura 1.3:

- É possível utilizar a teoria da codificação para a determinação de palavras código adequadas ao problema de detectar intrusões em uma rede de computadores. Nesta perspectiva, para cada tipo de ataque serão associadas palavras código pertencentes a um código do tipo BCH (binário) e Reed-Solomon (código BCH não binário) [24, 37];
- Pode-se utilizar uma variante de algoritmo genético para garantir que as palavras código pertençam ao código BCH escolhido e, ao mesmo tempo, selecionar as que melhor representem uma assinatura de intrusão;
- Consegue-se selecionar os atributos a serem monitorados pelos detectores por meio do uso de árvores de decisão C4.5 baseadas em medidas de informação de Rényi [40] e Tsallis [41, 42];
- Pode-se fazer uso de decodificação algébrica, baseado nos conceitos de decodificação tradicional [43] e na decodificação por lista [44, 45].

Figura 1.3 Esquema reunindo as etapas para a construção do modelo proposto.



Para efetuar uma validação das hipóteses desta tese, foi traçado um conjunto de objetivos, os quais são descritos na Seção 1.3.2.

1.3 Objetivos

Considerando as questões de pesquisa descritas, apresentam-se, nesta seção, os objetivos desta tese.

1.3.1 Objetivo Geral

O objetivo geral deste trabalho de tese é explorar as características e princípios dos códigos corretores de erros e medidas de informação para construir um esquema de classificação multiclasses aplicado à segurança computacional. Esse esquema deve possibilitar a detecção de ataques conhecidos e de ataques desconhecidos por intermédio de assinaturas de intrusão na forma de palavras código.

1.3.2 Objetivos Específicos

Este trabalho tem como objetivos específicos:

- Determinar quais atributos serão observados pelos detectores, por meio do uso de medidas de informação;
- Encontrar qual o melhor tamanho da palavra código de um código BCH (binário e não binário) a ser utilizado, dado o problema de classificação com M classes;
- Realizar a busca por uma tabela de palavras código adequada ao problema de classificação de tipos de ataques em redes de computadores;
- Validar o uso de medidas de informação, códigos corretores de erros e decodificação algébrica tradicional e decodificação por lista para o problema proposto.

1.4 Organização do Documento

A redação deste trabalho está organizada em duas partes maiores. A Parte I compreende a realização da revisão bibliográfica, a definição e aplicação de conceitos que direcionaram as pesquisas. Esta parte engloba três capítulos:

- O Capítulo 2 aborda os conceitos relativos à detecção de intrusão, apresentando vários modelos utilizados.
- No Capítulo 3, são apresentados os conceitos básicos sobre classificação de dados, exemplificado por meio do estudo dos algoritmos de árvore de decisão ID3 e C4.5 que usam medidas de informação de Shannon. Por fim, é apresentado o uso de medidas de informação de Rényi [40] e Tsallis [41,42] aplicadas na construção de árvores de decisão C4.5, como parte integrante deste trabalho de tese.

- No Capítulo 4, são apresentadas estratégias adotadas na solução de problemas multiclasses, com ênfase no uso de códigos corretores de erros, e que servem de base para a escolha da estratégia a ser utilizada nesta tese.

Após conhecidas as diversas abordagens e limitações existentes nas pesquisas envolvendo detecção de intrusos e classificação de dados, e também conhecendo algumas aplicações de códigos corretores de erros na solução de problemas de classificação multiclasses, a Parte II faz a descrição do esquema proposto, a descrição dos algoritmos que foram implementados, alguns testes realizados e os resultados obtidos. Os seguintes capítulos são englobados nesta parte:

- No Capítulo 5, é descrito um esquema de uso de códigos corretores de erros e medidas de informação aplicados à solução de problemas multiclasses, em especial a detecção de intrusão. Outrossim, são apresentados esquemas de seleção de atributos, usando árvore de decisão C4.5 baseada em medidas de informação de Rényi [40] e Tsallis [41, 42], bem como o esquema adotado para realizar a busca da tabela de palavras código, adequada ao problema de classificação de tipos de ataques em redes de computadores, usando uma variante de algoritmo genético. Por fim, é apresentado o esquema de decodificação utilizado no modelo.

- Alguns resultados selecionados da aplicação do esquema proposto são apresentados e analisados no Capítulo 6.

- No Capítulo 7, são apresentadas as conclusões e as perspectivas futuras advindas deste trabalho.

CAPÍTULO 2

Detecção de Intrusão em Redes de Computadores

Neste capítulo, são apresentados os principais conceitos e classificações de sistemas de detecção de intrusão em redes de computadores, destacando suas vantagens, limitações, características, seus métodos de análise e tempo de coleta. Por fim, são apresentados alguns métodos alternativos de detecção de intrusos e que servem de inspiração para este trabalho de tese.

2.1 Conceitos sobre Detecção de Intrusão

Para apresentar e discutir os aspectos de segurança computacional relacionados à detecção de intrusão, faz-se necessária a utilização de alguns termos comuns nessa área. Os termos mais relevantes para este trabalho e seus significados são descritos no decorrer desta seção.

Uma intrusão pode ser definida como sendo um conjunto de ações que tentam comprometer a integridade, confidencialidade ou a disponibilidade de recursos em um sistema computacional. A integridade demanda que a informação seja protegida contra modificações indevidas. A confidencialidade é a propriedade que busca garantir que a informação seja restrita somente àqueles usuários (indivíduos, entidades ou processos) com a devida permissão de acesso. A disponibilidade requer que a informação e os recursos do sistema estejam acessíveis quando requeridos por usuários autorizados [8–10].

A política de segurança é o documento formal que define, por meio de procedimentos e normas, quais atividades são permitidas na rede das organizações ou em servidores particulares. Um ataque compreende uma atividade maliciosa que procura violar a política de segurança. Uma intrusão também pode ser entendida como um ataque bem sucedido e pode ocorrer a partir da exploração de uma variedade de vulnerabilidades em um sistema ou rede de computadores, tais como erros de software e configurações impróprias ou simplesmente por usuários

não autorizados tentando acessar um sistema por meio da rede (externa ou interna). Pode-se definir vulnerabilidade como uma fraqueza no sistema computacional que pode ser explorada de modo a violar a política de segurança em questão [7].

A detecção de intrusão é o processo de monitoramento das atividades ocorridas em um sistema ou rede de computadores, analisando-as de forma a identificar possíveis violações da política de segurança [7]. A identificação de um ataque corresponde à detecção propriamente dita.

2.2 Sistema de Detecção de Intrusão

Um sistema de detecção de intrusão (SDI) é um produto baseado em *software* ou *hardware* que visa automatizar o processo de detecção de intrusão [7–10].

Em geral, os SDI usam um conjunto de características (aqui referenciados como atributos) extraídas dos pacotes que estão trafegando na rede ou extraídas do registro de uma atividade no *log* do sistema para detectar uma possível atividade indesejada ou desautorizada, que poderá resultar em um alerta. Alerta é dito como sendo uma mensagem de que uma atividade suspeita foi detectada, contendo as especificações das ocorrências (origem, destino, porta, serviço, informação de usuários, entre outros) [46].

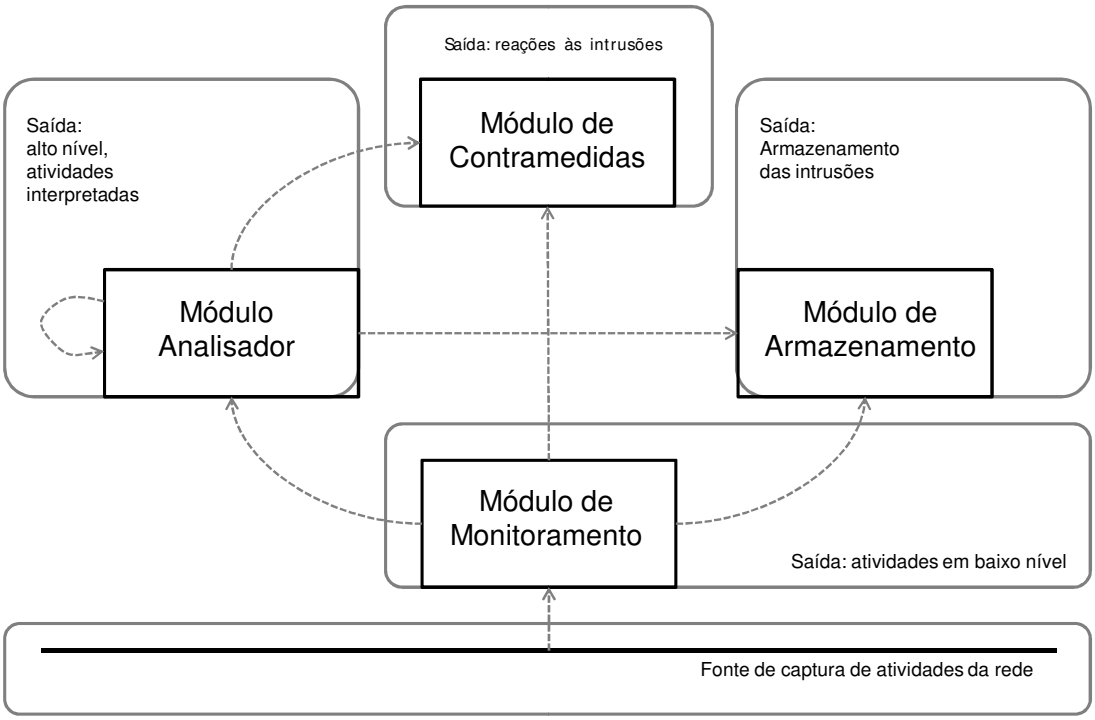
O conceito de sistema de detecção de intrusão foi proposto pela primeira vez por James Anderson [47], sendo difundido somente em 1987, com a publicação do artigo intitulado *Um Modelo de Detecção de Intrusão*, elaborado por Dorothy Denning [48]. Nos anos seguintes, outros modelos foram apresentados na tentativa de aumentar a segurança de um computador ou rede de computadores, dentre eles, os baseados nos métodos de classificação têm sido aplicados com sucesso [13–17,32]. Alguns desses modelos serão apresentados no decorrer deste capítulo.

Um modelo que representa o funcionamento de um SDI, por intermédio de um conjunto de componentes, por fluxo de informações e pelas suas funcionalidades básicas, é o *Common Intrusion Detection Framework* (CIDF) [49], conforme ilustrado na Figura 2.1.

Com base na Figura 2.1, o módulo de monitoramento é composto por detectores, cuja responsabilidade é observar o tráfego de rede e/ou registros de ocorrência nos arquivos de auditoria de um sistema computacional e enviar informações a respeito do que foi monitorado para os demais componentes do sistema. Os detectores, no contexto deste trabalho, são dispositivos de software chamados de detectores de rede (ou *sniffers*) quando realizam o monitoramento de pacotes em um segmento de rede e chamados de detectores de *host* quando coletam informações em arquivos de auditoria em computadores ou em servidores críticos, tais como, atividades suspeitas em *logon*, manipulação (criação, alteração e deleção) de grupos, usuários e objetos, privilégios de processos e usuários e uso de recursos do sistema monitorado, dentre outros.

Os detectores podem coletar um conjunto de atributos específicos da rede e dos servidores monitorados e também podem ser dotados de diferentes funcionalidades, tais como pos-

Figura 2.1 Relacionamento entre os componentes do modelo CIDF [50], adaptado pela autora



suir a capacidade de realizar cálculos, como por exemplo: calcular a taxa de conexões, tendo o mesmo serviço vindo de diferentes *hosts*; a contagem de conexões, tendo o mesmo *host* destino e mesmo serviço e emitir o resultado como informação a ser analisada pelo sistema de detecção de intrusão. O conjunto de atributos coletados pelos detectores é avaliado pelo módulo analisador do CDIF, com o objetivo de verificar se o mesmo corresponde, de fato, a uma atividade maliciosa ou trata-se apenas de uma atividade normal do sistema.

Seguindo o fluxo de informação do CDIF, caso um incidente de segurança seja identificado, o módulo de contramedidas é acionado para reagir às tentativas de invasão ou mau uso. Os tipos de reações a serem tomadas pelo módulo de resposta podem variar, de acordo com a segurança da corporação, tendo, obrigatoriamente, respostas padrão no caso de detecção de novos ataques. As reações são as mais diversas, podendo simplesmente enviar notificações ao administrador ou reagir de forma mais ativa, coletando-se informações adicionais do atacante, alterando as configurações do ambiente ou realizando-se ações diretas contra o intruso.

Por fim, as informações provenientes do módulo de monitoramento e dos mecanismos analisadores são registradas pelo módulo de armazenamento para serem úteis na identificação de tentativas de ataques provenientes de uma mesma origem ou domínio, ou simplesmente, para serem utilizadas em investigações futuras por parte dos administradores de segurança.

As vantagens mais comuns em se utilizar um SDI, segundo [7], são relatadas a seguir:

- Detectar ataques explorando vulnerabilidades que não podem ser corrigidas

Por meio de vulnerabilidades conhecidas, que em alguns casos não podem ser corrigidas, o invasor pode penetrar em muitas redes. A exemplo disso, têm-se os sistemas herdados, cujo sistema operacional pode ter sido descontinuado, impossibilitando que o mesmo seja atualizado.

Mesmo em sistemas passíveis de atualização, os administradores podem não dispor de tempo para instalar todas as atualizações necessárias em uma grande quantidade de computadores. Outra situação ocorre quando há a necessidade do uso de serviços e protocolos de rede que são notoriamente problemáticos e sujeitos a ataques. Embora, de maneira ideal, todas essas vulnerabilidades possam ser sanadas, na prática isso nem sempre é possível. Assim, uma boa solução pode ser o uso de SDI para detectar e, possivelmente, prevenir tentativas de ataque explorando tais vulnerabilidades.

- Prevenir que intrusos examinem a rede

Quando intrusos atacam uma determinada rede, eles geralmente o fazem em etapas. A primeira etapa é examinar o sistema em busca de alguma vulnerabilidade. Na ausência de um sistema de detecção de intrusão, o intruso se sente livre para fazer o seu trabalho sem riscos e pode descobrir com facilidade os pontos fracos do sistema e explorá-los. Com um SDI, embora o iminente intruso possa continuar a analisar o sistema, ele poderá ser detectado e os devidos alertas poderão ser dados.

- Documentar ameaças de invasão

É importante entender a frequência e as características dos ataques em um sistema computacional para medir o grau de hostilidade de um ambiente. Com isso, é possível tomar as medidas de segurança adequadas para proteger a rede. Um SDI pode quantificar, caracterizar e verificar as ameaças, internas ou externas, de intrusão e mau uso.

- Controlar qualidade

Com o uso contínuo de um SDI, padrões de utilização do sistema monitorado e problemas detectados tornam-se evidentes. Com essas informações, é possível tornar visíveis falhas no projeto, configuração e gerenciamento da segurança, proporcionado ao administrador a possibilidade de corrigi-las antes de ocorrer um incidente.

Existem várias abordagens que podem ser utilizadas no desenvolvimento dos componentes de um SDI. Essas abordagens determinam as características de cada SDI, influenciando na atuação desse SDI. A seguir, serão apresentadas as principais abordagens utilizadas no projeto desses componentes, classificadas quanto aos métodos de análises adotados, tipo de monitoramento, tempo entre a coleta e análise dos dados e o tipo de resposta a ser dada.

2.3 Métodos de Análise de Detecção de Intrusão

Para projetar um sistema de detecção de intrusão, uma variedade de métodos tem sido proposta nos últimos anos. Objetivando determinar a ocorrência de uma intrusão, existem dois métodos principais de análise de detecção de intrusão, conforme descritas em [9]: **a detecção por anomalia e a detecção por assinatura**.

2.3.1 O método de detecção de intrusão por anomalia

O método de detecção de intrusão por anomalia é baseado na observação de desvios de comportamentos esperados em atividades relevantes dos usuários ou processos do sistema monitorado. Os detectores de anomalia constroem perfis que representam o comportamento normal dos usuários, servidores ou conexões de rede. Estes perfis são gerados por intermédio de dados históricos coletados durante um período normal de operação. Os sistemas de detecção de intrusos por anomalia baseiam-se na premissa de que um ataque difere da atividade normal, podendo, dessa forma, ser identificado.

Contudo, nem sempre uma atividade anômala corresponde a uma atividade intrusiva. Por exemplo, um usuário que sempre trabalhou com determinada carga de uso de UCP (Unidade Central de Processamento) pode necessitar de mais poder de processamento e, mesmo assim, o usuário não pode ser considerado um intruso. Em [51], são apresentadas as seguintes definições de comportamento:

- **intrusivo e anômalo**: a intrusão coincide com a anormalidade. São os verdadeiros positivos.
- **não intrusivo e não anômalo**: não há intrusão, nem anormalidade. São os verdadeiros negativos.
- **intrusivo, mas não anômalo**: há intrusão sem que haja comportamento anormal. São os falsos negativos.
- **não intrusivo, mas anômalo**: não há intrusão, mas há comportamento anormal. São os falsos positivos.

A ocorrência de um grande número de falsos negativos e falsos positivos pode se tornar um problema dos SDI baseados em detecção por anomalia. Para reduzi-los, é preciso a definição dos limites que apontam a anormalidade.

Vários métodos foram propostos para implementar SDI baseados na detecção por anomalia [52–56], sendo os mais comuns: o método estatístico, o gerador de prognósticos de padrões e o uso de modelos baseados em técnicas de Aprendizado de Máquina.

- Método estatístico

No método estatístico [52], inicialmente, os perfis de comportamento dos usuários são criados. Isto é feito após um período de treinamento em que os usuários são monitorados em suas atividades cotidianas. O comportamento padrão é, então, mantido num perfil particular ou mesmo em perfis de grupos. À medida que o sistema continua a operar normalmente, o detector cria o perfil atual e em intervalos regulares, o compara com o perfil padrão que foi armazenado na fase de treinamento. Assim, se o desvio entre os dois for demasiadamente grande, pode-se inferir que há sinais de anormalidade, o que poderia implicar em uma provável intrusão.

Um assunto ainda em constante pesquisa, relacionado com o modelo estatístico na detecção por anomalia, é a escolha do que deve ser monitorado [52]. As métricas que afetam o perfil de comportamento podem ser as mais diversas, como: quantidade de arquivos abertos, tempo de uso da Unidade Central de Processamento (UCP), número de conexões de rede em um período, operações de E/S. Não é conhecido, de fato, qual subconjunto de todas as medidas possíveis, consegue predizer melhor as atividades intrusivas. Portanto, uma análise criteriosa do conjunto de medidas utilizadas pode contribuir bastante para o sucesso do método estatístico.

As principais vantagens deste método são que ele pode ser baseado em técnicas estatísticas bem conhecidas e aprender o comportamento dos usuários, adequando-se a mudanças no perfil. O uso de medidas estatísticas na detecção também pode revelar atividades suspeitas, levando à descoberta de vulnerabilidades.

É justamente devido a essa adaptabilidade, que ocorre a principal fragilidade do método estatístico, já que os sistemas baseados em estatística podem gradualmente ser treinados por intrusos de tal forma que eventualmente as intrusões sejam consideradas como normais. Outra desvantagem desse método está relacionada à dificuldade encontrada para efetuar a análise em tempo real, ocasionada pela quantidade de memória requerida para armazenar os perfis de comportamento.

Além disso, a análise estatística não permite estabelecer uma ordem sequencial de atividades, mas somente comparar a atividade atual com perfis construídos. Um problema encontrado no método estatístico é a dificuldade em se determinar um grau de tolerância adequado para que um perfil não seja considerado intrusivo (*threshold*). Se ele for muito alto ou muito baixo, então poderá haver um grande número de falsos negativos ou falsos positivos, respectivamente, levando alguns usuários a ignorar ou desabilitar o SDI.

Alguns problemas associados ao método estatístico foram remediados por outras abordagens, incluindo o método conhecido como gerador de prognósticos de padrões, que leva em consideração a ordem das atividade passadas em sua análise.

- Método gerador de prognósticos de padrões

No gerador de prognósticos de padrões, tenta-se predizer as atividades futuras com base em atividades que já ocorreram, conforme descrito em [53]. Portanto, pode-se ter uma regra do

tipo: $E1 \rightarrow E2 \rightarrow (E3 = 80\%, E4 = 15\%, E5 = 5\%)$. Isto significa que dadas as atividades $E1$ e $E2$, com $E2$ ocorrendo depois de $E1$ (lado esquerdo da regra), existe 80% de chance de que a atividade $E3$ ocorra em seguida, 15% de chance que $E4$ seja a próxima e 5% que $E5$ ocorra (lado direito da regra). Assim, na ocorrência de uma determinada atividade, ou uma série de atividades, é possível predizer quais as possíveis atividades subsequentes.

Da mesma forma que no método estatístico, aqui é necessário primeiro haver um período de treinamento. Por meio do acompanhamento dos usuários em suas atividades diárias, as regras temporais que caracterizam os padrões de comportamento podem ser geradas. Essas regras se modificam ao longo do período de aprendizado. Porém, somente são mantidas no sistema aquelas que apresentarem um desempenho ótimo no prognóstico, ou seja, só as regras que conseguem predizer a próxima atividade de forma correta são aceitas.

Há algumas vantagens em utilizar o método gerador de prognósticos de padrões, apresentadas a seguir:

- padrões sequenciais, baseados em regras, podem detectar atividades anômalas que são difíceis nos outros métodos;
- sistemas construídos usando esse modelo são bastante adaptativos a mudanças. Isto se deve ao fato de que padrões de má qualidade são continuamente eliminados, conservando-se os padrões de alta qualidade;
- atividades anômalas podem ser detectadas muito rapidamente a partir do recebimento dos registros de auditoria.

Um problema do método gerador de prognósticos de padrões, é que determinados padrões de comportamento podem não ser detectados como anômalos, simplesmente porque não é possível combiná-los em nenhuma regra. Essa deficiência pode ser parcialmente resolvida acusando-se todas as atividades desconhecidas como intrusivas, correndo o risco de aumentar o número de falsos positivos ou como não intrusivas, aumentando-se a probabilidade de falsos negativos.

Como uma possível solução para este problema, o uso de Aprendizado de Máquina tem sido pesquisado [54–56].

- Aprendizado de Máquina

A ideia básica para se obter um modelo baseado em técnicas de aprendizado de máquina, é submetê-lo a um processo de treinamento com o conjunto das ações representativas. O modelo gerado deverá ser capaz de predizer a próxima ação do usuário, dado o conjunto de suas ações anteriores [57].

Qualquer ação que não corresponda à realidade, pode medir um desvio no perfil normal do usuário. Algumas vantagens são que as técnicas de Aprendizado de Máquina têm a capacidade de inferir dados com ruídos. O sucesso desta abordagem não depende de nenhuma

suposição sobre a natureza dos dados e são mais fáceis de se modificar para uma nova comunidade de usuários.

Entretanto, o uso de modelos baseados em técnicas de Aprendizado de Máquina tem alguns problemas, pois, ao se dispor de um pequeno conjunto de informações para o treinamento, a tendência é que aumente a incidência de falsos positivos gerados pelo modelo. Da mesma forma, um conjunto grande de treinamento, mas com informações irrelevantes, irá aumentar as chances do modelo gerar mais falsos negativos.

Em síntese, o ponto forte do método de detecção por anomalia é a capacidade de reconhecer tipos de ataques que não foram previamente conhecidos, baseado apenas nos desvios de comportamento, enquanto que os pontos fracos podem ser o grande número de alarmes falsos gerados para uma rede treinada insuficientemente e a necessidade de um grande período de treinamento para se construir os perfis dos usuários.

2.3.2 Método de detecção de intrusão baseada em assinatura

A detecção de intrusão baseada em assinatura está relacionada à identificação de comportamentos intrusivos correspondentes à exploração de vulnerabilidades conhecidas, comumente chamadas de assinaturas de intrusão. Os detectores de assinatura analisam a atividade do sistema, observando o tráfego de rede, sequências de chamadas de sistema e outros atributos ou características que sejam semelhantes a um padrão pré-determinado que descreva uma intrusão conhecida. Por exemplo, 10 falhas de *logon* em 2 segundos pode indicar um ataque do tipo força bruta ou do tipo de adivinhação de senhas.

As assinaturas não servem apenas para detectar intrusões consumadas, elas são úteis, também, para identificar tentativas de ataques. Assim, quando um conjunto de ações satisfaz apenas parcialmente uma assinatura, pode ser um indicativo da ocorrência de uma intrusão.

São conhecidas várias abordagens para implementar SDI baseados em detecção por assinatura. As mais comuns são: a utilização de sistemas especialistas; a detecção por modelo; a análise de estados de transição e o uso de modelos baseados em técnicas de Aprendizado de Máquina.

- Sistemas Especialistas

Os sistemas especialistas são sistemas inteligentes que capturam o conhecimento de especialistas humanos em domínios limitados, geralmente na forma de regras do tipo SE-ENTÃO. A combinação entre as fases de aquisição dos dados e a ação propriamente dita é feita de acordo com a comparação entre as atividades atuais do sistema, capturadas por meio dos mecanismos de auditoria do sistema operacional e as regras já estabelecidas pelo engenheiro de conhecimento. As condições SE do sistema especialista codificam o conhecimento sobre os ataques, declarando suas características. Assim, satisfeitas tais condições, as ações ENTÃO são executadas [58].

Como vantagens da utilização de sistemas especialistas, tem-se que: com um conjunto de regras bem definido, é possível se ter um sistema ágil e com pouca sobrecarga; tem vários critérios podem ser analisados para se identificar um ataque e ainda manter o sistema especialista eficiente e sem sobrecarga; teoricamente, é possível deduzir simbolicamente a ocorrência de uma intrusão baseada apenas nos dados recebidos.

O uso de sistemas especialistas apresenta diversas desvantagens. Somente ataques conhecidos e bem definidos podem ser detectados, o que exige uma constante atualização da base de regras. Segundo [59], os sistemas especialistas são criados à semelhança de quem os programou, sendo limitado pelo conhecimento do programador ou do responsável pela segurança. Outra desvantagem é a dificuldade que há em tirar conclusões sobre situações consideradas ambíguas, devido à existência de conflitos nos fatos assumidos pelos antecedentes das regras.

- Detecção por modelos

O objetivo da detecção de intrusão por modelos, proposta por [60], é a construção de um banco de dados de cenários que representam uma sequência de comportamentos característicos de intrusões. Assim, intrusões podem ser inferidas por meio do monitoramento de determinadas atividades. Se um conjunto de ações ocorrer da forma descrita nos cenários, então é possível detectar a tentativa de ataque. Dessa forma, o sucesso do modelo é dependente da definição dos cenários de intrusão [60].

A detecção por modelo é baseada em três módulos, conforme descrito por [60]. O ANTICIPATOR busca, no banco de dados de cenários, aquele que melhor expressa as atividades atuais do sistema. Na ocorrência de um subconjunto de ações que representam um dado cenário de ataque, o ANTICIPADOR gera os próximos passos da possível tentativa de ataque, de acordo com o cenário escolhido, a ser verificado nas trilhas de auditoria do sistema. A sequência hipotética, sugerida pelo ANTICIPATOR, é transmitida ao PLANNER, encarregado de traduzi-la para um formato compatível com sua provável representação no sistema de auditoria. Por fim, cabe ao INTERPRETER procurar, nos registros de auditoria do sistema, somente as sequências de comportamento prováveis, para confirmar ou rejeitar o cenário de ataque.

A vantagem dos SDI baseados no método de detecção por modelo é supor as ações futuras que o invasor irá realizar e, dessa forma, permitir a emissão de conclusões, ainda que parciais, sobre situações suspeitas, por meio do raciocínio matemático (na presença de incertezas). É também possível reduzir a quantidade de processamento exigida por cada registro auditor, por meio de um monitoramento superficial, buscando somente aquelas ações necessárias para a confirmação do cenário de intrusão.

A desvantagem dos SDI baseados neste método é de não detectar situações que não tenham sido especificamente evidenciadas, ou que não se aproximem de nenhum cenário existente.

- Análise de estado de transição

Segundo [61], a detecção por assinatura baseada na análise de estado de transição considera que um sistema monitorado pode ser representado como uma sequência de transição de estados. Uma transição acontece quando alguma condição é tida como verdadeira (por exemplo, a abertura de um determinado arquivo). Sucessivos estados estão conectados por arcos que representam as ações necessárias para a mudança de estado ocorrer (diagrama de transição de estados).

Os tipos de ações permitidos são descritos no modelo e não precisam ter uma correspondência um a um com os registros de auditoria. Para o estado final (comprometido) ser alcançado, diversas condições precisam ser satisfeitas. Assim, se todas as condições intermediárias forem verdadeiras, então é quase certo que uma tentativa de intrusão está ocorrendo. Entretanto, se alguma dessas condições é falsa, a probabilidade de ser uma intrusão diminui. Algumas vantagens dessa abordagem é poder detectar ataques cooperativos, mesmo aqueles que variam entre várias sessões de usuário e prever o acontecimento de ataques iminentes, podendo-se, com isso, tomar medidas preventivas.

Como desvantagens têm-se que, nesta abordagem os padrões de ataques são especificados somente por uma sequência de ações, ao invés de formas mais complexas; não há um mecanismo eficiente que reconheça as tentativas parciais de ataques; não é possível a detecção de alguns tipos de ataques, tais como negação de serviço (ver Apêndice H) e variações do uso normal do sistema protegido, pois estes itens não são gravados pelos mecanismos de *log* e, portanto, não podem ser representados por diagramas de transição de estado.

- Aprendizado de Máquina

Assim como na detecção de intrusos por anomalia, as técnicas de Aprendizado de Máquina têm sido utilizadas para reconhecer padrões de ataques ou assinaturas [62, 63]. Mas, diferentemente das diversas abordagens analisadas anteriormente, os modelos baseados em técnicas de Aprendizado de Máquina têm a capacidade de generalização. Por exemplo, dado um ataque que não seja exatamente igual à sua descrição armazenada, o modelo pode reconhecê-lo simplesmente pelas semelhanças existentes com a descrição.

Contudo, para que um modelo baseado em técnicas de Aprendizado de Máquina possa identificar tentativas de ataques de forma eficaz e eficiente, é necessário que haja um treinamento adequado com dados e métodos apropriados. Em geral, com o objetivo de obter resultados mais precisos na detecção de intrusão, é condição necessária haver uma grande e diversificada massa de dados de treinamento, com inúmeras sequências de ataques imersas em atividades normais do sistema. Uma base de dados com estas características não é simples de ser obtida e consiste no grande entrave na utilização desta abordagem no desenvolvimento de um SDI.

O método de detecção por assinatura tem sido amplamente utilizado no desenvolvimento de sistemas de detecção de intrusos, devido à facilidade de configuração, custo computacional

reduzido e pequeno comprometimento do desempenho comparados ao método de detecção por anomalia [62,64]. Quanto ao número de falsos positivos, a literatura científica disponível mostra que os sistemas de detecção de intrusos por assinatura geram menos alarmes falsos e conseguem detectar ataques conhecidos com eficiência e alto nível de precisão [9].

A limitação principal da abordagem do método de detecção por assinatura é que somente ataques conhecidos são detectados e, dessa forma, este método pode ser ineficiente em identificar todas as variações possíveis de um mesmo tipo de ataque, pois estudos indicam que os intrusos usam diferentes abordagens para explorar a mesma vulnerabilidade [36], ou seja, o ataque apresenta-se diferente da assinatura conhecida, porém é funcionalmente equivalente [65]. Assim, novas assinaturas de intrusão devem ser constantemente adicionadas. Outra limitação dessa abordagem é que é necessário a definição do que precisa ser auditado para determinar uma assinatura de intrusão.

Todavia, o método de detecção por anomalia oferece a capacidade para resistir a ataques polimórficos no momento em que novos ataques são inseridos em um sistema, uma vez que possui a habilidade de identificar ataques desconhecidos. Portanto, conforme descrito anteriormente, os dois métodos possuem vantagens e desvantagens quanto à sua utilização. Consequentemente, uma combinação de ambos pode gerar um sistema de detecção de intrusos com maior capacidade para detectar atividades maliciosas em um ambiente computacional.

2.3.3 Categorias de sistemas de detecção de intrusão

Embora muitos sistemas de detecção sejam vistos conceitualmente como compostos por um detector, um analisador e uma interface de usuário, os tipos de dados examinados e gerados pelos SDI podem variar significantemente. Os sistemas de detecção de intrusos podem ser classificados em uma das seguintes categorias, de acordo com os tipos de dados que eles examinam [9]:

- **SDI baseado em rede**: detectam intrusos por intermédio da captura e análise de pacotes que trafegam na rede, podendo, com isso, monitorar vários *hosts* conectados ao mesmo segmento de rede.
- **SDI baseado em *host***: examina trilhas de auditoria e arquivos de *logs* das aplicações para monitorar atividades locais em um sistema de computador em particular, podendo detectar ataques que não são vistos pelo SDI baseado em rede.
- **SDI baseado em aplicação**: examina o comportamento de um programa de aplicação, geralmente na forma de arquivos de *log*. Estes SDI são um subconjunto dos SDI baseados em *host*.

As vantagens dos **SDI baseados em rede** são que: se localizados em pontos estratégicos, podem monitorar uma rede bem extensa; tem pouco impacto na rede monitorada, exigindo

um mínimo esforço de instalação, pois os detectores são dispositivos passivos (apenas capturam informações da rede monitorada); diversas técnicas podem ser usadas para camuflar os detectores de rede, deixando-os invisíveis aos intrusos; é possível monitorar ataques distribuídos à rede.

Entretanto, os SDI baseados em rede tem alguns pontos negativos, mostrados a seguir [7, 9]:

- em longos períodos de tráfego intenso na rede, os detectores podem não conseguir capturar todos os pacotes necessários para análise, podendo, dessa forma, não reconhecer tentativas de ataque;
- ainda não é possível inferir informação alguma sobre o conteúdo de pacotes criptografados. Enquanto as organizações optam pela criptografia para manter a confidencialidade das suas informações, os intrusos a usam para melhorar as chances de sucesso de seus ataques;
- os detectores de rede não conseguem operar bem com os *switches* de rede [39], pois estes geralmente não possuem portas de monitoramento universal, o que limita bastante a capacidade de captura de pacotes;
- a maioria dos SDI baseados em rede não indica se a intrusão foi bem sucedida, só há a indicação de que a tentativa de intrusão começou. Determinar se houve êxito ou não, é importante para o administrador de segurança.

Os **SDI baseados em *host*** operam analisando as atividades de algum dos computadores da rede em particular. Isto permite ao SDI coletar informações que refletem o que está acontecendo nos computadores de uma forma bastante precisa. Para tanto, é geralmente utilizado o mecanismo de auditoria do sistema operacional. Assim, o detector de *host* tem acesso aos diversos *logs* do sistema.

Dentre outras, as vantagens da abordagem baseada em *host*, são que o sistema pode: monitorar o acesso à informação em termos de quem acessou o quê; mapear atividades problemáticas com um usuário específico; rastrear mudanças de comportamento associadas com mau uso; operar em ambientes criptografados; distribuir a carga do monitoramento ao redor dos diversos computadores da rede.

Dentre as desvantagens encontram-se: a atividade da rede não é visível pelos detectores de *host* e, dessa forma, alguns tipos de ataques, não podem ser identificados; o acionamento dos mecanismos de auditoria acarreta uma sobrecarga no sistema; às vezes, é requerido um espaço em disco considerável para os registros; as vulnerabilidades dos sistemas operacionais podem ser usadas para corromper os detectores de *host*; os detectores de *host* são específicos por plataforma, o que acarreta em maior custo no desenvolvimento.

Os **SDI baseados em aplicação** são um subconjunto dos baseados em *host*, que analisam as atividades ocorridas nas aplicações. A maior fonte de informação são os arquivos de *log* das transações das aplicações.

A habilidade para interagir diretamente com a aplicação, por meio do conhecimento do domínio ou aplicação específica incluída na máquina em análise, permite que sejam detectados comportamentos suspeitos, oriundos de usuários tentando exceder o uso de seus privilégios.

Essa abordagem possui algumas vantagens, tais como: monitorar a interação entre usuários e aplicação, permitindo que sejam traçadas atividades desautorizadas para usuários individuais e trabalhar em ambientes criptografados, desde que a interação seja no ponto final da transação, em que as informações são apresentadas aos usuários na forma descriptografada.

As desvantagens são que pode ser mais vulnerável a ataques do que na abordagem baseada em *host*, pois os *logs* de aplicação são menos protegidos do que as trilhas de auditoria dos sistemas operacionais. Como as atividades monitoradas são no nível de usuário, um SDI baseado em aplicação não pode, por exemplo, detectar "cavalos de tróia", ou outro tipo de ataque à aplicação.

Portanto, com base nesta exposição, pode-se concluir que é desejável um SDI que combine as vantagens apresentadas pelas três abordagens, para se ter informações (a nível de rede, *host* e aplicação) suficientes para uma análise mais precisa.

2.3.4 Tempo de coleta e análise dos dados

Dentre as características dos métodos de análise e os tipos de dados examinados, deve-se considerar particularmente importante o tempo de coleta e análise dos dados [6, 8].

- Orientada a intervalo

Na abordagem *batch*, também conhecida como orientada a intervalo, os mecanismos de obtenção de dados guardam os registros em arquivos, que serão periodicamente analisados pelo SDI em busca de sinais de ataques ou mau uso. Desse modo, o fluxo de dados dos pontos de monitoramento não é continuamente analisado.

Este método é adequado para situações em que a ameaça de ataque é pequena e o interesse maior é saber quem são os intrusos e não tomar medidas imediatas contra o ataque. O modo *batch* requer menos carga de processamento, sendo ideal para instituições que dispõem de recursos limitados.

As desvantagens básicas são que os incidentes de segurança raramente são detectados em tempo hábil para que as devidas providências sejam executadas, maximizando os danos e pode haver a necessidade de grande espaço em disco para armazenar os arquivos gerados, principalmente em ambientes de rede.

- Em tempo real

Os métodos em tempo real garantem que a análise dos dados coletados será feita continuamente, de forma que o SDI possa agir com a rapidez necessária para frustrar uma tentativa de intrusão qualquer. Dependendo da velocidade da análise, o ataque pode ser detectado rápido o bastante, para que ele seja interrompido. É possível, também, coletar as informações necessárias prontamente para que o invasor seja logo penalizado.

No entanto, neste método, há um consumo de memória e demanda de processamento bem maior que no método *batch*. A configuração de sistemas de tempo real é crítica, pois qualquer assinatura mal caracterizada pode gerar uma enorme quantidade de falsos alarmes em um curto espaço de tempo.

2.4 Métodos Alternativos de Detecção de Intrusão

Por meio de uma outra visão do problema, algumas abordagens estão sendo aplicadas na área detecção de intrusão, dentre elas, as que serão descritas a seguir:

2.4.1 Abordagem baseada em agentes

Abordagens baseadas em agentes inteligentes são compostas por sistemas computacionais residentes em ambientes dinâmicos complexos, os quais percebem e atuam autonomamente neste ambiente e, ao fazê-lo, realizam um conjunto de objetivos e tarefas para os quais foram designados [46]. Num sistema multiagentes, os agentes precisam se comunicar, visando seus objetivos locais ou convergindo para um objetivo global.

Existem alguns fatores motivadores quanto ao uso de agentes para detectar intrusões, os quais venham garantir as características desejáveis em um SDI:

- os agentes podem ser adicionados e removidos do sistema sem precisar reiniciá-lo, tornando fácil a sua manutenção e atualização;
- os agentes podem ser treinados para identificar novos ataques;
- os agentes podem ser ativados e desativados dinamicamente para realizarem suas tarefas, proporcionando, com isso, melhor uso dos recursos do sistema;
- um agente pode ser configurado especificamente para as necessidade de um *host* ou uma rede, aumentando o poder de configuração do sistema;
- o uso de agentes garante maior tolerância a falhas do que em sistemas tradicionais. Uma vez que os sistemas multiagentes possuem uma estrutura descentralizada e dotada de alto grau de intercomunicação, estes sistemas apresentam maior flexibilidade, bem como, maior probabilidade de sobrevivência;

- agentes podem ser adicionados para aumentar a capacidade do SDI, permitindo que os mesmos operem em sistemas maiores.

Visando atingir essas características, o uso de agentes em sistemas de detecção de intrusão tem sido proposto por diversos grupos de pesquisa [46, 66, 67].

2.4.2 Abordagens baseadas no sistema imunológico

As primeiras pesquisas envolvendo imunologia e segurança computacional surgiram em 1994 com os trabalhos de [68, 69], desencadeando uma série de outros trabalhos, tais como os de [70, 71].

Com a mesma concepção dos agentes, o sistema imunológico, fundamental para a sobrevivência de um indivíduo, constitui um sistema distribuído em que um grande número de células e moléculas, com funções específicas e trabalhando de forma independente, age cooperativamente, utilizando mecanismos de comunicação ou sinalização para atuarem de forma eficiente na defesa do organismo contra microorganismos tipo vírus e bactérias.

É possível encontrar no sistema imunológico biológico um modelo que apresenta uma série de características desejáveis a um sistema de segurança. Dentre elas, pode-se destacar, segundo [72]:

- Unicidade: cada indivíduo possui seu próprio sistema imunológico, com suas capacidades e vulnerabilidades particulares;
- Reconhecimento de padrões internos e externos ao sistema: as células que não pertencem ao organismo são identificadas e eliminadas pelo sistema imunológico;
- Detecção de anomalia: o sistema imunológico pode detectar e reagir a agentes invasores que o organismo nunca foi exposto anteriormente;
- Detecção imperfeita (tolerância a ruídos): não é necessário um reconhecimento perfeito para que o sistema imunológico humano reaja contra um elemento causador de patologia;
- Diversidade: existe uma quantidade limitada de células e moléculas no sistema imunológico humano que são utilizadas para se obter o reconhecimento de um número praticamente ilimitado de elementos, incluindo aqueles sintetizados em laboratório;
- Aprendizagem por reforço: a cada encontro com o mesmo patógeno, o sistema imunológico melhora a qualidade de sua resposta; e
- Memória: os componentes do sistema imunológico humano bem sucedidos no reconhecimento e combate às patologias são armazenados para uma resposta futura mais intensa e efetiva.

Considerando essas características, Dasgupta [73] propôs uma arquitetura que serviu como base para a implementação do sistema de detecção de intrusão SANTA (*Security Agents for Network Traffic Analysis*) [74]. Outras pesquisas nessa linha podem ser referenciadas tais como o sistema adaptativo distribuído chamado *Artificial Immune System* (ARTIS), o qual foi aplicado na área de segurança de computadores sob a forma de um sistema de detecção de intrusos baseado em anomalia e assinatura denominado *Lightweight Intrusion Detection System* (LISYS) [75]. Além disso, de Paula [76] projetou o ADENOIDS para detectar ataques na camada de aplicação e automatizar a extração de assinaturas para ataques remotos do tipo *buffer overflow*.

Os sistemas imunológicos artificiais são um conjunto de sistemas inspirados no sistema imunológico humano, que podem ser aplicados a diversas áreas, como reconhecimento de padrões, classificação, otimização de problemas, detecção de anomalias, aprendizado de máquina, dentre outros. De um modo geral, os sistemas imunológicos artificiais podem ser classificados em três categorias principais: os inspirados na teoria de redes imunológicas, os que são baseados no princípio da seleção clonal e os que utilizam técnicas inspiradas no mecanismo de reconhecimento *próprio/nãopróprio*.

Exemplos de sistemas imunológicos artificiais aplicados na otimização, reconhecimento de padrões e aprendizado de máquina têm-se: o algoritmo de seleção clonal (CLONALG) [77], o sistema de classificação clonal (CSCA) [78], o sistema de reconhecimento imunológico artificial (AIRS) [79], a rede imunológica (aiNet) [72] e IMMUNOS-99 [80].

Um estudo comparativo sobre os algoritmos CLONALG, CSCA, AIRS e IMMUNOS-99 foi realizado como parte integrante desta tese e os resultados obtidos deste estudo foram publicados em [17] (encontrados nos anexos desta tese). Uma descrição sucinta de cada algoritmo pode ser encontrada no Apêndice J.

2.4.3 Mineração de dados

Considerando que na detecção de intrusão uma das dificuldades se dá ao grande volume de dados a serem analisados, faz-se necessário o uso de técnicas que possibilitem a descoberta de padrões de forma automatizada. Essas técnicas são conhecidas como descoberta de conhecimento em bases de dados ou simplesmente KDD (*Knowledge Discovery in Databases*) [12].

O processo de KDD é realizado por meio de vários métodos de extração de informações, conhecido como mineração de dados, principal fase do processo de KDD, responsável pela seleção dos métodos a serem utilizados para localizar padrões, que podem não estar aparentes em outros métodos de inspeção. Dentre os principais métodos, têm-se a classificação, regressão, agrupamento e associação [11].

A tarefa de classificação visa identificar a qual classe uma determinada instância pertence. A regressão (ou estimação) é similar à classificação, porém é usada quando a instância é identificada por um valor numérico e não um categórico. Assim, pode-se estimar o valor de

uma determinada variável, analisando-se os valores das demais. Por exemplo, um conjunto de instâncias contendo os valores mensais gastos por diversos tipos de consumidores e de acordo com os hábitos de cada um. Após ter analisado os dados, o modelo de regressão é capaz predizer qual será o valor gasto por um novo consumidor [12].

O agrupamento de dados visa identificar e aproximar as instâncias similares. Um agrupamento (ou *cluster*) é uma coleção de instâncias similares entre si, porém diferentes das outras instâncias nos demais agrupamentos. Esta tarefa difere da classificação, pois não necessita que as instâncias sejam previamente categorizadas (aprendizado não-supervisionado). Além disso, a tarefa de agrupamento não tem a pretensão de classificar, estimar ou predizer o valor de uma variável, ela apenas identifica os grupos de dados similares [81, 82].

Por fim, o processo de associação consiste em identificar quais atributos estão relacionados. Em geral, apresentam a forma: SE atributo X, ENTÃO atributo Y. Um exemplo de aplicação deste modelo pode ser na análise das chamadas "Cestas de Compras", em que são identificados quais produtos são levados juntos pelos consumidores [12].

Uma vez que é pretendido categorizar diversos tipos de ataques a uma redes de computadores, então, no contexto deste trabalho, será dado enfoque ao método de classificação. Dentre os algoritmos baseados na classificação de tipos de ataques em redes de computadores, tem-se as árvores de decisão [13–15], que se baseiam na estratégia de dividir para conquistar. Esses algoritmos procuram encontrar formas de dividir sucessivamente o universo em vários subconjuntos até que cada um deles contemple apenas uma classe ou até que uma das classes demonstre uma clara maioria, não justificando posteriores divisões.

Como parte integrante deste trabalho de tese, foi realizado um estudo comparativo entre o algoritmo de árvore de decisão C4.5 [83], que usa medidas de informação de Shannon [84] para a construção de árvores de decisão e os algoritmos CLONALG [77], CSCA [78], o AIRS [79] e o IMMUNOS-99 [80], inspirados no sistema imunológico humano, aplicados na detecção de intrusos visando encontrar alternativas mais eficientes na construção de sistemas que sejam tolerantes a intrusões.

Os resultados obtidos no estudo supracitado, que podem ser vistos no Apêndice J, apontam o algoritmo de árvore de decisão C4.5 obtendo o melhor desempenho na classificação de tipos de ataques. Este estudo comparativo serviu de inspiração para novas pesquisas utilizando árvore de decisão C4.5 e que serão abordadas nos capítulos seguintes.

2.5 Considerações Finais

Neste capítulo foram apresentados os principais conceitos relacionados à detecção de intrusão. Utilizando esses conceitos, foram discutidas as principais abordagens e pesquisas constantes no desenvolvimento de tecnologias aplicadas à detecção de intrusos em redes de computadores.

A detecção de intrusos é uma solução, que associada ao uso de outras tecnologias, tais como anti-vírus, *firewalls*, criptografia, busca aumentar o nível de segurança de uma rede privada, visto que, isoladamente, nenhuma tecnologia oferece um elevado grau de segurança, permitindo, simultaneamente, flexibilidade e liberdade no uso dos recursos protegidos [5].

Portanto, a continuidade dos projetos e a busca por métodos alternativos de segurança de redes têm sido uma preocupação constante dos pesquisadores. Neste contexto, o presente trabalho representa mais uma possibilidade de expansão do conhecimento teórico e prático de novas tecnologias aplicadas a essa área.

Como pretende-se aplicar códigos corretores de erros (ver Apêndice D) e medidas de informação ao problema de classificação de atividades ocorridas num sistema computacional como uma operação normal ou suspeita (indicando qual o tipo de ataque ocorrido), coletados em uma rede de computadores ou em um sistema em particular, no próximo capítulo serão apresentados, de uma forma resumida, os conceitos sobre classificação de dados, com ênfase nos modelos baseados em árvore de decisão, descrevendo os aspectos relevantes para esta tese.

CAPÍTULO 3

Técnicas e Algoritmos de Classificação

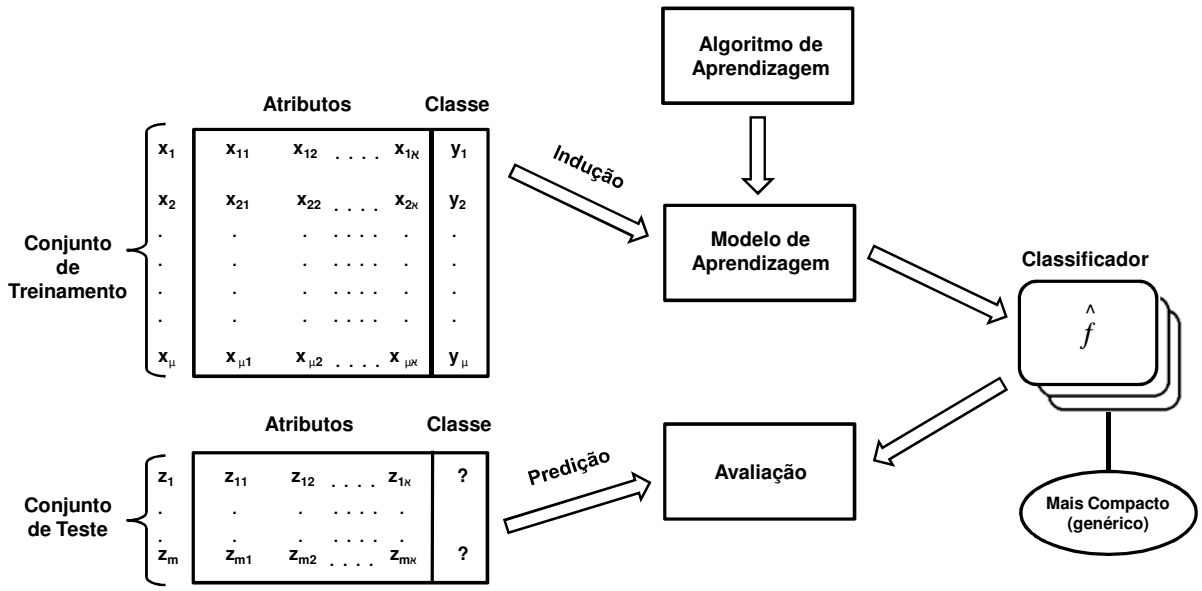
Uma vez descritos os conceitos e os métodos aplicados na detecção de intrusão, é possível buscar novos modelos que representem melhor as condições atuais da segurança computacional. Neste trabalho, a detecção de intrusão é vista como um problema de classificação. Então, para melhor compreensão sobre os problemas de classificação, neste capítulo são apresentados os conceitos básicos associados, bem como será apresentado o algoritmo de árvore de decisão C4.5, baseado em medidas de informação de Shannon [84], que faz parte do contexto deste trabalho. Por fim, será apresentado o uso de medidas de informação de Rényi [40] e Tsallis [41,42] aplicadas na construção de árvores de decisão C4.5, como parte integrante deste trabalho de tese.

3.1 O Processo de Classificação de Dados

Para se determinar a classe a que pertence uma dada instância, com base nos valores de seus $\aleph$ atributos, pode ser utilizado um classificador, também chamado de modelo de decisão [11]. Este modelo é construído por meio de um processo de aprendizagem, normalmente categorizado como aprendizagem supervisionada, realizado com base em um conjunto de instâncias rotulados em diferentes classes, levando em consideração os valores de seus atributos, utilizado para treinar um classificador, conforme ilustrado na Figura 3.1.

Em geral, o conjunto de instâncias é dividido em dois subconjuntos disjuntos: o conjunto de μ instâncias de treinamento, que é usado para o aprendizado do conceito e o conjunto de m (normalmente $m < \mu$) instâncias de teste usado para medir a capacidade de generalização do modelo indutivo na predição da classe de novos exemplos. Esse processo de divisão da base de dados é realizado com o objetivo de assegurar que as medidas obtidas, utilizando o conjunto de teste, sejam de um conjunto diferente do usado para realizar o aprendizado, tornando a medida estatisticamente válida.

Figura 3.1 Processo de indução a partir do aprendizado supervisionado.

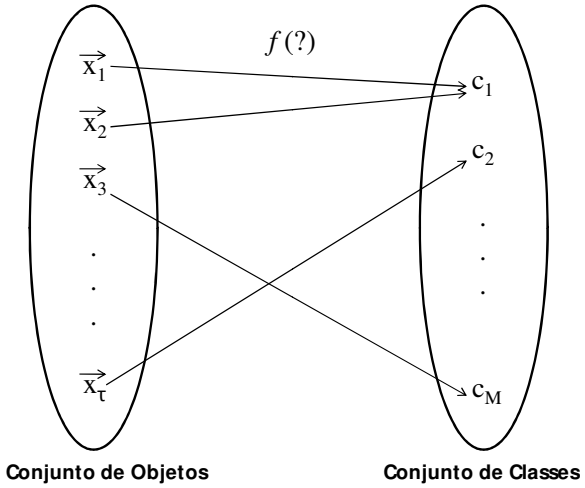


Com base na Figura 3.1, dado $T = \{(\vec{x}_1, y_1), ..., (\vec{x}_\mu, y_\mu)\}$ um conjunto de μ instâncias de treinamento com M classes e $D = \{(\vec{z}_1, \hat{y}_1), ..., (\vec{z}_m, \hat{y}_m)\}$ um conjunto de m instâncias de teste, ambos com $\aleph$ atributos $A_1, ..., A_\aleph$, em que:

- i refere-se à i-ésima instância do conjunto T ($i = 1, 2, ..., \mu$);
- o objeto $\vec{x}_i = (x_{i1}, ..., x_{iK}) \in X = \{\vec{x}_1, ..., \vec{x}_\mu\}$;
- a componente x_{ij} refere-se ao valor do j-ésimo ($j = 1, 2, ..., \aleph$) atributo A_j da instância i, em que $x_{ij} \in V_j = \{v_1, ..., v_\eta\}$ em que η refere-se ao número de valores distintos, no caso discreto (binário ou multivalorado), ou $x_{ij} \in V_j = I \subset R$, em que I é um intervalo fechado, quando o atributo é do tipo contínuo.
- y_i é uma variável que tem valores discretos e pertence a uma das M classes, ou seja, $y_i \in Y = \{c_1, ..., c_M\}$;
- l refere-se à l-ésima instância do conjunto D ($l = 1, 2, ..., m$);
- o objeto $\vec{z}_l = (z_{l1}, ..., z_{lJ}) \in Z = \{\vec{z}_1, ..., \vec{z}_m\}$;
- a componente z_{lj} refere-se ao valor do j-ésimo ($j = 1, 2, ..., \aleph$) atributo A_j da instância l, podendo ser do tipo discreto ou contínuo;
- $\hat{y}_l = f(z_i)$ é uma função, também chamada de regra de decisão, que tentará predizer a classe de um objeto $\vec{z}_i$, sendo que $\hat{y}_l$ tem valores discretos e pertence a uma das M classes, ou seja, $\hat{y}_l \in Y = \{c_1, ..., c_M\}$.

Conforme ilustrado nas Figuras 3.1 e 3.2, supondo que exista uma função $f : X \rightarrow Y$, normalmente desconhecida, que mapeia cada objeto $\vec{x}_i \in X$ a um elemento de Y, estabelecendo uma relação funcional de dependência entre o conjunto de atributos observados em cada objeto e a variável classe. Então, a tarefa de inferência indutiva é encontrar uma função $\hat{f}$ que se aproxime da função f, dado o conjunto de instâncias de treinamento. Uma vez identificada, essa função pode ser aplicada a novas instâncias de forma a predizer, com a maior precisão possível, a classe em que tais instâncias se enquadram. O classificador gerado após o processo de aprendizagem fornece uma interpretação compacta das instâncias fornecidas.

Figura 3.2 Associação entre o conjunto de objetos e as classes.



Observa-se que, nesse contexto, todo objeto do conjunto T de instâncias possui exatamente o mesmo número $\aleph$ de atributos e $\vec{x}_i$ pode ser entendido ou referenciado como um vetor de atributos. Um problema de classificação no qual $M = 2$, é denominado binário. Para $M > 2$ configura-se um problema multiclasses, que é objeto de estudo deste trabalho de tese.

Baseado no que foi exposto, o processo de indução a partir da aprendizagem supervisionada pode ser formulado como uma busca, ou seja, procura-se, entre todas as hipóteses que o algoritmo seja capaz de gerar, a partir das instâncias de treinamento, aquela com melhor capacidade de descrever o domínio em que ocorre o aprendizado.

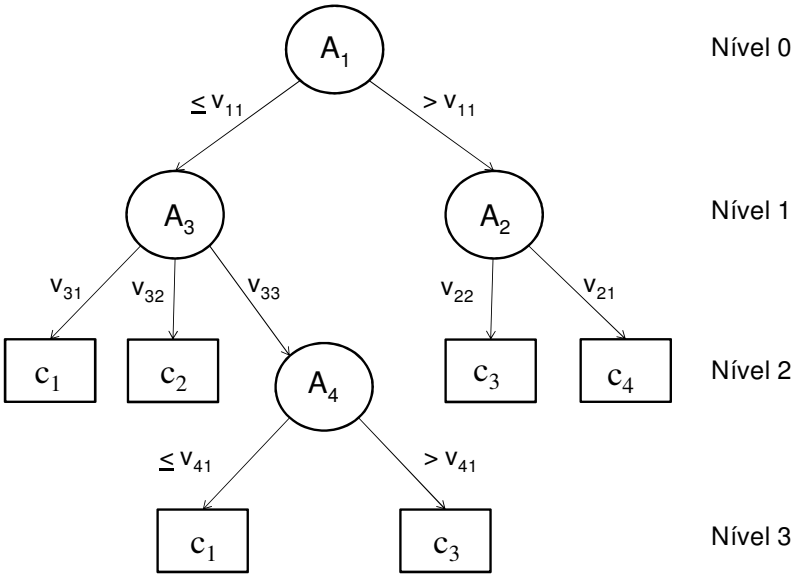
3.2 Classificação por Indução de Árvores de Decisão

Existem diversos algoritmos de classificação [57, 85], dentre eles, as árvores de decisão [86, 87]. Nesta seção é descrito, de forma resumida, o algoritmo de árvore de decisão ID3 [88] e seu sucessor, o algoritmo de árvore de decisão C4.5 [83], em razão do último ser objeto de estudo neste trabalho de tese. Os referidos algoritmos usam as medidas de informação de Shannon [84] como critério de escolha dos atributos que serão utilizados como nós da árvore gerada.

Segundo Breiman *et al.* [86], árvores de decisão são modelos utilizados em problemas de aprendizado de máquina e predição supervisionada, em que o referido modelo é induzido a partir de um conjunto de instâncias de treinamento, com as classes previamente conhecidas. Assim, pode-se definir árvore de decisão como sendo uma função que retorna uma decisão (uma classe em particular), dado um conjunto de valores de atributos de entrada. O mapeamento que ocorre entre o espaço de entrada (ou espaço de atributos) e a saída é denominado modelo preditivo.

Uma Árvore de Decisão, ilustrada na Figura 3.3, pode ser representada por meio de um grafo dirigido, em que cada nó (também chamado de vértice ou nó de decisão) especifica um atributo de teste, os arcos (ou arestas) representam o resultado desse teste. Um nó folha (ou simplesmente folha) representa uma classificação ou resultado predito. Além disso, o mesmo modelo também pode ser expresso por meio de um conjunto de regras SE-ENTÃO, sendo, portanto, mais intuitivo no nível de usuário.

Figura 3.3 Exemplo de árvore de decisão.



A classificação de uma nova instância $\vec{z}_l$ é realizada, percorrendo a árvore a partir de um nó raiz (nó A_1 na Figura 3.3), testando o atributo relacionado a este nó e seguindo o arco que corresponde ao valor do atributo na instância em questão. Este processo é repetido então para a sub-árvore abaixo, até chegar a um nó folha. A folha corresponderá à classe c_ℓ predita para essa instância. Dessa forma, a árvore de decisão é uma estrutura intuitivamente compreensível para classificar um padrão por meio de uma sequência de perguntas, em que o próximo questionamento depende da resposta da pergunta atual.

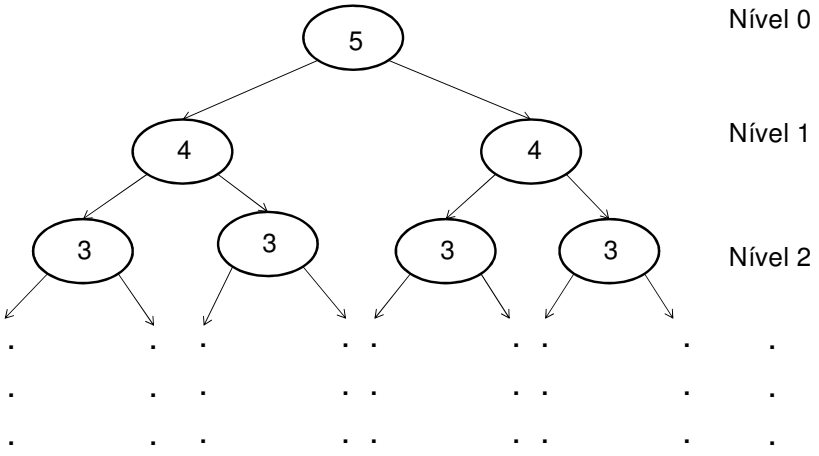
Em geral, na construção da árvore, quando o atributo é do tipo contínuo, é realizada a divisão em intervalos de valores. Por exemplo, é escolhido um único valor v, dentre os valores do intervalo (limiar de partição), com base num determinado critério adotado pelo algoritmo, e é feito um teste binário cujos resultados são $A_j \leq v$ e $A_j > v$, conforme ilustrado na Figura 3.3,

para os nós A_1 e A_4. O critério para a escolha do limiar de partição adotado pelo algoritmo C4.5 será descrito ainda nesta Seção.

Com base nos $\aleph$ atributos e seus η valores distintos, existem muitas árvores de decisão consistentes com uma base de dados. Por exemplo, para um conjunto de instâncias de treinamento, contendo 5 atributos binários e dada a condição de que a árvore resultante deve conter no mínimo 1 nó e no máximo 5 níveis de profundidade 5, tem-se o seguinte conjunto de regras:

1. Escolhe-se um dos 5 atributos como nó raiz;
2. No nível seguinte, para cada arco, escolhe-se um dos 4 atributos restantes;
3. O processo é repetido para os demais níveis, ou seja, escolhe-se um dos 3, 2 e 1 atributos (ver Figura 3.4);
4. Fazer o mesmo procedimento (1 a 3), iniciando com $\aleph - j$ atributos, $1 \leq j \leq 4$, ou seja, para os casos em que a árvore utiliza somente 4 atributos, 3 atributos, 2 atributos ou 1 atributo.

Figura 3.4 Contagem de número de árvores.



Para uma árvore com profundidade 5, tem-se $(5 \times 4^2 \times 3^4 \times 2^8 \times 1) = 1658880$ árvores distintas. Assim, o número de árvores que podem ser geradas para a solução de um problema, para todas as profundidades possíveis, ou seja, com as profundidades 5, 4, 3, 2 ou 1, dadas as condições supracitadas, é:

$(5\times4^2\times3^4\times2^8\times1)+(4\times3^2\times2^4\times1)+(3\times2^2\times1)+(2\times1)+1 = 1659471$ árvores possíveis.

Portanto, realizar uma busca exaustiva da melhor árvore que descreve os dados nesse espaço de soluções pode ser computacionalmente inviável. Desse modo, os métodos heurísticos que exploram um espaço de soluções reduzido, são comumente utilizados na tarefa de selecionar um subconjunto de atributos. Como exemplo, tem-se o *Top-Down Induction of Decision Tree*

(TDIDT), que é um algoritmo utilizado como base para muitos algoritmos de indução de árvores de decisão, dentre eles os algoritmos ID3 [88] e C4.5 [83].

O TDIDT é um algoritmo recursivo de busca guloso (*greedy*), ou seja, nunca retrocede para reconsiderar escolhas feitas anteriormente [57], no sentido de que a busca pelo espaço de soluções é sempre conduzida pela melhor escolha a cada momento. O algoritmo TDIDT produz regras de decisão de forma implícita numa árvore de decisão, a qual é construída por sucessivas divisões das instâncias de treinamento (conhecido como particionamento recursivo [89]), de acordo com os valores de seus atributos.

Na prática, o método guloso é geralmente muito eficiente. Contudo, o sucesso de um algoritmo de aprendizado por árvore de decisão depende do critério utilizado para escolher o atributo que irá particionar o conjunto de instâncias em cada iteração. Existem vários métodos usados na seleção de atributos [90]. Em Ben Bassat [91], uma classificação de regras de avaliação de atributos é apresentada em três categorias básicas, conforme a seguir:

- Regras derivadas de medidas de distância entre a distribuição de probabilidade das classes, por exemplo, uso do índice Gini [86];
- Regras derivadas de medidas de dependência estatística entre duas variáveis aleatórias [90].
- Regras derivadas da teoria da informação, que usam, por exemplo, o cálculo de entropia e informação mútua. O uso das medidas de informação de Shannon [84] é bastante utilizado no reconhecimento de padrões [92].

A utilização do conceito de medidas de informação no processo de construção de árvores de decisão tem como objetivo a criação de árvores com menor profundidade e mais eficazes na classificação [93]. As medidas de informação de Shannon [84], Rényi [40] e Tsallis [42], que serão usadas neste trabalho de tese, estão descritas no Apêndice C.

3.2.1 Indução de árvores de decisão C4.5

A indução de árvores de decisão, ilustrada no algoritmo da Tabela 3.1, baseada nos algoritmos ID3 - *Induction of Decision Tree* [88] e C4.5 [83], é realizada por meio de um processo recursivo, guloso, de cima para baixo (*top-down*), a partir da raiz em direção aos nós folhas, com base nos atributos que melhor reduzem a incerteza de cada iteração, enquanto a árvore é construída. Os algoritmos ID3 [88] e C4.5 [83] usam a medida de entropia de Shannon [84] como critério padrão de seleção dos atributos que serão utilizados na geração do modelo a partir de uma base de dados de treinamento. Esse processo se repete recursivamente até que todas as instâncias de treinamento ou todos os atributos já tenham sido utilizados (ver Apêndice I).

Seja C uma variável aleatória discreta definida num alfabeto com M símbolos, tais que $Pr[C = c_\ell] = p_\ell$, $\sum_{\ell=1}^{M} p_\ell = 1$, c_ℓ é a ℓ-ésima classe e p_ℓ é a probabilidade de uma instância pertencer à classe c_ℓ. A entropia $H(C)$ (Equação (C.1)) é o número médio de perguntas (sim/não) para identificar uma classe.

ENTRADA: $T = \{(\vec{x}_1, y_1), ..., (\vec{x}_\mu, y_\mu)\}$, lista de atributos $A = \{A_1, ..., A_J\}$
SAÍDA: Árvore de decisão induzida
INICIO
Passo 0. Se todos os exemplos em T são de uma única classe
Passo 0.1. Retornar uma única folha com o rótulo da classe.
Passo 0.2 Apresentar DT induzida
Passo 1. Escolher o atributo que ocasiona maior redução de incerteza (maximiza o ganho de informação).
Passo 2. Estender a árvore, adicionando um arco para cada valor do atributo escolhido.
Passo 3. Passar os exemplos para os arcos (tendo em conta o valor do atributo escolhido).
Passo 4. Para cada folha, faça
Passo 4.1. Se todos os exemplos são da mesma classe, associar essa classe à folha.
Passo 4.2. Senão repetir os passos 1 a 4.
Passo 5. Apresentar DT induzida
FIM

Tabela 3.1 Algoritmo de Construção de uma Árvore de Decisão C4.5.

Seja A uma variável aleatória discreta definida num alfabeto com η símbolos, tais que $Pr[A = v_\nu] = q_\nu$, $\sum_{\nu=1}^{\eta} q_\nu = 1$, em que v_ν é o ν-ésimo valor do atributo A_j, para $j = 1, ..., \aleph$, que está sendo testado e q_ν é a probabilidade do valor v_ν ocorrer na base de dados. A informação mútua $I(C; A)$, dada pela Equação (C.5), mede a redução da incerteza da classificação, ocasionada pela escolha de um atributo A_j como nó da árvore.

O algoritmo ID3 usa a informação mútua $I(C; A)$ como critério para seleção dos atributos. Entretanto, segundo Quinlan [83], este critério, apesar de apresentar bons resultados, possui uma forte tendência a favorecer os testes de atributos com muitos valores.

Por exemplo, considerando um conjunto de instâncias de diagnóstico médico no qual um dos atributos contém o código de identificação do paciente (atributo ID). Uma vez que cada código ID é único, o particionamento do conjunto de treinamento, com base nos valores deste atributo, gerará um grande número de subconjuntos, cada um contendo somente um caso. Como todos os subconjuntos gerados contêm um único elemento, ou seja, contêm exemplos de uma única classe, então a informação mútua deste atributo será máximo. Para contornar este problema, Quinlan [83] propõe a utilização de uma normalização no algoritmo C4.5, conhecida como razão de ganho (GR, do inglês **gain ratio**), dado por:

$$GR = \frac{I(C; A)}{H(A)}, \tag{3.1}$$

em que $H(A)$ é a quantidade de informação gerada pela divisão de T em $T_1, ..., T_\eta$ subconjuntos, devido à escolha de um atributo A_j como nó da árvore.

Outra diferença entre os algoritmos supracitados é que o algoritmo ID3 trabalha apenas com atributos discretos, enquanto que o algoritmo C4.5 possui a capacidade de trabalhar com atributos contínuos, atributos com ruídos e instâncias que possuem atributos cujos valores são desconhecidos. Além disso, o algoritmo C4.5 pode realizar o procedimento de poda, removendo

as comparações redundantes ou remover subárvores para melhorar o desempenho da árvore de decisão.

O algoritmo C4.5 define o limiar de partição (*split threshold*) para cada atributo contínuo da base de dados de treinamento, com base num conjunto de limiares candidatos. Em [94], o limiar de um dado atributo A_j é selecionado da seguinte forma:

1. Ordena-se os valores distintos de A_j para obter a sequência $v_1, v_2...v_\eta$;

2. Para alguns indutores, os limiares candidatos são todos os valores pertencentes ao conjunto
$$\{\theta \in \Re \mid \theta = \frac{(v_\nu + v_{\nu+1})}{2}, \forall \nu \in \{1, ..., \eta\}\}.$$
Dessa forma, existe $\eta - 1$ limiares de partição, cada um devendo ser examinado;

3. O algoritmo C4.5, entretanto, escolhe o maior valor de v, dentro do intervalo, que não exceda o ponto médio θ, assegurando que todos os valores que aparecem na árvore, de fato ocorrem nos dados.

4. Após a determinação dos limiares candidatos, o algoritmo C4.5 *Release* 7 seleciona o melhor limiar, por meio da razão do ganho, Equação (3.1). Assim, todos os limiares são testados e é selecionado aquele que apresenta maior razão de ganho;

5. É gerado um teste binário cujos resultados são $A_j \leq v$ e $A_j > v$, sendo v o limiar escolhido.

O algoritmo C4.5 *Release* 8 difere do *Release* 7 pela aplicação do princípio MDL (*Minimum Description Length*) [95]. A seguir, uma breve descrição desse princípio, segundo [96].

Supondo duas entidades: um transmissor e um receptor, que possuem uma lista ordenada de instâncias de uma base de dados. Admite-se que o transmissor, conhecendo as classes de suas instâncias, desenvolve uma teoria para classificá-las. Contudo, esta teoria pode ser imperfeita, ou seja, nem todos os casos serão corretamente classificados. Então, analisa-se que, quanto melhor for a teoria, maior será o custo de desenvolvê-la e menor será o número de classificações incorretas. Do contrário, quanto pior a teoria, menor o custo de desenvolvê-la e maior o número de classificações incorretas. Dessa forma, objetiva-se encontrar um meio termo entre os dois casos. Esta é a descrição geral do MDL.

Segundo [94], uma adaptação do princípio MDL, no algoritmo C4.5 *Release* 8 considera que o custo para a escolha do melhor atributo classificador está associado ao número de limiares distintos de cada atributo. Na prática, o autor sugere que a informação mútua $I(C; A)$ calculada para um atributo contínuo é penalizada na medida do acréscimo de custo, determinado pela transmissão do limiar, conhecida como ganho ajustado. Assim, sendo η o número de valores distintos do atributo considerado, a Equação (C.5) é modificada para.

$$GA = I(C; A) - \log_2 \frac{\eta - 1}{|T|}. \tag{3.2}$$

Os limiares com ganho negativo, ou seja, cuja informação mútua não consegue superar a penalização, são automaticamente descartados. Experimentalmente, foi observado que as árvores geradas pelo algoritmo C4.5 *Release* 8 tendem a ser mais robustas do que as geradas pelo *Release* 7 [94].

A seguir, será apresentado, com mais detalhe, o processo de criação da árvore de decisão, usando o algoritmo C4.5 *Release* 8, com base nas medidas de informação de Shannon [84]. Um exemplo ilustrativo de construção de uma árvore de decisão C4.5 pode ser consultado no Apêndice I.

1. **Inicialização**. Dado μ o número total de instâncias de treinamento, em que cada objeto $\vec{x}_i$ é formado por $\aleph$ atributos $A_1, ..., A_j$ para $j = 1, ..., \aleph$, com seus respectivos valores $x_{ij} \in V_j = \{v_1, ..., v_\eta\}$, os quais definem uma classe $c_\ell \in C = \{c_1, c_2, ..., c_M\}$, em que M é o número de classes (eventos) do problema. Caso um atributo A_j assuma valores contínuos, é realizado o procedimento descrito anteriormente.

2. **Determinar o esquema eficiente de perguntas**. Qual o melhor atributo a ser utilizado como nó da árvore?

   - **Calcular a entropia**. Inicialmente deve-se calcular a quantidade média de informação necessária para se classificar uma determinada instância de T, dada pela Equação (C.1).
   - **Calcular a informação mútua para cada atributo** A_j, com seus respectivos valores $v_1, ..., v_\eta$, dada pela Equação (C.5).
   - **Calcular o ganho ajustado**. Calcular o ganho ajustado para cada atributo candidato, dada pela Equação (3.2).
   - **Escolher atributo**. O atributo que maximizar o ganho ajustado, será escolhido para ser o nó da árvore. Será criado um arco para cada valor do atributo escolhido.
   - **Estender a árvore**. Adicionar um ramo para cada valor $v_1, ..., v_\eta$ do atributo escolhido A_j como nó da árvore, de acordo com seu tipo (contínuo ou discreto). Nesse momento, o conjunto de treinamento T é subdividido $T_1, ..., T_\eta$ partições, ou seja, cada subdivisão da árvore terá um novo conjunto de treinamento T_ν, contendo todas as instâncias em T que possuem como resultado daquele teste o valor v_ν. Para cada ramo criado, se as instâncias em T_ν são da mesma classe, então associar essa classe a uma folha, senão, voltar ao passo *Calcular a entropia*, usando o conjunto T_ν como nova instância de treinamento.

3. **Finalização**. O modelo gerado é a solução do problema proposto.

Um problema que pode ocorrer na construção de uma árvore é que a mesma pode ficar sobre-ajustada (*overfitting*) aos dados de treinamento e, assim, diminuir a sua capacidade de generalização. Para evitar tal situação e para garantir que fossem construídas árvores mais compactas e com menos folhas, é necessário que seja executado o procedimento de poda da árvore.

3.2.2 Procedimento de poda em árvores de decisão C4.5

Os métodos de poda podem ser divididos em duas abordagens principais:

- a pré-poda: durante a criação da árvore, algumas instâncias de treinamento são deliberadamente ignoradas, de forma que a árvore gerada não classifique todas as instâncias de treinamento corretamente;
- a pós-poda: após a criação da árvore, esta é generalizada, por meio da eliminação de alguns ramos e, consequentemente, reduzir subárvores a folhas.

Segundo [86], a pós-poda é mais lenta, porém mais confiável que a pré-poda, pelo fato do risco de interromper o crescimento da árvore ao selecionar uma árvore subótima.

A estratégia pós-poda é adotada pelo algoritmo *C4.5*. Podar uma árvore neste contexto, significa cortar um nó da árvore, transformado-o em folha. O corte de um determinado nó i da árvore é baseado na comparação das taxas de estimativa de erro de classificação de cada subárvore e do nó folha. São processados sucessivos testes a partir do nó raiz da árvore, de forma que, se a estimativa de erro indicar que o desempenho da árvore não diminuirá consideravelmente, se os nós descendentes (filhos) do nó i forem eliminados, então estes nós descendentes serão eliminados e o nó i passará a ser o novo nó folha.

O algoritmo *C4.5* faz uma estimativa pessimista do erro, para cada nó [97]. Para estimar a probabilidade de erro, buscam-se limites de confiança CF. O limite superior desta probabilidade pode ser obtido a partir dos limites de confiança para a distribuição binomial e é escrito como $U_{CF}(E, L)$ (E erros de classificação entre L instâncias classificadas). Os limites superior e inferior são simétricos na distribuição binomial, portanto a probabilidade de que a média real de erros de classificação exceda $U_{CF}(E, L)$ é $CF/2$. A estimativa de erro pessimista de classificação de L instâncias é $LxU_{CF}(E, L)$ [83]. Foi utilizado nos experimentos o valor padrão de $CF = 0,25$.

Um dos parâmetros utilizados no procedimento de poda é o número mínimo de instâncias em uma folha, que define um critério de parada para o algoritmo. Isso significa que, a partir do momento que uma folha atinge um certo número de instâncias, mesmo que ainda exista ruído, ela não será subdividida. Nos experimentos, foi utilizado o valor mínimo de duas instâncias por folha, valor padrão do algoritmo.

3.2.3 Medidas de informação de Rényi e Tsallis aplicadas na construção de árvores de decisão C4.5

As medidas de entropia propostas por Shannon [84] encontram muitas aplicações não somente na engenharia, mas também em diversas áreas, tais como a estatística, economia, reconhecimento de padrões e aprendizado de máquina. Nessa perspectiva, dentro da teoria da informação, foram formuladas propostas de generalização da entropia, algumas claramente relacionadas, chegando a reduzirem-se à mesma expressão de Shannon [84], em alguns casos particulares, tais como a formulação de Rényi [40] e Tsallis [41, 42] (ver Apêndice C).

As entropias de Rényi [40] e Tsallis [42] contêm um coeficiente α que pode ser usado para ajustar a sensibilidade em relação à distribuição de probabilidade. Usando a entropia de Shannon [84], os eventos com probabilidade alta ou baixa têm pesos iguais no cálculo da entropia.

No entanto, usando a entropia de Tsallis [42], para $\alpha > 1$, eventos mais frequentes são enfatizados, ou seja, os eventos com alta probabilidade contribuem mais do que os de baixa probabilidade. Assim, quanto maior for o valor de α, maior será a contribuição dos eventos de alta probabilidade para o resultado final [98].

Da mesma forma, o aumento de α ($\alpha \to \infty$) contribui para que a entropia de Rényi [40] seja determinada pelos eventos com probabilidades mais elevadas, e, ao contrário, diminuindo os valores de α ($\alpha \to 0$), os eventos passam a ter pesos iguais, independentemente de suas probabilidades.

Em [99], foi verificado que a entropia de Rényi [40] possui características que a torna mais eficiente do que a entropia de Shannon [84] na resolução de problemas binários e em [100], os autores também comprovam empiricamente que a entropia de Tsallis [41, 42] pode ser utilizada em problemas de classificação, apresentando resultados motivadores.

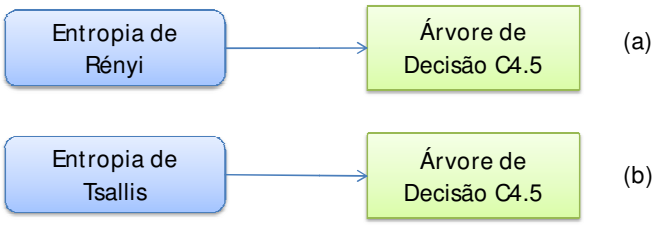
Portanto, as características supracitadas servem de motivação para que as medidas de informação de Rényi [40] e Tsallis [42] sejam aplicadas, uma vez que estas podem determinar esquemas de perguntas mais eficientes, construindo, assim, árvores de decisão mais adequadas ao problema de detectar intrusos em redes de computadores.

Nesse contexto, as medidas de informação de Rényi [40] e Tsallis [42] foram implementadas (ver códigos fonte no Apêndice K), como parte integrante desta tese, na ferramenta Weka [101], no algoritmo J48 (implementação em linguagem Java do algoritmo C4.5, *Release* 8, descrito no Capítulo 3), em substituição às medidas de informação de Shannon [84], conforme ilustrado na Figura 3.5.

A adoção do WEKA no contexto deste trabalho se deu por vários fatores, a saber:

- Contempla uma série de algoritmos de preparação de dados, aprendizagem de máquina para tarefas de mineração de dados, e contém ferramentas para pré-processamento de dados, classificação, regressão, agrupamento, descoberta de regras de associação, visualização e validação de resultados;

Figura 3.5 Esquemas para construção de árvore de decisão C4.5 baseada no cálculo de entropia de Rényi (a) e Tsallis (b)



- É gratuita, possui código aberto sob licença GNU (Licença Pública Geral) e é escrita em linguagem de programação Java, o que possibilita a sua portabilidade para diversos sistemas operacionais.

Como as entropias de Rényi [40] e Tsallis [42] oferecem diferentes resultados no cálculo de entropia, de acordo com o valor de α adotado, buscou-se encontrar coeficientes ótimos para o problema de detectar intrusos em redes de computadores.

Dessa forma, foi verificado empiricamente que as medidas de informação de Rényi [40] (com parâmetro $\alpha \in (0, 1)$) e Tsallis [42] (com parâmetro $\alpha > 1$), alcançaram melhor desempenho na solução do problema proposto, construindo árvores mais compactas e eficientes, de forma mais rápida, se comparados às medidas de informação de Shannon [84].

Essas duas estratégias empregadas na construção de árvore de decisão C4.5 serão utilizadas nos esquemas de seleção de atributos propostos neste trabalho.

3.3 Considerações Finais

Neste capítulo, foram abordados os conceitos relacionados ao processo de classificação de dados. Como classificadores, há diversos algoritmos de árvore de decisão implementados, sendo os mais populares provavelmente o *ID3* (*Induction of Decision Tree*), desenvolvido por Quinlan [88] com base no *CLS* (*Concept Learning System*) [102] e seu sucessor, o *C4.5* [83]. Ambos constroem a árvore em um processo *top-down*, selecionando o atributo de teste apropriado para cada nó de decisão da árvore, por meio de medidas de informação de Shannon [84], determinando, por fim, um esquema de perguntas adequado ao problema proposto. Contudo, outras estratégias podem ser utilizadas.

Nesse contexto, como uma das contribuições deste trabalho de tese, duas estratégias foram apresentadas neste capítulo, a saber: uma estratégia utilizando as medidas de informação de Rényi [40] e outra utilizando as medidas de informação de Tsallis [42], com o objetivo de encontrar alternativas mais eficientes na construção de árvores de decisão *C4.5* [83]. Alguns resultados da aplicação destas estratégias foram apresentadas em [16, 103] e podem ser encontrados nos anexos deste trabalho de tese.

Neste trabalho, como mais uma contribuição, as medidas de informação de Rényi [40] e Tsallis [42] são utilizadas na seleção dos atributos que servirão como pontos de observação dos detectores de rede e de *host* (ver Capítulo 5), semelhante ao critério de seleção de atributos adotado na construção de árvores de decisão C4.5, a fim de obter um subconjunto de atributos ideais que incremente a capacidade de um Sistema de Detecção de Intrusão para classificar uma atividade como normal ou como suspeita (categorizando o tipo de ataque).

A partir das pesquisas sobre classificação de dados, surgiu a necessidade de usar novas abordagens que contribuíssem para aprimorar a utilização de códigos corretores de erros (revisado no Apêndice D). Em busca de novas soluções, foi feito um estudo sobre estratégias de decomposição de problemas multiclasses em subproblemas binários baseadas em matrizes código [19, 38]. Nesse contexto, no Capítulo 4 são abordadas estratégias usadas em classificação multiclasses, com ênfase no uso de códigos corretores de erros, e que servem de base para a escolha da estratégia a ser utilizada neste trabalho de tese.

CAPÍTULO 4

Solução de Problemas Multiclasses baseadas em Estratégias Decomposicionais

No Capítulo 3 foram apresentados os conceitos básicos sobre classificação de dados, dando enfoque aos algoritmos de árvore de decisão baseados em medidas de informação. Como identificar ataques pode ser visto como uma tarefa de classificação multiclasses, em geral, busca-se por soluções que possam diminuir a complexidade do problema, em que uma das formas é adotar alguma estratégia decomposicional. Portanto, neste capítulo é referenciado em que contexto surge a decomposição de problemas multiclasses. Em seguida, são apresentadas as estratégias decomposicionais comumente encontradas na literatura científica, fazendo-se um comparativo com a estratégia baseada em códigos corretores de erros.

4.1 Decomposição de Problemas Multiclasses

A detecção de intrusão é uma tarefa de classificação que está recebendo grande atenção, devido ao contínuo aumento do número de ataques em redes de computadores. A tarefa de classificação consiste em construir um modelo preditivo que seja capaz de distinguir entre intrusões e conexões normais. Portanto, o referido modelo pode ser construído, considerando o problema de duas classes, com foco na distinção do que é um padrão normal ou uma atividade intrusiva, ou ser tratado como um problema com múltiplas classes, referenciada pelo termo multiclasses, envolvendo a classificação dos diferentes ataques sofridos.

Existem duas principais abordagens para lidar com problemas que envolvem a classificação multiclasses. Na primeira abordagem, busca-se soluções para o problema, por meio da utilização de algoritmos multiclasses, tais como os algoritmos de árvores de decisão utilizados nesta tese. Na segunda abordagem, o problema multiclasses é decomposto em subproblemas, de forma que, após sua decomposição, estes possam ser tratados com menor complexidade.

Esta estratégia, conhecida como decomposicional, segue o critério *dividir para conquistar*, que baseia-se na divisão do problema em vários subproblemas de menores dimensões, até que uma solução para cada um dos problemas mais simples possa ser encontrada. Esta decomposição apresenta algumas vantagens que motivam seu uso, tais como:

- Possibilita a redução da complexidade envolvida na separação das classes [104, 105];
- Viabiliza a utilização de algoritmos específicos que só conseguem tratar problemas com duas classes [18, 38];
- Proporciona o processamento paralelo [18, 38];
- Permite reduzir o número de erros de classificação de diferentes classes que possuem pesos distintos. Dessa forma, pode-se gerar classificadores binários de maneira a impor preferências para algumas classes [18, 38].

As desvantagens do uso de estratégias decomposicionais dependem da forma como ocorre a decomposição do problema e, portanto, serão discutidas no decorrer da Seção 4.2.

Determinar qual estratégia a ser usada na decomposição do problema multiclasses é uma questão importante, uma vez que o desempenho da solução está fortemente relacionado com a estratégia escolhida. Dentre as estratégias decomposicionais existentes, as mais comumente encontradas na literatura [20–23], e que podem ser representadas por uma matriz $C_{M\times N}$, aqui referenciada como matriz código (não confundir com matriz geradora do código, definida no Apêndice D), serão apresentadas na Seção 4.2. No contexto das estratégias decomposicionais, M é o número de classes do problema multiclasses ($M > 2$) e N é o número de subproblemas binários, determinando, assim, os classificadores $f_1, f_2, ...f_N$ a serem utilizados na solução.

4.2 Estratégias Decomposicionais

Dado um problema multiclasses, definido por $T = \{(\vec{x}_i, y_i)\}_{i=1}^{\tau}$, um conjunto de treinamento, em que $\vec{x}_i$ é um vetor de atributos, $y_i \in \{c_1, c_2, ..., c_M, M > 2\}$ e M é o número de classes do problema. O emprego de estratégias decomposicionais envolve basicamente a realização de dois procedimentos: primeiro o problema multiclasses é decomposto em N subproblemas, determinando os classificadores $f_1, f_2, ...f_N$ a serem utilizados na solução. Em seguida, é realizado o procedimento de decodificação ou reconstrução, que determina a forma como as saídas dos classificadores são combinadas para predizer a classe de uma instância.

Conforme ilustrado nas Figuras 1.2, 4.1 e 4.2, cada linha $\mathbf{C}_\ell, , 1 \leq \ell \leq M$ da matriz $C_{M\times N}$ representa um vetor de hipóteses que está associado a uma única classe $c_\ell, 1 \leq \ell \leq M$. As suas colunas correspondem às hipóteses dadas a todas as classes por cada classificador $f_j, 1 \leq j \leq N$ gerado na fase de decomposição.

Na fase de decodificação, uma dada instância de teste $\vec{x}$ é aplicada a cada classificador $f_1, f_2, f_3 e f_N$, dando origem a N hipóteses distintas. O vetor de hipóteses gerado pelos classificadores é definido como $\mathbf{F}(\mathbf{x}) = (f_1(\vec{x}), f_2(\vec{x}), ..., f_N(\vec{x}))$. Então, o referido vetor de hipóteses $\mathbf{F}(\mathbf{x})$ será comparado com as linhas $\mathbf{C}_\ell$ da matriz código $C_{M\times N}$ e aquela que for mais próxima de $\mathbf{F}(\mathbf{x})$, de acordo com uma dada medida de distância d (por exemplo, a distância de *Hamming* [24], descrita no Apêndice D), será atribuída como a classe da instância $\vec{x}$. Em outras palavras, a classe predita é aquela que minimiza $d(\mathbf{C}_\ell, \mathbf{F}(\mathbf{x}))$.

4.2.1 Estratégia Um-Contra-Todos

Na estratégia Um-Contra-Todos [20], um problema multiclasses (com M classes) é decomposto em $N = M$ subproblemas binários, gerando N classificadores binários. Cada classificador $f_j(\vec{x}), j = 1, 2, ..., N$ é treinado de forma a distinguir uma classe $c_\ell, 1 \leq \ell \leq M$ das demais, ou seja,

$$f_j(\vec{x}) = \begin{cases} 1, & \text{se } \vec{x} \in \{c_\ell\}; \\ 0, & \text{caso contrário.} \end{cases} \tag{4.1}$$

A representação da estratégia Um-Contra-Todos é dada por uma matriz identidade $C \in \{1, 0\}^{M\times N}$, conforme ilustrado na Figura 4.1(a).

Figura 4.1 Exemplo de estratégias decomposicionais (a) Matriz Um-Contra-Todos (b) Matriz Todos-Contra-Todos.

	f_1	f_2	f_3	f_4
c_1	1	0	0	0
c_2	0	1	0	0
c_3	0	0	1	0
c_4	0	0	0	1

(a)

	$f_{1,2}$	$f_{1,3}$	$f_{1,4}$	$f_{2,3}$	$f_{2,4}$	$f_{3,4}$
c_1	+1	+1	+1	0	0	0
c_2	-1	0	0	+1	+1	0
c_3	0	-1	0	-1	0	+1
c_4	0	0	-1	0	-1	-1

(b)

Em síntese, a estratégia Um-Contra-Todos consiste em selecionar cada uma das M classes do problema original, considerando-a como sendo a classe principal, contra as $M - 1$ classes restantes, formando N conjuntos de treinamento. Dessa forma, cada conjunto de treinamento binário será composto pelas instâncias da classe principal e pelas instâncias de todas as demais classes (as classes restantes são renomeadas para uma única classe, aqui chamada de *outras*). Os N conjuntos de treinamento binários terão a mesma dimensão do conjunto de treinamento multiclasses original. Com base nesses conjuntos de treinamento, serão gerados um total de N classificadores binários.

Por exemplo, em um problema contendo 4 classes c_1, c_2, c_3, c_4, a decomposição pelo método Um-Contra-Todos conduz à criação de 4 problemas binários: c_1 *vs outras*, c_2 *vs outras*,

c_3 *vs outras* e c_4 *vs outras*. Assim, os 4 conjuntos de treinamento binários terão todas as classes e o mesmo número de atributos do conjunto de treinamento multiclasses. No entanto, as classes que não fossem da classe em questão, seriam consideradas como uma classe única - *outras*. Com base nos conjuntos de treinamento gerados a partir da estratégia de decomposição Um-Contra-Todos [20], serão treinados um total de 4 classificadores $f_1, f_2, f_3 e f_4$ (ver Figura 4.1(a)).

Cada instância $\vec{x}$ do conjunto de teste é classificada por todos os classificadores binários, para a obtenção da classificação final. Na etapa de decodificação, pode ser utilizado o método baseado em margem [30], em que a classe escolhida correspondente ao classificador que produz a maior saída.

O método Um-Contra-Todos pode apresentar desvantagens quando a proporção de instâncias de uma classe é muito pequena em relação ao conjunto formado pelas instâncias das demais classes. Esse desbalanceamento pode dificultar a indução de um classificador com alto desempenho no reconhecimento da classe minoritária. Além disso, do ponto de vista da correção de erros, a codificação Um-Contra-Todos é incapaz de se recuperar de qualquer erro, uma vez que a distância de *Hamming* dessa estratégia é $d = 2$.

4.2.2 Estratégia Todos-Contra-Todos

Na estratégia Todos-Contra-Todos [21, 22], um problema multiclasses (com M classes) é decomposto em $N = \frac{M(M-1)}{2}$ subproblemas binários, com todas as combinações possíveis das classes do problema original, formando subproblemas contendo apenas uma par de classes $(c_a, c_b)|\forall(a, b) \in [1, ..., M], a \neq b$, cada um com seu classificador $f_{a,b}$ independente, num total de N classificadores, ou seja,

$$f_{a,b}(\vec{x}) = \begin{cases} +1, & \text{se } \vec{x} \in \{c_a\}; \\ -1, & \text{se } \vec{x} \in \{c_b\}. \end{cases} \tag{4.2}$$

A representação da estratégia Todos-Contra-Todos é dada por uma matriz código ternária $C \in \{1, 0, -1\}^{M \times N}$, indicando que uma determinada classe não é considerada, por meio do valor 0, conforme ilustrado na Figura 4.1(b).

Basicamente, a estratégia consiste em criar subconjuntos de dados para cada par de classes $\{c_a \cup c_b | c_a \cap c_b = \emptyset\}$. Estes subconjuntos têm dimensão inferior ao conjunto de treinamento do problema multiclasses original. Para cada uma das combinações, é gerado um classificador binário $f_{a,b}$ que consegue distinguir somente um par de classes.

Por exemplo, supondo-se um problema de quatro classes c_1, c_2, c_3, c_4, a aplicação do método Todos-Contra-Todos implica na decomposição das seguintes combinações possíveis: $c_1 - c_2$, $c_1 - c_3$, $c_1 - c_4$, $c_2 - c_3$, $c_2 - c_4$ e $c_3 - c_4$. Para cada uma das combinações, é criado um classificador usando apenas as instâncias das classes envolvidas no conjunto de treinamento, ou seja, para o problema $c_1 - c_2$ é criado um classificador $f_{1,2}$ a partir de um conjunto de

treinamento composto pelas instâncias da classe c_1 e pelas instâncias da classe c_2 e assim é feito em todas as demais combinações.

No procedimento de decodificação, existem muitas abordagens para combinar os resultados obtidos, tais como o esquema de votação por maioria [106] e grafo de decisão direcionado acíclico (*Decision Directed Acyclic Graph* - DDAG) [107].

Uma vantagem da estratégia Todos-Contra-Todos em relação à estratégia Um-Contra-Todos é que como o treinamento de cada classificador envolve dados de apenas duas classes, mesmo que se tenha um número elevado de classes, o tempo total despendido na geração dos referidos classificadores geralmente não é grande. Entretanto, como o número de classificadores gerados cresce quadraticamente ao valor de M, o processo de obtenção da classificação final pode ser lento, dependendo do método de decodificação utilizado.

Uma potencial desvantagem, segundo [35], é exatamente o fato de que nessa estratégia, um classificador treinado para um par de classes $c_a - c_b$ não consegue produzir informação alguma quando a instância a ser testada não pertence ao conjunto $\{c_a, c_b\}$.

4.2.3 Estratégia baseada nos princípios dos códigos corretores de erros

A estratégia conhecida como decomposição por códigos corretores de erros (ver definições no Apêndice D) [24], utilizada nessa pesquisa, consiste em associar uma palavra código a cada classe do problema (ver Figura 4.2). Nessa abordagem, a decomposição mínima de um problema com M classes pode ser realizada com o uso de $N = \lceil \log_2 M \rceil$ classificadores binários e a decomposição máxima é dada por $N = 2^{M-1} - 1$. Dentro desse intervalo, qualquer valor pode ser usado. Portanto, o número de subproblemas e, consequentemente, o número de classificadores binários depende do tamanho da palavra código.

Figura 4.2 Exemplo de estratégia decomposicional para um problema com sete classes.

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9
c_1	1	0	0	1	0	1	1	1	1
c_2	0	1	0	0	1	0	1	0	0
c_3	0	0	1	1	0	1	0	1	0
c_4	0	1	0	1	1	0	1	1	0
c_5	1	0	1	0	1	0	1	0	1
c_6	1	1	0	1	0	0	0	0	0
c_7	0	0	1	1	0	1	0	1	1

Tomando o exemplo ilustrado na Figura 4.2, o conjunto de dados T é decomposto em vários subconjuntos de dados binários. Para treinar o classificador f_1, é necessário que se tenha uma base de dados binária contendo valor 1 para as classes c_1, c_5, c_6 e 0 para as demais. Esse processo se repete para os demais classificadores. Cada classificador gerado nesta fase,

corresponde a 1 bit da palavra código. Na fase de decodificação, uma estratégia comumente usada é a baseada na distância de *Hamming* [18, 108, 109].

Com base na teoria da codificação, no artigo [23], apesar de cometerem equívocos em relação ao uso de códigos BCH (ver Apêndice D), os autores sugerem que o número de classificadores seja maior que o mínimo necessário para diferenciar cada classe unicamente. Esses classificadores adicionais inserem redundância na codificação das classes e têm a utilidade de fornecer ao sistema a capacidade de corrigir, ou pelo menos detectar possíveis erros que possam estar presentes nas instâncias de treinamento utilizadas no projeto da matriz código, por exemplo, amostras limitadas, complexidade no limite das classes ou quaisquer erros cometidos por alguns dos classificadores binários no processo de classificação de uma instância [110].

Para possibilitar a correção de erros e determinar a classe de uma instância com maior precisão, a distância de *Hamming* mínima entre as linhas e entre as colunas da matriz código deve ser grande. As matrizes códigos obtidas devem ainda obedecer algumas regras, conforme [18, 23]:

- Inexistência de colunas iguais ou colunas complementares, pois representam o mesmo problema, dando origem ao mesmo modelo de decisão;
- Nenhuma coluna contendo apenas 0s ou 1s, pois não representam problemas de decisão.

Em geral, o número de classificadores binários é menor que a quantidade de palavras código disponíveis e isto possibilita que sejam selecionadas palavras código com distâncias razoáveis entre linhas e colunas. Além disso, estas referidas palavras código devem representar as classes do problema, a fim de que permita o aumento no desempenho da classificação multiclasses. Contudo, apesar da vantagem de se poder definir o tamanho da palavra código, dentro do limite mínimo e máximo estabelecidos, tem-se como desvantagem, o crescimento exponencial deste número, dada a quantidade de classes do problema, dificultando a definição do tamanho da palavra código, bem como a escolha dos subconjuntos de treinamento dos classificadores f_j, da palavra código .

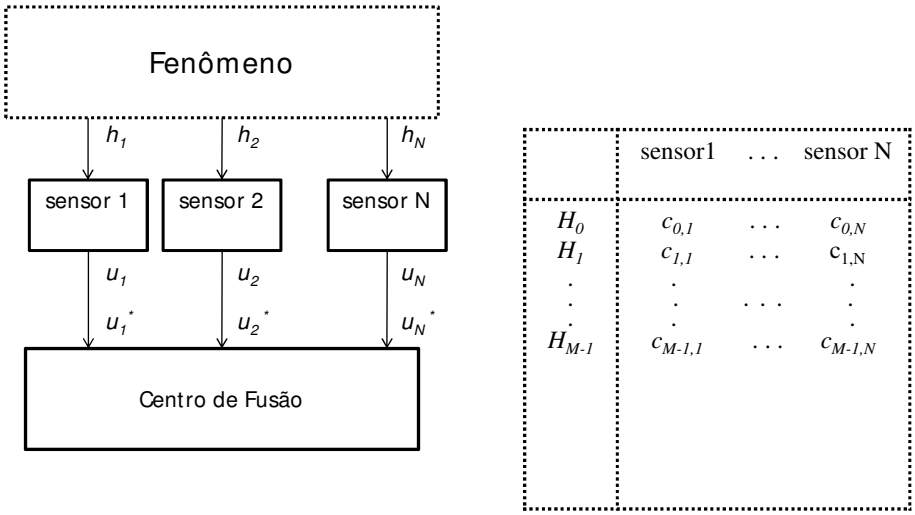
Para minimizar as dificuldades supracitadas, uma solução apresentada neste trabalho de tese envolve o uso de árvores de decisão para definir o tamanho da palavra código e o uso de um algoritmo genético para encontrar a palavra código com melhor desempenho na classificação de cada classe.

4.3 Outras Abordagens sobre Uso de Códigos Corretores de Erros em Classificação

Em [28, 111] é proposto um modelo de classificação baseado em códigos de bloco lineares (ver Apêndice D) [24] para redes de sensores sem fio [112] que seja tolerante a falhas, conforme ilustrado na Figura 4.3. As redes de sensores sem fio são, em geral, compostas por

nós sensores distribuídos em uma determinada região com o objetivo de detectar e transmitir uma característica física do ambiente. Esta abordagem é similar à proposta em [113], diferindo basicamente no projeto da matriz código.

Figura 4.3 Modelo do sistema de classificação baseado em códigos [111, 113], adaptado pela autora



Sendo M o número de classes ou hipóteses possíveis $(H_\ell\ , 0 \leq \ell \leq M-1)$ e N o número de sensores observando um fenômeno físico, as observações dos nós sensores são representadas por h_j, em que $j = \{1...N\}$.

Dado um problema de classificação, cada nó sensor distribuído no ambiente, com base em sua observação, envia sua decisão a um centro de fusão responsável por tomar a decisão final por uma das H_ℓ classes possíveis, com base numa matriz código com elementos $c_{\ell,j} \in \{0, 1\}$, cujas linhas representam uma palavra código de comprimento igual a N, associada a cada uma das M classes.

Como as decisões enviadas pelos nós sensores podem ser corrompidas por ruído, ou seja, a palavra $u^* = (u_1^*, u_2^*, ..., u_N^*)$ recebida pelo centro de fusão pode ser diferente da palavra $u = (u_1, u_2, ..., u_N)$ transmitida, uma alternativa para minimizar este problema consiste em utilizar códigos corretores de erro.

Na abordagem proposta por Santos [28, 111], as palavras código que formam a matriz código formam um sub-código de um código BCH. Dessa forma, é possível utilizar algoritmos de decodificação algébrica, diferentemente de [113, 114], em que os códigos utilizados não possuem nenhuma estrutura e a decodificação só pode ser feita por comparação da palavra recebida com as palavras pertencentes ao código. Isto evita uma decodificação massiva em sistemas em que o número de classes é elevada.

Para realizar a seleção da matriz código para a abordagem proposta por Santos *et al* [28, 111], foi utilizado um algoritmo genético (AG) guiado por códigos para garantir que as palavras código pertencessem ao código BCH [24, 37] escolhido. Este algoritmo é uma modificação do

algoritmo genético padrão proposto em [115, 116]. Foi verificado empiricamente pelos autores [28, 111] que a abordagem baseada em código BCH apresenta um desempenho similar à abordagem de [113]. Outras pesquisas, tais como [19,38], abordam o uso de algoritmo genético e servem de inspiração para esta tese.

4.4 Considerações Finais

Neste capítulo, analisou-se as estratégias decomposicionais mais comuns: um-contra-todos, todos-contra-todos e baseadas em códigos corretores de erro. Foi observado que a abordagem um-contra-todos utiliza somente os classificadores que identificam uma classe dentre as demais. A estratégia decomposicional todos-contra-todos, por sua vez, usa somente classificadores que distinguem pares de classes. Na estratégia baseada em códigos tem-se maior flexibilidade na escolha dos classificadores, dentro dos limites permitidos, podendo envolver todas as classes simultaneamente.

Usar códigos corretores de erros [24] para identificar intrusos em redes de computadores foi idealizado antes de ser realizada a pesquisa bibliográfica sobre o tema abordado. Porém, após analisar as estratégias aqui comentadas, uma crítica comum que pode ser feita é que estas realizam a decomposição do problema a *priori*, sem levar em consideração as propriedades e as características de cada aplicação [19, 35].

Outrossim, nas referências bibliográficas pesquisadas, foi encontrado um projeto intitulado *Evaluating multi-class classification using binary SVMs (IT-642: Course Project)* [117], que teve o objetivo de estudar as Máquinas de Vetores de Suporte (*Support Vector Machines - SVMs*), originalmente concebidas para a solução de problemas com apenas duas classes, aplicadas na multiclassificação. Neste projeto foram realizados experimentos executados em vários conjuntos de dados, dentre estes o *Knowledge Discovery and Data Mining Competition - KDD Cup 99* [118] que contém tráfego de rede normal e com diversos tipos de ataques.

Embora não conste no trabalho de [117] uma análise mais aprofundada dos resultados obtidos e também, apesar de novas pesquisas não tenham sido derivadas deste trabalho, seus resultados experimentais demonstram a eficiência do uso de códigos corretores de erros e encorajam o desenvolvimento de novas pesquisas para combinar métodos de classificação e utilizar a vantagem que o uso de códigos pode oferecer na área de classificação de tipos de ataques em redes de computadores. Essa análise reforça a motivação para se propor uma nova estratégia baseada em códigos corretores de erros, com o objetivo de encontrar palavras código adaptadas à solução de cada problema multiclasses. Essa proposta é descrita no Capítulo 5.

Ainda nesta de tese, tenta-se minimizar também o número de atributos que serão observados pelos detectores. No esquema proposto neste trabalho, existe uma relação direta entre o número de atributos e o tamanho da palavra código, uma vez que cada detector é responsável por enviar um bit para compor os símbolos palavra recebida.

Dessa forma, definir qual subconjunto de todos os atributos possíveis de um tráfego de rede e *log* de servidores consegue predizer melhor as atividades intrusivas é objeto de estudo desta tese e também será abordado no Capítulo 5.

CAPÍTULO 5

Esquema para Classificar Intrusões usando Códigos Corretores de Erros e Medidas de Informação

A partir das pesquisas realizadas sobre estratégias que usam matrizes código, com ênfase nas estratégias baseadas em códigos corretores de erros, aplicadas na classificação de dados, bem como as pesquisas envolvendo árvore de decisão e medidas de informação de Renyi [40] e Tsallis [42], será apresentado, neste capítulo, um esquema para detectar e classificar intrusões em uma rede de computadores, baseado no uso de um código corretor de erro do tipo BCH (binário e não binário) e medidas de informação.

Os códigos BCH [24,37], revisados no Apêndice D, são códigos de bloco lineares largamente utilizados, que pertencem à classe dos códigos cíclicos e representam uma importante generalização dos códigos de *Hamming*, permitindo múltipla correção de erros. Um código BCH pode, em alguns casos, corrigir mais que t erros. Nesse contexto, $d_{min} = 2t + 1$ é chamada de distância de projeto do código, pois essa distância mínima pode ser maior.

A princípio, as aplicações dos códigos BCH eram restritas a códigos binários, porém, a generalização para códigos não binários, com símbolos representados em $GF(q), q > 2$, foi realizada em 1961 por Gorenstein e Zierler [119]. A estrutura imposta pelos espaços vetoriais propiciam meios estruturados para a construção de codificadores e decodificadores para códigos BCH (binários e não binários).

Os códigos Reed-Solomon [120] (BCH não binário) são uma sub-classe dos códigos BCH [24, 37], com símbolos em sequências de $s-$bits, em que $s \geq 3$. Estes estão entre os mais poderosos códigos no que diz respeito à capacidade de correção de erro, sendo largamente aplicados nos dispositivos de armazenamento (fita magnética, CD's, DVD's), RAID (*Redundant Array of Independent Disks*), códigos de barra, comunicação móvel e sem fio, comunicação via satélite, televisão digital e aplicações no contexto espacial, entre outras.

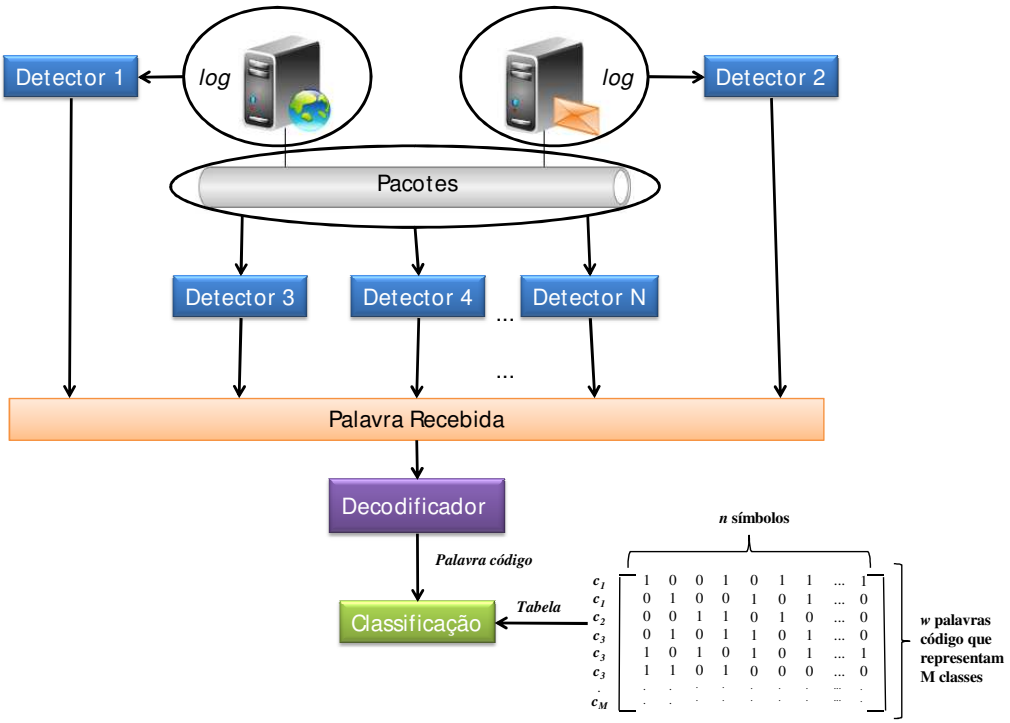
Os códigos Reed-Solomon [120] são chamados de códigos separáveis por máxima distância (MDS) porque atingem o limitante de *Singleton* com igualdade, $d_{min} = n - k + 1$, em que d_{min} é a distância mínima do código, n é o número de símbolos da palavra código e k é o número de símbolos da palavra de informação. Esses códigos são capazes de recuperar até $t = \left\lfloor \frac{n-k}{2} \right\rfloor$ erros aleatórios.

As características dos códigos BCH supracitadas servem de motivação para a sua utilização no modelo de classificação que será apresentado a seguir.

5.1 Descrição Geral do Modelo

O modelo proposto, ilustrado na Figura 5.1, prevê a metodologia de detecção por assinatura como método de análise. A escolha se deu em virtude da maioria dos ataques ter formato de uma assinatura, conforme já abordado no Capítulo 2. Outrossim, uma vez que uma intrusão pode ser identificada por meio de uma assinatura, o modelo proposto aplica códigos corretores de erros para classificar intrusões, associando uma ou mais palavras código a uma assinatura de intrusão.

Figura 5.1 Estratégia baseada em Códigos Corretores de Erros.



O esquema proposto prevê que uma dada palavra recebida pode ser decodificada, por meio de um algoritmo de decodificação algébrica [24, 37], para uma palavra código correspondente a uma assinatura de intrusão, dentro do limite $t \leq \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$ de correção de erros do código utilizado para esta finalidade.

Nesta perspectiva, para cada tipo de ataque serão associadas palavras código pertencentes a um código do tipo BCH [24, 37]. Como atualmente os ataques às redes de computadores são polifórmicos [36], existe a necessidade de se ter mais de uma palavra código associada a cada tipo de ataque. Portanto, no modelo ora apresentado, utiliza-se uma tabela $T_{w\times n}$ de palavras código, em que w é o número de palavras código que define um conjunto de classes $c_\ell, 1 \leq \ell \leq M$, podendo ter mais de uma palavra código para a mesma classe.

No monitoramento, o modelo adota uma combinação de N detectores (em substituição aos classificadores binários $f_j, 1 \leq j \leq N$), que são instalados em pontos estratégicos da rede e em servidores críticos. Os detectores (ver definição no Capítulo 2) coletam as informações em *batch* ou em tempo real, dependendo do ambiente e da política de segurança da organização. Os fatores que mais influenciam nesta escolha são o nível de risco associado, o custo computacional e os recursos disponíveis, bem como o papel que o próprio sistema de detecção deve desempenhar.

Os N detectores utilizados neste trabalho são responsáveis por monitorar as informações (aqui denominadas de atributos) contidas nos pacotes que trafegam pela rede e/ou informações armazenadas nos *logs* dos servidores e emitir um bit, com base em sua regra de decisão. Cada bit enviado pelos detectores é utilizado para compor os n símbolos da palavra recebida.

Devido à ocorrência de uma atividade maliciosa, a palavra recebida, que corresponde às saídas dos detectores, pode representar um padrão de assinatura de intrusão. Essa palavra é decodificada por meio de um algoritmo de decodificação algébrica para códigos BCH [24, 37] e, posteriormente, comparada com a tabela de palavras código $T_{w\times n}$, selecionada por meio do algoritmo genético. Deste modo, espera-se que a classe seja identificada.

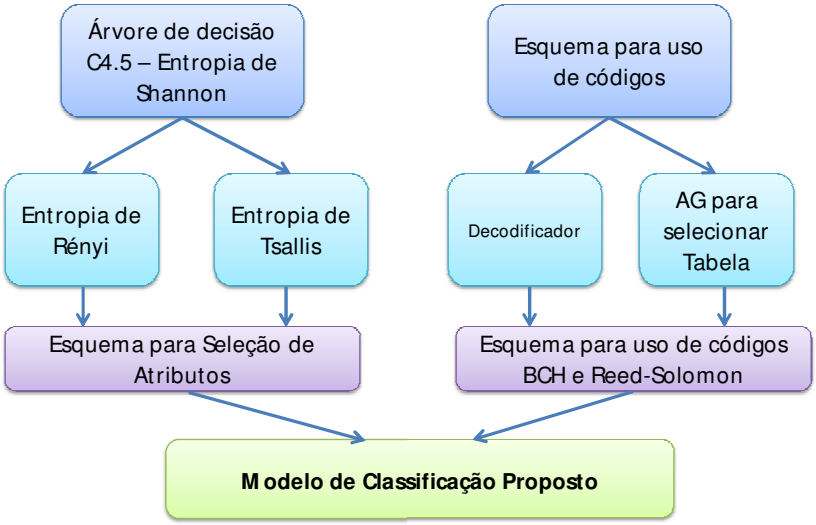
Dessa forma, por meio do uso do modelo de classificação proposto, é possível a identificação de ataques conhecidos e suas variações, bem como a detecção de ataques desconhecidos, por meio da sua capacidade de generalização, proveniente da capacidade de correção de erros do código utilizado. Com o objetivo de manter o sistema de detecção de intrusão sempre atualizado, o modelo proposto permite que sejam adicionados à tabela $T_{w\times n}$ novos padrões de ataques, na forma de palavras código, à medida que estes padrões forem surgindo.

No modelo proposto, existe uma relação direta entre o número de atributos e o tamanho da palavra código. Nesse contexto, serão apresentados esquemas para seleção de atributos, baseados nas medidas de informação de Rényi [40] e Tsallis [41, 42], com o objetivo de selecionar um subconjunto de atributos que possa incrementar a capacidade de um SDI em classificar o tráfego como normal ou como suspeito (categorizando o tipo de ataque).

Para a construção do modelo de classificação apresentado neste trabalho, foram seguidas as seguintes etapas, ilustradas na Figura 5.2:

- Selecionar os atributos a serem monitorados pelos detectores por meio do uso de árvores de decisão C4.5 baseadas em medidas de informação de Rényi [40] e Tsallis [41, 42];

Figura 5.2 Etapas para a construção do modelo proposto.



- Utilizar uma variante de algoritmo genético (AG) para selecionar a tabela de palavras código $T_{w \times n}$, garantindo que as palavras código pertençam ao código BCH escolhido e, ao mesmo tempo, selecionar as que melhor representem uma assinatura de intrusão;

- Fazer uso de decodificação algébrica, baseado nos conceitos de decodificação tradicional [43] e de decodificação por lista [44, 45, 121].

5.2 Etapa de Seleção de Atributos

Em domínios complexos, tais como a detecção de intrusão, grandes quantidades de dados referentes às atividades do sistema computacional são coletados da rede, gerando grandes arquivos de auditoria e de tráfego de rede, o que torna difícil a inspeção humana. Portanto, essas atividades precisam ser resumidas em um alto nível de abstração, em termos de atributos.

Contudo, é possível que alguns atributos possam conter correlações falsas, dificultando a tarefa de aprendizagem. Outrossim, outros atributos podem ser irrelevantes ou redundantes, os quais podem aumentar a complexidade e o tempo computacional, ocasionando um impacto negativo sobre a precisão do sistema de classificação.

Nesse contexto, de acordo com [122], definir qual subconjunto de todas as características (atributos) possíveis consegue predizer melhor atividades intrusivas é um dos principais problemas da detecção de intrusão.

Dessa forma, um ponto importante no esquema de classificação ora proposto está relacionado com o que precisa ser monitorado pelos detectores para determinar se uma conexão de rede é suspeita ou não. Assim, nesta seção são apresentados dois esquemas para seleção de atributos, com o objetivo de obter um subconjunto de atributos que incremente a capacidade de um SDI para classificar o tráfego como normal ou como suspeito (categorizando o tipo de

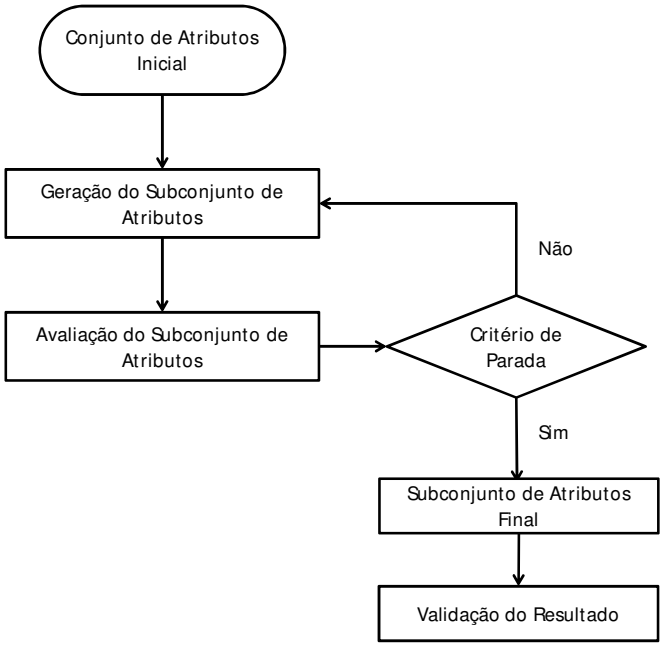
ataque). Cada atributo $A_j, 1 \leq j \leq N$ selecionado é utilizado como ponto de observação dos detectores de rede e de *host*.

5.2.1 Conceitos básicos sobre seleção de atributos

Dado um conjunto de dados, que pode ser descrito usando um conjunto $\aleph$ de atributos, o objetivo do processo de seleção de atributos é encontrar um número mínimo de N atributos relevantes que descreve o conjunto de dados, definidos de tal forma que espaço de atributos é reduzido de acordo com algum critério [12].

O processo de seleção de atributos consiste, basicamente em: buscar por um subconjunto de atributos e avaliar o subconjunto de atributos selecionado [123], conforme ilustrado na Figura 5.3.

Figura 5.3 Processo de Seleção de Atributos [123], adaptado pela autora.



Na atividade de busca dos subconjuntos de atributos, seleciona-se um subconjunto de atributos relevantes, candidatos para a avaliação, com o apoio de um algoritmo de busca. Para um conjunto de $\aleph$ atributos, existem $2^{\aleph}$ subconjuntos possíveis. Contudo, uma busca exaustiva por um subconjunto de atributos ideal pode ser inviável, especialmente quando $\aleph$ e o número de classes do problema aumenta [123].

Portanto, os métodos heurísticos que exploram o espaço de busca reduzido são comumente usados para a seleção subconjunto de atributos. Estes métodos são tipicamente gulosos no sentido de que, durante a pesquisa por meio do espaço de atributos, fazem a melhor escolha de atributo a cada iteração. Sua estratégia é fazer a escolha ótima local, com o objetivo de en-

contrar uma solução ótima global. Na prática, esses métodos gulosos são eficazes e podem se aproximar de soluções ótimas [12].

No processo de avaliação do subconjunto de atributos selecionados, cada subconjunto de atributos candidato é comparado com um subconjunto de atributos anterior. De acordo com um critério de avaliação (tais como, informação, distância, precisão), se o novo subconjunto se tornar melhor, o mesmo substituirá o subconjunto anterior.

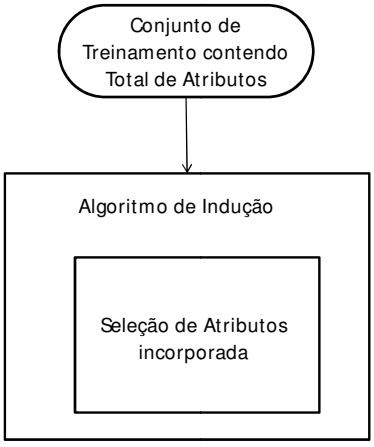
O processo de geração e avaliação de subconjuntos é realizado repetidamente até satisfazer um determinado critério de parada. O critério de parada pode ocorrer quando o objetivo foi alcançado, quando a busca termina ou no momento em que um subconjunto é considerado ótimo.

Para validação dos resultados, pode-se obter estimativas de desempenho, utilizando um classificador induzido com o conjunto de treinamento contendo $\aleph$ atributos e induzido com o mesmo conjunto de treinamento contendo o subconjunto de atributos selecionado.

As diferentes abordagens de seleção de atributos são geralmente classificadas como [124, 125]:

- ***Embedded***: a seleção de atributos está diretamente incorporada no algoritmo responsável pela indução do modelo de classificação, conforme ilustrado na Figura 5.4. Na abordagem *embedded* (embutidos), o subconjunto de atributos é selecionado no próprio processo de construção do modelo de classificação, durante a fase de treinamento, e são geralmente específicos para um dado algoritmo de classificação. Por exemplo, as árvores de decisão ID3 e C4.5 utilizam esta abordagem [124, 125].

Figura 5.4 Abordagem *Embedded* [126], adaptado pela autora.

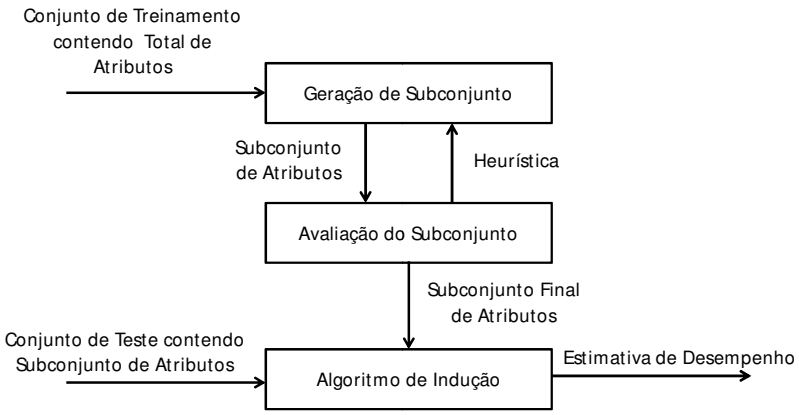


- **filtro**: Nesta abordagem, o subconjunto de atributos é escolhido independentemente do algoritmo de indução, que receberá somente o subconjunto de atributos selecionados pelo filtro, examinando as características intrínsecas dos dados de treinamento e estimando a

qualidade de cada atributo, considerando apenas os dados disponíveis, conforme ilustrado na Figura 5.5.

O nome filtro deriva da ideia de que os atributos irrelevantes são filtrados da base de dados antes da aplicação do algoritmo de classificação [124]. Algumas técnicas do tipo filtro avaliam os atributos individualmente e escolhem um subconjunto selecionando os N atributos mais bem avaliados. Esse é o procedimento utilizado, por exemplo, pelo esquema *Information Gain Attribute Ranking* [127].

Figura 5.5 Abordagem Filtro [128], adaptado pela autora.



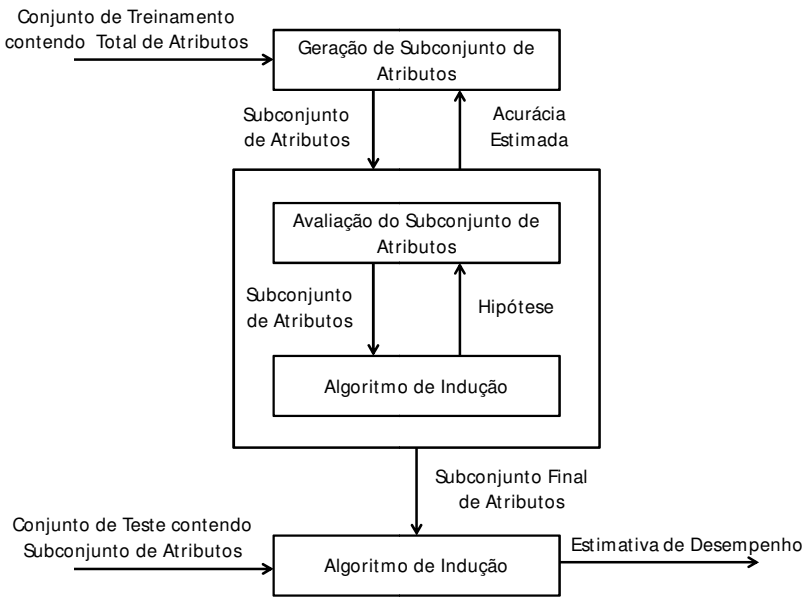
- ***Wrapper***: Nesta abordagem, o desempenho do subconjunto atributos é avaliado pela aplicação de um algoritmo de indução predeterminado no subconjunto de atributos selecionado. Desse modo, para cada subconjunto de atributos novo, o modelo *wrapper* induz um classificador e utiliza o seu desempenho para avaliar e determinar qual subconjunto de atributos deve ser selecionado, conforme ilustrado na Figura 5.6.

  A abordagem *wrapper* tende a encontrar os melhores atributos adequados, considerando o algoritmo de classificação pré-determinado, resultando em um melhor desempenho de aprendizagem, mas também tende a ser computacionalmente mais cara do que a abordagem filtro [129].

5.2.2 Esquema de seleção de atributos por indução de árvore de decisão C4.5

Com o objetivo de selecionar um subconjunto de atributos que possa incrementar a capacidade de um SDI para classificar o tráfego como normal ou como suspeito (categorizando o tipo de ataque), utiliza-se a indução de árvore de decisão C4.5 [87]. Nesta abordagem, todos os

Figura 5.6 Abordagem Wrapper [129], adaptado pela autora.



atributos que não aparecem na árvore são considerados irrelevantes, isto é, os atributos que correspondem aos nós da árvore, pertencem ao subconjunto de atributos ótimo, conforme ilustrado na Figura 5.7.

Na abordagem de seleção de atributos aqui proposta, são apresentados dois esquemas que utilizam a indução de árvore de decisão C4.5 para selecionar um subconjunto de atributos, com base nas medidas de informação de Rényi [40] e nas medidas de informação de Tsallis [41, 42] (medidas abordadas no Capítulo 3 e Apêndice C). Os esquemas propostos são ilustrados na Figura 5.8.

No modelo de classificação aqui proposto, a seleção de atributos ocorrerá antes da seleção da tabela $T_{w\times n}$, e, portanto, enquadra-se melhor como uma abordagem filtro.

Foi verificado empiricamente que aplicando as medidas de Rényi [40] e Tsallis [41,42], a seleção de atributos diminui significativamente o número de atributos e a dimensionalidade dos dados, o que resulta em menor tempo de processamento em comparação com a situação em que todos os atributos da base de dados original foram utilizados (com 41 atributos). Também foi verificado que o desempenho dos classificadores testados no conjunto de dados KDD 99 (ver artigos nos anexos desta tese) pode permanecer o mesmo ou até tornar-se melhor em comparação com o uso do conjunto de dados completo.

Nesse contexto, aplicando a abordagem de seleção de atributos proposta neste trabalho, consegue-se determinar os atributos que serão utilizados como pontos de observação dos detectores e, como consequência, determina-se o tamanho da palavra código.

Definido o esquema para selecionar os atributos que serão observados pelos detectores, na próxima seção é apresentado um esquema para selecionar a tabela de palavras código que representam assinaturas de intrusão, envolvendo o uso de um algoritmo genético.

Figura 5.7 Indução de Árvore de Decisão aplicada na Seleção de Atributos.

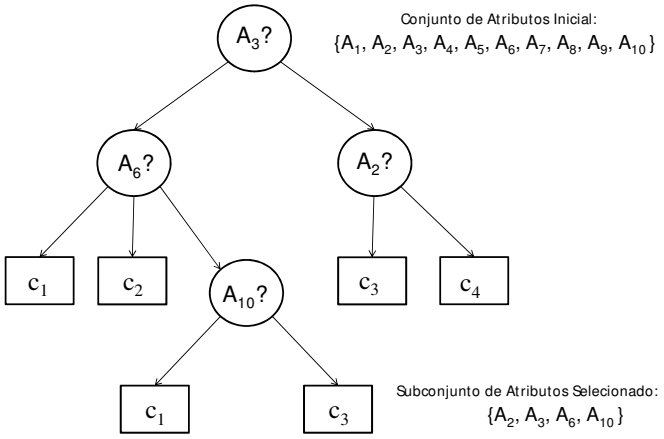
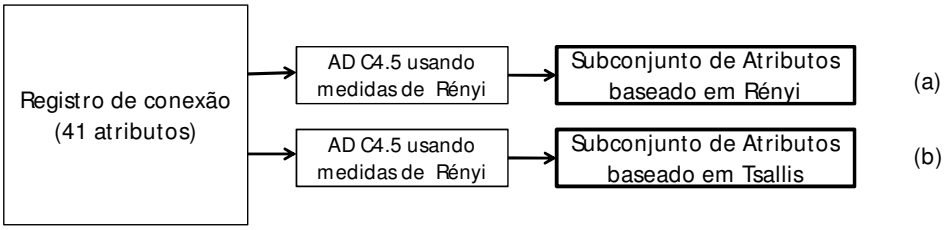


Figura 5.8 Esquemas para selecionar atributos: (a) usando as medidas de Rényi e (b) usando as medidas de Tsallis.



5.3 Etapa de Seleção da Tabela de Palavras Código

As palavras código que formam a tabela de palavras código $T_{w\times n}$, formam um subcódigo de um código BCH (binário) [130, 131] ou de um código Reed-Solomon (BCH não binário) [120]. Este subcódigo não é necessariamente linear, mas a distância entre as palavras código é limitada pela distância de *Hamming* mínima do código, podendo até aumentar, enquanto que para um código sem estrutura podem ocorrer valores de distância menores.

Normalmente, o número de atributos a serem observados é muito menor que a quantidade de palavras código disponíveis e isto permite que haja diferentes combinações para a tabela de palavras código. Portanto, é necessário selecionar a mais adequada, ou seja, realizar a busca por um subcódigo dentro de um código BCH. Para esta finalidade, propõe-se o uso de uma variante de algoritmo genético (AG), descrito a seguir.

5.3.1 Algoritmo genético para seleção da tabela de palavras código

A seleção da tabela de palavras código para a abordagem aqui apresentada, será realizada por uma variante de algoritmo genético multiobjetivo para garantir que as palavras código

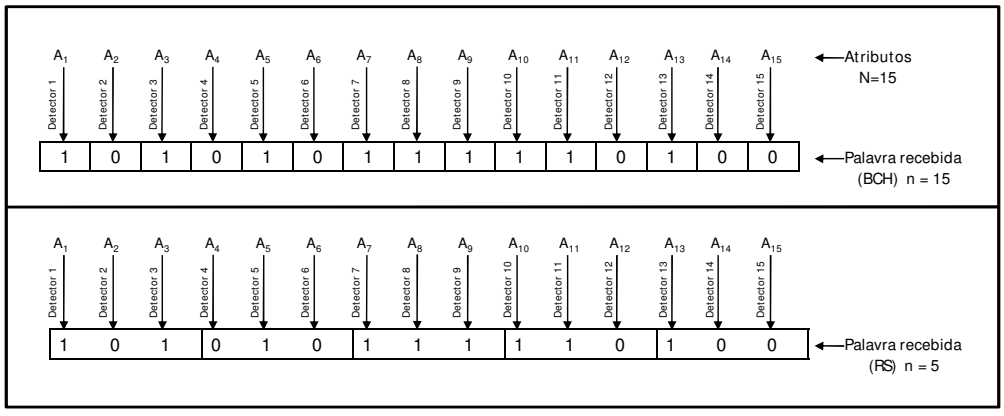
selecionadas pertençam ao código BCH escolhido e sejam as melhores representantes de uma determinada classe.

Os algoritmos genéticos, revisados no Apêndice F, são algoritmos de busca e otimização, com princípios embasados na genética e na teoria da evolução natural. Parte-se de uma população inicial de possíveis soluções para o problema a ser resolvido, as quais são referenciadas como indivíduos. Para simplificar a notação, no esquema ora apresentado, cada indivíduo é denotado por i. A cada geração, uma nova população é produzida por meio de um processo evolutivo (seleção, cruzamento e mutação) até evoluir para uma solução ótima ou sub-ótima.

Como as palavras recebidas são formadas a partir da observação dos N atributos, foi verificado empiricamente que a ordem destes atributos influencia diretamente na escolha da tabela $T_{w\times n}$. No caso de um código BCH, cada atributo corresponde a um símbolo da palavra, ou seja, $N = n$ (ver Figura 5.9 (a)). Já para o caso de um código RS, com símbolos em sequências de $s-$bits, um grupo de s atributos representa um símbolo da palavra, e neste caso $N > n$ (ver Figura 5.9 (b)), podendo ocasionar em um código modificado (ver Apêndice D).

A definição da ordem dos atributos pode ser reduzida a um problema de permutação, no qual se ordena um vetor de atributos. Busca-se a ordem de um conjunto de tributos que resulte em uma tabela $T_{w\times n}$ com bom desempenho na detecção de intrusão. Por esse motivo, a representação dos indivíduos e a definição dos operadores genéticos empregados no AG foram adaptadas a partir das estratégias para a solução do problema do caixeiro viajante [132] apresentadas no Apêndice F.

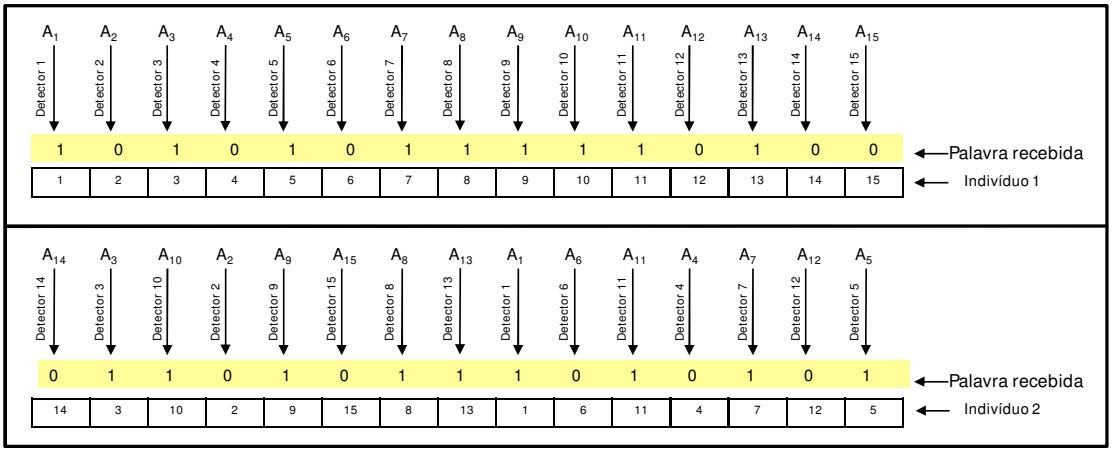
Figura 5.9 Formação das palavras recebidas: (a) usando um código BCH e (b) usando um código RS.



Assim, cada indivíduo é representado por um vetor com índices de 1 a N, indicando a ordem dos atributos, conforme ilustrado na Figura 5.10. Com essa representação, apesar de não ser intuitiva, reduz-se o custo computacional com o armazenamento de tabelas. Contudo, a avaliação das possíveis soluções do AG multiobjetivo engloba o poder preditivo da tabela $T_{w\times n}$ de palavras código de um código $BCH(n, k, d)$ ou $RS(n, k, d)$ em classificar tipos de

intrusões, bem como a ordem dos N atributos selecionados na etapa de seleção de atributos que irão compor os n símbolos da palavra recebida.

Figura 5.10 Ilustração de dois possíveis indivíduos para o problema proposto.



No AG ilustrado na Figura 5.11, assim como no AG básico, cada ciclo dos passos 3 a 10 é denominado uma geração g e, estes passos, em conjunto com os passos 1 e 2, constituem uma iteração do algoritmo.

Figura 5.11 Algoritmo Genético para busca dentro de um código BCH ou RS.

ENTRADA: BD de treinamento, BD de teste, N, $BCH(n,k,d)$ ou $RS(n,k,d)$
SAÍDA: tabela ótima ou subótima $T_{w\times n}$, vetor $OA[N]$
INÍCIO
Passo 1: $g \leftarrow 0$;
Passo 2: Gerar população $P(g)$ de indivíduos inicial aleatória;
Passo 3: Avaliar a aptidão dos indivíduos em $P(g)$; Passo 4: **Repita**
Passo 5: Selecionar indivíduos de $P(g)$ para reprodução por roleta;
Passo 6: Aplicar cruzamento PMX aos indivíduos do passo 5 e inserir indivíduos obtidos em $P(g)$;
Passo 7: Aplicar o operador de mutação *Order Based Mutation* a $P(g)$;
Passo 8: Voltar ao Passo 3;
Passo 9: $g \leftarrow g + 1$;
Passo 10: **Até que** (critério de parada seja atingido)
FIM

Na etapa de avaliação de cada indivíduo i em $P(g)$ (passo 3), são aplicadas as etapas ilustradas na Figura 5.12.

Conforme ilustrado na Figura 5.12, inicialmente, as bases de dados de treinamento T e de teste D são reordenadas de acordo cada indivíduo i, conforme ilustrado na Figura 5.10. A definição da base de dados de treinamento e teste é discutida na Seção 6.1. Em seguida, é utilizado um algoritmo de decodificação algébrica [24], baseado no algoritmo de *Berlekamp-*

Figura 5.12 Etapas para avaliação de cada indivíduo $\imath$ em $P(g)$

1. Ordenar BD de treinamento T, de acordo com indivíduo;
2. Ordenar BD de teste D, de acordo com indivíduo;
3. Receber BD de treinamento de palavras recebidas;
4. Decodificar palavras recebidas, via BMA;
5. Gerar $T'_{w\times n}$;
6. Receber BD de teste de palavras recebidas;
7. Decodificar palavras recebidas;
8. $ap(\imath) \leftarrow$ Aval $T'_{w\times n}$;
9. **Se** $ap(\imath) > fit$ **Então**
   $fit \leftarrow ap(\imath)$;
   $OA \leftarrow$ indivíduo $\imath$;
   $T_{w\times n} \leftarrow T'_{w\times n}$;
10. **Fim Se**

Massey (BMA) [133] descrito no Apêndice D para gerar a tabela $T'_{w\times n}$, com base nas instâncias da base de dados T.

A aptidão dos indivíduos é realizada por meio da avaliação das tabelas de palavras código $T'_{w\times n}$ geradas, com base na ordem dos N atributos. O valor da aptidão do indivíduo é calculado por meio da função objetivo representada na Equação (5.1).

$$ap(\imath) = \text{Aval } T'_{w\times n}. \tag{5.1}$$

Aval $T'_{w\times n}$ refere-se ao percentual médio de acerto obtido pelo classificador multiclasses, com base na tabela de palavras código $T'_{w\times n}$ em uma base de dados de teste. O processo de avaliação de desempenho consiste basicamente em decodificar as instâncias de uma base de dados de teste de palavras recebidas D, por meio do esquema de decodificação algébrica que será abordado na Seção 5.4, e em seguida, comparar cada instância decodificada com a tabela $T'_{w\times n}$. O AG implementado deve buscar soluções que maximizem $ap(\imath)$.

Dessa forma, privilegia-se a adaptação das soluções à resolução correta do problema proposto. Contudo, caso mais de uma tabela apresente a mesma medida de desempenho $ap(\imath)$, o critério de desempate usado consiste em escolher a tabela com menor número de linhas. Este critério segue o princípio de Occam [85], que orienta que entre várias hipóteses com o mesmo desempenho, deve-se dar preferência à mais simples.

São considerados como erros de classificação e estão embutidos na medida de desempenho $ap(\imath)$ as seguintes situações:

- Quando uma palavra recebida, que representa um padrão de intrusão, não está na região de decodificação, gerando uma falha de decodificação (ocasionando o problema de completude, abordado no Apêndice B). Adota-se, nesta situação, que a palavra recebida é uma atividade normal do sistema e não será classificada como intrusão.

- Quando a palavra recebida, que representa um padrão de intrusão, não corresponde a nenhuma linha da tabela gerada $T'_{w\times n}$. Neste caso, considera-se que a palavra corresponde a uma atividade normal do sistema.

- As classificações desconhecidas, que ocorrem quando duas ou mais linhas da tabela são iguais, ou seja, quando intrusões diferentes correspondem à mesma palavra código.

A partir da população avaliada $P(g)$, aplica-se um mecanismo para a seleção de indivíduos (passo 5) que passarão para a fase de reprodução, com probabilidade p_i, produzindo descendentes que formarão uma nova população $P(g)$. Existem diversos métodos de seleção, sendo que no algoritmo ora apresentado, foi utilizado o método da roleta, abordado no Apêndice F.

Na reprodução dos indivíduos selecionados (passo 6), busca-se combinar as suas características ou genes na obtenção dos descendentes, representando o conceito de hereditariedade. Essa combinação é realizada pela aplicação de um operador genético denominado cruzamento. O operador de cruzamento aplicado no AG utilizado neste trabalho é o do tipo PMX (*Partially Mapped Crossover*), abordado no Apêndice F, com uma taxa pré-fixada p_c, em que $0,6 \leq p_c \leq 0,99$.

Por meio da combinação produzida pelo cruzamento (passo 7), obtém-se também uma variabilidade nas novas soluções. O conceito de variabilidade genética é reforçado pela aplicação de um operador unário denominado mutação. A mutação altera genes dos indivíduos da nova população gerada na etapa de cruzamento. O método de mutação utilizado no algoritmo genético é a mutação *Order Based Mutation*, abordada no Apêndice F, segundo uma taxa p_m, $0,001 \leq p_m \leq 0,1$.

No passo 10 do algoritmo, ilustrado na Figura 5.11, pode-se utilizar diversos critérios de parada. O critério escolhido neste trabalho foi o número máximo de gerações. No final da execução do algoritmo é dada a tabela ótima ou subótima $T_{w\times n}$ e o vetor $OA[N]$, contendo a ordem dos atributos.

A próxima etapa da construção do modelo consiste na descrição do esquema de decodificação adotado.

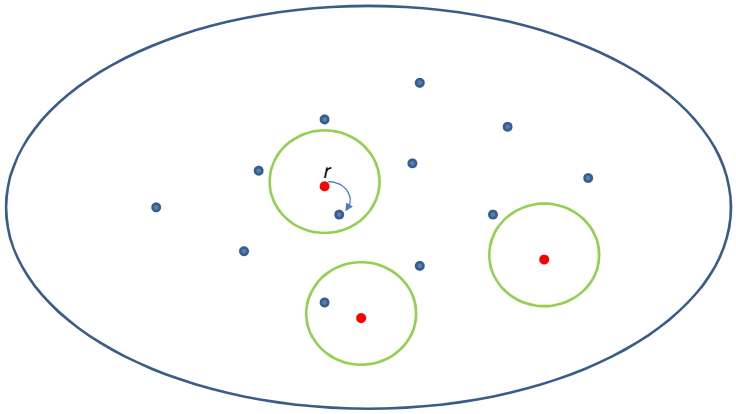
5.4 Etapa de Decodificação

Considerando que o número de tipos de ataques em uma rede de computadores, bem como o número de atributos a ser observado, podem ser bastante elevado, o modelo proposto prevê a utilização de decodificadores algébricos no processo de classificação, sendo este um processo menos exaustivo do que os utilizados por códigos que não possuem estrutura, cuja decodificação só pode ser realizada por comparação da palavra recebida com as palavras do código.

5.4.1 Decodificação tradicional

Tradicionalmente, o processo de decodificação de um código consiste na busca por uma única palavra código mais próxima da palavra recebida r na esfera de *Hamming* de raio t, caso exista, no limite de correção de erros $t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$, conforme ilustrado na Figura 5.13 (ver Apêndice D). Se até t erros ocorrem, então a palavra recebida será decodificada corretamente, senão haverá um erro de decodificação.

Figura 5.13 Esquema de Decodificação Tradicional $t = \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$.



O primeiro algoritmo de decodificação para códigos BCH foi descoberto por Peterson em 1960 [134]. Desde então, o algoritmo de Peterson tem sido refinado por *Berlekamp* [135], *Massey* [43], *Chien* [136], *Forney* [137] e muitos outros.

Entre todos os algoritmos para a decodificação dos códigos BCH, o algoritmo iterativo de *Berlekamp-Massey* (desenvolvido por *Berlekamp* em 1968 e simplificado por *Massey* em 1969) e o algoritmo de busca de *Chien*, revisados no Apêndice D, são hoje em dia de uso generalizado e foram utilizados no AG para selecionar as tabelas e na etapa de decodificação do modelo proposto.

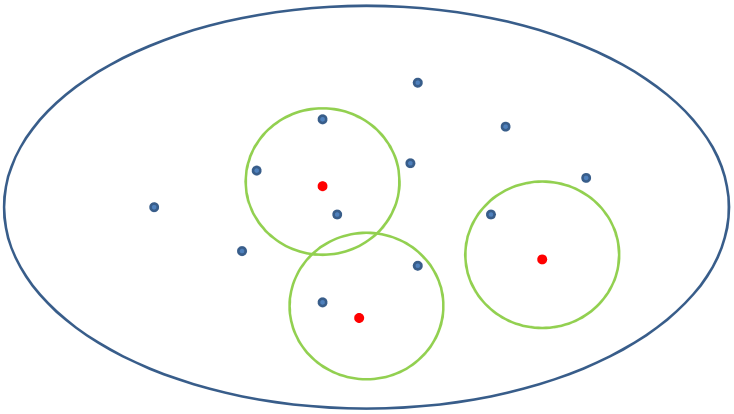
5.4.2 Decodificação por lista

O processo de decodificação descrito anteriormente não é suficiente para canais ruidosos, em que a palavra código mais próxima nem sempre é a que foi transmitida, devido ao número de erros ocorrido. Nesse contexto, uma noção alternativa de decodificação proposta independentemente por Elias em 1957 [44] e Wozencraft em 1958 [45], tem sido amplamente pesquisada, chamada de **decodificação por lista**, pois torna possível recuperar as informações com erros além do limite tradicional de correção de erros do código.

A decodificação por lista é um relaxamento da decodificação única (ver Apêndice D), convencionalmente utilizada, que permite que uma lista de palavras é obtida ao invés de uma única palavra código.

O decodificador gera, como saída, uma lista de palavras que é diferente de uma palavra recebida r em um determinada distância de *Hamming*, conforme ilustrado na Figura 5.14. Neste caso, o raio de decodificação t é aumentado ($\tau > t$), e é possível que dentro da esfera de decodificação tenha-se mais de uma palavra código. Por conseguinte, haverá erro de decodificação somente se a palavra de informação transmitida não estiver presente na referida lista.

Figura 5.14 Esquema de Decodificação por Lista $\tau > t$.



Devido à sua variedade de aplicação, os algoritmos de decodificação algébricos para os códigos Reed-Solomon são objeto de estudo de muitos pesquisadores.

Nesse contexto, em 1997, utilizando-se da consequência do teorema de Bezout [138] e usando uma generalização do algoritmo Welch-Berlekamp [139], Sudan [140] demonstrou que era possível fazer a decodificação de códigos Reed-Solomon, ultrapassando a capacidade de correção clássica do canal $t = (n-k)/2$, utilizando um método de decodificação por lista, abordando-o como um problema de interpolação de duas variáveis $Q(x, y) \in GF(q)$.

Assim, para um código $RS(n, K)$, o algoritmo de Sudan [140] produz todas as palavras código cuja distância do vetor recebido não exceda $n - \sqrt{2kn}$. Todavia, o método de decodificação de Sudan [140] é assintoticamente melhor que a decodificação única de até $(n-k)/2$ erros, se a taxa do código (k/n) for menor do que $1/3$, ou seja, o código teria mais redundância do que informação propriamente dita, não representando, portanto, uma grande contribuição do ponto de vista prático.

Em 1999, Guruswami e Sudan [141] apresentaram um aprimoramento do método de decodificação proposto no algoritmo de Sudan [140], demostrando que poderiam ser corrigidos ainda mais erros se fizessem com que a interpolação em cada ponto ocorresse não apenas uma vez, como ocorria anteriormente, e sim $\jmath$ vezes, em que $\jmath$ é uma variável inteira arbitrária.

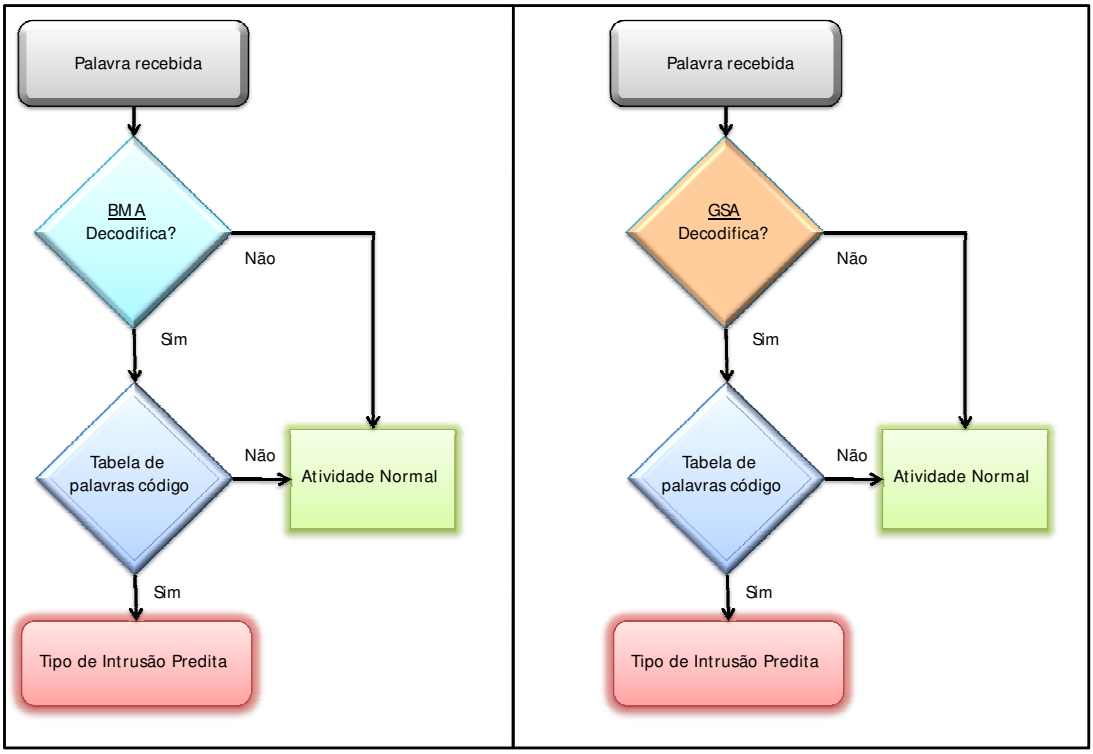
Para $\jmath \to \infty$, o algoritmo de decodificação por lista de *Guruswami-Sudan*(GSA) [141] corrige até $\tau = \left\lfloor n - \sqrt{nk} \right\rfloor$ erros, e, desta vez, para qualquer taxa k/n, superando a capacidade de correção do algoritmo de *Berlekamp-Massey* [43] (ver Apêndice D).

5.4.3 Esquema de decodificação

Com base nas características dos métodos de decodificação algébrica abordado anteriormente, a etapa de decodificação do modelo proposto é composta por dois esquemas de decodificação, ilustrado na Figura 5.15, a saber:

- Um esquema de decodificação tradicional para códigos BCH (Figura 5.15 (a)), baseado no algoritmo de *Berlekamp-Massey* (BMA) [43].
- Um esquema de decodificação para códigos Reed-Solomon (Figura 5.15 (b)), baseado no algoritmo de decodificação por lista de *Guruswami-Sudan* (GSA) [141], revisado no Apêndice E.

Figura 5.15 Esquemas de decodificação: (a) para códigos BCH e RS (b) para códigos RS



A abordagem de decodificação por lista foi utilizada no modelo de classificação (ver Figura 5.15) para minimizar a ocorrência de falhas de decodificação, as quais podem gerar erros de classificação, uma vez que estas são consideradas como atividade normais do sistema, e também para permitir que variações de um tipo de ataque pudesse ser identificado, mesmo que a palavra recebida, contendo este padrão, estivesse fora da região de decodificação.

Os algoritmos de decodificação tradicional dos códigos BCH e Reed-Solomon, implementados no AG e nos esquemas de decodificação, são versões modificadas dos códigos fontes disponibilizados por [133]. O algoritmo de decodificação por lista de Guruswami-Sudan é uma versão modificada do código fonte disponibilizado por [142, 143].

5.5 Considerações Finais

Neste capítulo, buscou-se encontrar alternativas mais eficientes, baseadas na teoria da informação e códigos corretores de erros, para gerarem modelos para serem aplicados na detecção de intrusos em redes de computadores.

Inicialmente, abordou-se a descrição geral do modelo proposto que associa uma assinatura de intrusão a uma palavra código de um código BCH ou Reed-Solomon. O tamanho da palavra código é definida segundo critérios de seleção de atributos baseados nas medidas de entropia de Rényi e Tsallis. As palavras código que formam a tabela de palavras código são selecionadas por meio de um AG. Por fim, o esquema de decodificação adotado no modelo utiliza decodificadores algébricos tradicionais e por lista.

Alguns resultados da aplicação de versões prévias dos algoritmos de árvore de decisão C4.5 aplicados na seleção de atributos foram apresentados em [144–146] e encontram-se nos anexos deta tese.

Com base no modelo ora proposto, no próximo capítulo são apresentados e analisados os resultados experimentais, baseados em um exemplo de aplicação do modelo de classificação aqui apresentado.

CAPÍTULO 6

Resultados

Neste capítulo, as estratégias para a obtenção de um classificador multiclasses baseada em códigos corretores de erros, apresentadas no capítulo anterior são avaliadas por meio de análises experimentais, demonstrando a viabilidade do esquema apresentado. Inicialmente será descrita a base de dados utilizada nos experimentos. Em seguida é apresentada a metodologia utilizada na condução dos experimentos. Por fim, os resultados obtidos nos experimentos são apresentados e analisados.

6.1 Base de dados KDD Cup 99

Um ponto importante para o desenvolvimento do modelo de detecção de intrusão é encontrar uma fonte de dados que possa ser utilizada como conjunto de treinamento e teste pelos classificadores. Nesse contexto, nos experimentos realizados foi utilizado um subconjunto da base de dados disponibilizada na competição internacional de mineração de dados, *Knowledge Discovery and Data Mining Competition - KDD Cup 99* [118] contendo um conjunto padrão de dados para serem auditados, incluindo uma grande variedade de intrusões simuladas durante nove semanas em uma rede militar de computadores.

A distribuição dos ataques na base de dados KDD CUP 99 é desbalanceada. A maioria dos ataques são das categorias DoS e *Probing*. Os ataques mais perigosos, como U2R e R2L, são representados por um número reduzido de instâncias. Contudo, a base de dados KDD CUP 99 tem a capacidade de permitir aos pesquisadores comparar diferentes técnicas de detecção de intrusão em um conjunto de dados comum.

Na base de dados KDD CUP 99 original há 494.021 instâncias, em que 97.278 indicam o tráfego de rede normal e 396.744 registros contendo 22 tipos de ataques identificados, os quais se enquadram em uma das quatro categorias principais de ataques:

- **User to Root (U2R) - Usuário para Superusuário** - quando uma pessoa que possui acesso autorizado como um usuário e tenta obter acesso como superusuário (*root*);

- **Remote to Local (R2L) - Remoto para Local** - quando uma pessoa, em uma máquina remota, tenta obter acesso ao servidor local;
- **Probing - Reconhecimento** - um intruso varre uma rede em busca de informação ou para encontrar vulnerabilidades conhecidas para explorar;
- **DOS - Negação de Serviço** - O objetivo do invasor é causar a indisponibilidade do serviço alvo, seja por meio do consumo de recursos computacionais como memória, processador ou rede.

Na base de dados KDD CUP 99, cada instância (ou conexão de rede) contém 41 atributos para caracterizar as assinaturas de ataques, conforme ilustrado na Tabela 6.1. Esses atributos são divididos em três grupos: atributos básicos de conexões TCP individuais, atributos de tráfego e atributos com conteúdos relacionados ao domínio de conhecimento.

6.2 Exemplo de Aplicação do Modelo Proposto

Nesta seção são descritos os procedimentos adotados para a realização dos experimentos, os quais englobam as etapas de pré-processamento da base de dados KDD Cup 99, a aplicação das estratégias de seleção de atributos, a aplicação do AG para selecionar a tabela de palavras código e a análise de seus resultados. A definição desse processo tem como objetivo validar o esquema de detecção de intrusão por meio de um exemplo de aplicação do modelo proposto.

6.2.1 Pré-processamento da base de dados KDD cup 99

Normalmente, um subconjunto do tráfego de rede é necessário ser coletado com antecedência para a concepção de sistemas de detecção de intrusão. No entanto, é difícil reunir todas as informações de ataque, porque, em cenários práticos, os intrusos constantemente desenvolvem novas variações de ataques para explorar as vulnerabilidades de segurança das organizações.

Portanto, os dados coletados sempre envolvem incerteza quando apenas informações limitadas sobre as atividades intrusivas estão disponíveis. Assim, a fim de simular o problema da incerteza existente no conjunto de dados KDD Cup 99 e diminuir o custo computacional sem compromissos com os resultados da pesquisa, quatro subconjuntos separados por categoria de ataque foram selecionados aleatoriamente a partir da base de dados original.

Os subconjuntos de dados selecionados contendo os tipos de ataques, organizados por categoria e seus respectivos números de instâncias (valores entre parênteses) podem ser visualizados na Tabela 6.2. No Apêndice H é feita uma descrição resumida desses ataques.

Como os subconjuntos gerados não possuem subconjuntos de treinamento T e teste D definidos, então foi utilizado o método de validação cruzada 10-*folds*, descrito no Apêndice B.

Tabela 6.1 Descrição dos Atributos da Base de Dados - KDD 99, adaptada de [118].

Nr.	Atributo	Descrição	Tipo
1	**duration**	tam da conexão em seg.	cont.
2	**protocol_type**	tipo do protocolo (tcp, udp, icmp)	disc.
3	**service**	serviço de rede no destino (http, ftp)	disc.
4	**flag**	normal ou erro de conexão	disc.
5	**src_bytes**	num. de bytes da origem para destino	cont.
6	**dst_bytes**	num. de bytes do destino para origem	cont.
7	**land**	1 se conexão é de/para memso host/porta; 0 senão	disc.
8	**wrong_fragment**	num. de fragmentos errados	cont.
9	**urgent**	num de pacotes urgentes	cont.
10	**hot**	num. de indicadores *hot*	cont.
11	**num_failed_logins**	num de falhas de login	cont.
12	**logged_in**	1 logged in com sucesso; 0 senão	disc.
13	**num_compromised**	num. de condições comprometidas	cont.
14	**root_shell**	1 se root shell é obtido; 0 senão	disc.
15	**su_attempted**	1 se comando *su root*; 0 senão	disc.
16	**num_root**	num. de acessos *root*	cont.
17	**num_file_creations**	num. de operações de criação de arquivos	cont.
18	**num_shells**	num. de shell prompts	cont.
19	**num_access_files**	num. de operações a controle de acesso a arquivos	cont.
20	**num_outbound_cmds**	num. de comandos outbound em sessões *ftp*	cont.
21	**is_hot_login**	1 se login foi para a lista *hot*; 0 senão	disc.
22	**is_guest_login**	1 se *guest* login; 0 senão	disc.
23	**count**	num. de conexões para o mesmo host em 2 seg.	cont.
24	**srv_count**	de conexões para o mesmo serviço em 2 seg	cont.
25	**serror_rate**	% de conexões com erro *SYN*	cont.
26	**srv_serror_rate**	% de serviços com erro *SYN*	cont.
27	**rerror_rate**	% de conexões com erro *REJ*	cont.
28	**srv_rerror_rate**	% de serviços com erro *REJ*	cont.
29	**same_srv_rate**	% de conexões para o mesmo serviço	cont.
30	**diff_srv_rate**	% de conexões para diferentes serviços	cont.
31	**srv_diff_host_rate**	% of connections to different hosts	cont.
32	**dst_host_count**	contagem de conexões tendo o mesmo host destino	cont.
33	**dst_host_srv_count**	contagem de conexões tendo o mesmo host destino e mesmo serviço	cont.
34	**dst_host_same_srv_rate**	% de conexões tendo o mesmo host destino e mesmo serviço	cont.
35	**dst_host_diff_srv_rate**	% de diferentes serviço para o host atual	cont.
36	**dst_host_same_src_port_rate**	% de conexões tendo o mesmo host dest e mesma porta	cont.
37	**dst_host_srv_diff_host_rate**	% de conexões tendo o mesmo serviço vindo de dif. hosts	cont.
38	**dst_host_serror_rate**	% de conexões para o host atual com erro *S0*	cont.
39	**dst_host_srv_serror_rate**	% de conexões para o host atual e espec. serviço com erro *S0*	cont.
40	**dst_host_rerror_rate**	% de conexões para o host atual com erro *RST*	cont.
41	**dst_host_srv_rerror_rate**	% de conexões para o host atual e espec. serviço com erro *RST*	cont.

Tabela 6.2 Relação de ataques por categorias.

DoS	PROBING	R2L	U2R
back (1026)	ipsweep (586)	ftp-write (8)	buffer-overflow (21)
land (11)	nmap (151)	guess-passwd (53)	loadmodule (10)
neptune (10401)	portsweep (155)	imap (11)	perl (3)
pod (69)	satan (16)	multihop (11)	rootkit (7)
smurf (7669)	normal (1704)	phf (5)	normal (1676)
teardrop (15)		wareszclient (60)	
normal (2573)		warezmaster (20)	
		spy (4)	
		normal (1934)	

Os mesmos subconjuntos de treinamento e teste são utilizados em todos os experimentos para permitir comparações entre as diferentes estratégias avaliadas.

Os atributos na base de dados KDD Cup 99 [118] contêm dados de diversos tipos, com resolução variando significativamente em intervalos. Esses atributos foram utilizados sem necessitarem de formatação em diversos experimentos realizados. Entretanto, como os códigos BCH e Reed-Solomon não são capazes de processar dados em todos os formatos, uma etapa de pré-processamento se faz necessária para transformar os dados em um formato aceitável pelo código.

Assim, os atributos contínuos foram discretizados, com o objetivo de se obter uma base de dados contendo somente atributos com valores binários. Os métodos de discretização de atributos podem ser divididos em: supervisionados e não supervisionados [147].

Enquanto os métodos supervisionados utilizam informações referentes às classes das instâncias da base de dados para estimar os pontos de corte durante o processo de discretização de um atributo, os não supervisionados consideram somente os valores do atributo a serem discretizados. Alguns autores atribuem um desempenho melhor aos métodos de discretização de atributos supervisionados [147].

Na classe dos métodos supervisionados, várias propostas são encontradas na literatura científica. Em [147], é apresentado um algoritmo de discretização utilizando uma heurística recursiva de minimização de entropia, proposta por [148].

No algoritmo de Fayyad e Irani [148] é determinado um ponto de corte que divida um intervalo de valores em dois subintervalos. Esse ponto de corte, definido a partir da medida de informação mútua [84], será aquele que resultar em subintervalos com maior ganho de informação. Enquanto um determinado critério de parada não for alcançado, o procedimento é recursivamente aplicado aos subintervalos resultantes das iterações anteriores. Na heurística proposta por [148], o critério de parada adotado é o *Minimal Description Length Principle* [95, 149].

Pelo princípio do MDL, quando vários modelos competem pela explicação de um conjunto de dados, deve-se selecionar aquele modelo que melhor comprime os dados. Outrossim, a capacidade de compressão de um conjunto de dados está diretamente relacionada à descoberta

de regularidade nesse conjunto de dados, ou seja, quanto maior a compressão, maior o aprendizado sobre os dados [95, 149].

Dado um conjunto T de μ instâncias, contendo M classes, a discretização de um dado atributo $A_j, j = 1, ..., N$, pela heurística recursiva de minimização de entropia, é resumidamente descrita a seguir:

1. Cada valor v do atributo A_j é considerado como um possível limiar de intervalo, ou seja, um valor v pode dividir as instâncias de T em dois subconjuntos T_1 e T_2, satisfazendo as condições $A_j < v$ e $A_j \geq v$. Os subconjuntos T_1 e T_2 contêm M_1 e M_2 classes, respectivamente.

2. Sejam as variáveis aleatórias discretas C, C_1, C_2, A, têm-se que: os possíveis valores de C são $c_1, c_2, ...c_M$, $\{p_\ell = \Pr[C = c_\ell]\}_{\ell=1}^{M}$ e c_ℓ é a $\ell-$ésima classe da base de dados T; os possíveis valores para C_1 são $c_{1_1}, c_{1_2}, ...c_{1_{M_1}}$, $\{p_{1_\ell} = \Pr[C_1 = c_{1_\ell}]\}_{\ell=1}^{M_1}$ e c_{1_ℓ} é a $\ell-$ésima classe da base de dados T_1; os possíveis valores para C_2 são $c_{2_1}, c_{2_2}, ...c_{2_{M_2}}$, $\{p_{2_\ell} = \Pr[C_2 = c_{2_\ell}]\}_{\ell=1}^{M_2}$ e c_{2_ℓ} é a $\ell-$ésima classe da base de dados T_2; os possíveis valores para A são $v_1, v_2, ...v_\eta$, $p_\nu = \Pr[A = v_\nu]_{\ell=1}^{\eta}$, em que v_ν é o ν-ésimo valor do atributo A_j, para $j = 1, ..., N$, que está sendo testado e p_ν é a probabilidade do valor v_ν ocorrer na base de dados. Dado T, o ponto de corte v escolhido será aquele que maximiza a informação mútua $I(C, A)$ (Equação (C.1)) do particionamento, tal qual é calculado na indução de árvore de decisão C4.5 [83] (ver Capítulo 3).

3. O método é, em seguida, aplicado de forma recursiva para os dois subintervalos. O particionamento recursivo do conjunto T continuará enquanto o critério de parada não for alcançado. O critério de parada consiste em aceitar uma partição induzida por v se e somente se o custo de codificar a partição e as classes dos casos no intervalo induzido por v é menor do que o custo de codificar as classes dos casos antes divisão, ou seja, a partição induzida pelo ponto de corte v é aceita se somente se:

$$I(C; A) > \frac{\log_2(\mu - 1)}{\mu} + \frac{\log_2(3^M - 2) - [M.H(C) - M_1.H(C_1) - M_2.H(C_2)]}{\mu}. \tag{6.1}$$

Segundo [128, 147], comparando os efeitos da utilização de diversas técnicas de discretização, tem-se que que o método de discretização proposto por Fayyad e Irani [148], em geral, produz resultados com desempenho melhor. Por esta razão, este método de discretização é usado nos experimentos realizados neste trabalho de tese, por meio da ferramenta WEKA (*Waikato Environment for Knowledge Analysis*).

Os atributos discretos, tais como *service*, com 70 símbolos diferentes e os atributos discretizados foram mapeados para valores binários, também utilizando a ferramenta WEKA.

Todos os atributos originalmente binários, tais como o atributo *logged-in* (ver Tabela 6.1), permaneceram inalterados.

Um ponto importante é que quando há um elevado número de instâncias duplicadas no conjunto de treinamento, alguns algoritmos de aprendizado ficam polarizados para os registros mais frequentes. Por outro lado, os registros menos frequentes tais como os ataques U2R e R2L, que geralmente são mais prejudiciais para as redes, não são aprendidos pelos classificadores. Além do mais, a existência destas instâncias repetidas no conjunto de teste, faz com que os resultados de avaliações sejam polarizados pelos classificadores que têm melhor desempenho na detecção destes registros.

Para minimizar as situações supracitadas, as instâncias repetidas foram eliminadas. O número de instâncias por tipo de ataque, bem como as instâncias contendo tráfego normal, utilizadas nos experimentos envolvendo códigos corretores de erros, após a eliminação de redundância, estão relacionados nas Tabelas 6.3 e 6.4.

Tabela 6.3 Número de instâncias por categoria de ataques, usando a seleção de atributos por meio das medidas de informação de Rényi.

DoS	PROBING	R2L	U2R
back (46)	ipsweep (10)	ftp-write (5)	buffer-overflow (2)
land (7)	nmap (19)	guess-passwd (9)	loadmodule (4)
neptune (56)	portsweep (21)	imap (6)	perl (1)
pod (11)	satan (10)	multihop (5)	rootkit (3)
smurf (17)	normal (1704)	phf (2)	normal (17)
teardrop (7)		wareszclient (14)	
normal (479)		warezmaster (6)	
		spy (2)	
		normal (95)	

Tabela 6.4 Número de instâncias por categoria de ataques, usando a seleção de atributos por meio das medidas de informação de Tsallis.

DoS	PROBING	R2L	U2R
back (17)	ipsweep (8)	ftp-write (5)	buffer-overflow (21)
land (7)	nmap (11)	guess-passwd (8)	loadmodule (10)
neptune (22)	portsweep (24)	imap (4)	perl (3)
pod (12)	satan (11)	multihop (6)	rootkit (7)
smurf (15)	normal (273)	phf (3)	normal (1676)
teardrop (7)		wareszclient (16)	
normal (172)		warezmaster (6)	
		spy (1)	
		normal (95)	

6.2.2 Seleção dos Atributos

Considerando os resultados obtidos nos experimentos realizados e publicados em [16] (nos anexos deste trabalho), foi escolhido para ser analisado o melhor modelo de árvore de decisão C4.5 construído para cada categoria de ataque (DoS, Probing, R2L e U2R), segundo estimativas de desempenho Acc, TFN, TFP, índice *kappa* e profundidade da árvore (ver Apêndice B). Por exemplo, para a categoria DoS (negação de serviço), foi selecionado o modelo de árvore de decisão construído usando as medidas de Rényi com $\alpha = 0,5$, e pelas medidas de Tsallis com $\alpha = 1,2$.

Como as diferentes categorias de ataques podem ter diferentes subconjuntos de atributos, foram realizados dois experimentos para investigar qual subconjunto de atributos é mais adequado para a detecção de uma determinada categoria de ataque. Assim, depois de selecionar as árvores de decisão com melhor desempenho de classificação, dois subconjuntos de atributos foram selecionados, com base no esquema proposto (usando as medidas de Rényi e Tsallis), para cada conjunto de dados de treinamento (ver Tabela 6.2), de acordo com a categoria de ataque.

Os atributos selecionados, descritos pelo seu número correspondente na base de dados (ver Tabela 6.1), pelos diferentes esquemas propostos podem ser visualizados na Tabela 6.5. Um estudo sobre o uso das medidas de Rényi e Tsallis, comparando-as com as medidas de Shannon e uma abordagem conjunta que reúne as três medidas (abordagem *ensemble*) pode ser encontrado no Apêndice J e nos artigos [144–146], que se encontram nos anexos desta tese.

Tabela 6.5 Atributos selecionados usando árvore de decisão baseada nas medidas de informação de Rényi e Tsallis.

Categoria	Esquema	Atributos Selecionados
DoS	Rényi	2, 5, 7, 8, 23, 32, 35, 36, 39
	Tsallis	2, 5, 7, 8, 23, 26, 34, 39
Probing	Rényi	1, 2, 5, 6, 25, 30, 32, 33, 37, 38, 40
	Tsallis	1, 2, 4, 6, 23, 30, 31, 33, 37, 38, 40
R2L	Rényi	2, 5, 6, 10, 11, 12, 19, 33, 35, 37, 38, 39
	Tsallis	1, 3, 5, 6, 10, 11, 17, 19, 22, 37, 38
U2R	Rényi	13, 18, 32, 33, 36
	Tsallis	13, 16, 18, 32, 33

Fazendo uma pequena análise dos atributos selecionados, tem-se que o atributo 5 lidera como o mais importante para a detecção das categorias de ataques DoS, Probing e R2L. Entretanto, para diferentes categorias de ataques, alguns atributos não são os mesmos, porque diferentes tipos de ataques têm seu próprio padrão.

Por exemplo, ao se analisar as diferenças entre os dois vetores de ataque do tipo *smurf* (ver Apêndice H), da categoria DoS, ilustrado na Tabela 6.6 e comparando-os a dois vetores de tráfego normal, ilustrado na Tabela 6.7, ambos presentes na base de dados utilizada nos

experimentos, observa-se que nesse exemplo em particular a diferença entre dois vetores de ataque é muito menor do que a diferença entre os dois vetores normais.

Tabela 6.6 Ataque do tipo smurf.

Atributo	Reg. 410	Reg. 1537
protocol_type	icmp	icmp
service	ecr_i	ecr_i
flag	SF	SF
src_bytes	1032	1032
count	511	511
srv_count	511	511
same_srv_rate	1	1
dst_host_count	255	255
dst_host_srv_count	255	169

Geralmente, os vetores de ataque são muito semelhantes entre si, senão idênticos. Entretanto, segundo Dorothy Denning [48], um tráfego normal de rede é estatisticamente diferente de um ataque e isso foi observado nos exemplos analisados.

Por exemplo, o atributo 1 (*duration*) é uma importante característica para as categorias de ataques *Probing* e R2L , mas são irrelevantes para ataques do tipo DoS e U2R. O atributo 23 (*count*) é importante para identificar padrões de ataques do tipo DoS. Além disso, foi constatado nos experimentos que os atributos 20 e 21 não apresentam nenhuma variação na base de dados e, portanto, foram considerados irrelevantes para a detecção de intrusão.

Tabela 6.7 Tráfego normal.

Atributo	Reg. 410	Reg. 1537
service	http	http
flag	SF	SF
src_bytes	303	173
dst_bytes	1263	4522
logged_in	1	1
count	6	2
srv_count	7	2
same_srv_rate	1	1
srv_diff_host_rate	0.29	0
dst_host_count	255	2
dst_host_srv_count	255	255
dst_host_same_srv_rate	1	1
dst host same src port rate	0	0.5

O primeiro passo do experimento consistiu em realizar a seleção dos atributos, usando árvore de decisão C4.5 baseada em medidas de informação de Rényi e Tsallis. Os atributos mais relevantes foram determinados e estão apresentados na Tabela 6.5.

Os atributos selecionados nesta etapa foram utilizados na próxima etapa que consistiu em selecionar a tabela de palavras código $T_{w\times n}$, após serem discretizados pelo método de Fayyad e Irani [148] e transformados para binários.

6.2.3 Seleção da Tabela $T_{w\times n}$

No experimento realizado, foi utilizado um AG para encontrar as palavras código de um código BCH e de um código de Reed-Solomon que melhor se adequassem ao problema proposto, conforme algoritmo ilustrado na Figura 5.11, Capítulo 5.

Foram geradas tabelas contendo os atributos selecionados pelas medidas de Rényi e atributos selecionados pelas medidas de Tsallis, utilizando as base de dados apresentadas nas Tabelas 6.3 e 6.4, respectivamente.

Com o objetivo de adaptar a estrutura do código com o número de atributos selecionados, optou-se por utilizar, nesse experimento, códigos BCH binários modificados (ver Apêndice D) e códigos RS primitivos. Portanto, foram feitas simulações com AG, usando códigos estruturados tais como $BCH(60, 54, 3)$ e $RS(15, 13, 3)$.

Em todas as buscas realizadas pelo referido algoritmo, utilizou-se uma taxa de cruzamento $p_c = 0,9$ e uma taxa de mutação de $p_m = 0,1$. A condição de parada foi o número máximo de iterações, em geral 50 gerações, com uma população de 100 indivíduos.

Para realizar a avaliação de cada indivíduo, foi utilizada a função objetivo representada na Equação (5.1), Capítulo 5. Verificou-se o poder preditivo das palavras código selecionadas, na tarefa de detectar tipos de ataques, bem como identificar o tráfego normal numa rede de computadores.

6.2.4 Etapa de decodificação

Na etapa de decodificação, foram realizados três experimentos, a saber:

- Utilizando um decodificador para códigos BCH, por meio do algoritmo de Berlekamp-Massey (BCH_BMA);
- Utilizando um decodificador para códigos Reed-Solomon, por meio do algoritmo de Berlekamp-Massey (RS_BMA);
- Utilizando um decodificador por lista para códigos Reed-Solomon por meio algoritmo de Guruswami-Sudan (RS_GS).

Foi Verificado o poder preditivo das tabelas de palavras código selecionadas na etapa anterior, usando os esquemas de decodificação apresentados no Capítulo 5, na tarefa de detectar tipos de ataques, bem como identificar o tráfego normal numa rede de computadores, traçando um comparativo entre os resultados obtidos. Também foi utilizada uma árvore de decisão C4.5 como referência de desempenho, uma vez que esta é um classificador multiclasses.

6.2.5 Análise dos resultados

A tabela de palavras código gerada pelo AG permite que mais de uma palavra código seja associada a cada classe. Assim, é possível ter palavras código que representem um pequeno número de exemplos de treinamento, denominados pequenos disjuntos. Segundo Danyluk e Provost [150], em uma aplicação do mundo real, a exemplo na detecção de intrusão, o conjunto dos pequenos disjuntos pode vir a cobrir em torno de 50% dos exemplos de treinamento. Portanto, se o algoritmo de classificação ignora os pequenos disjuntos, a precisão preditiva do processo de classificação pode ser reduzida de forma significativa.

A seleção de atributos usando as medidas de Rényi e Tsallis diminuiu significativamente o número de atributos e a dimensionalidade de dados, conduzindo a um melhor desempenho no modelo de classificação proposto e em outros classificadores testados (ver resultados adicionais no Apêndice J), o que resulta em menor tempo de processamento, em comparação com a situação em que todos os atributos da base de dados original foram utilizados.

Como a base de dados utilizada nestes experimentos foi reduzida ao máximo, as médias dos resultados encontrados por todos os algoritmos testados tiveram um desempenho inferior a 99% (ver Tabelas 6.8 e 6.9). Observou-se, também, um elevando desvio padrão nos resultados encontrados. Contudo, o algoritmo que apresentou maior desvio padrão foi o algoritmo de árvore de decisão C4.5, uma vez que uma das vantagens de se utilizar códigos corretores de erros, é a sua habilidade para tratar os pequenos disjuntos.

Tabela 6.8 Resultado obtido no experimento realizado usando atributos selecionados por medidas de Rényi.

Classificador	DoS	Probing	R2L	U2R
C4.5	98,42%	93,54%	80,00%	66,66%
BCH_BMA	85,71%	90,32%	80,00%	66,66%
RS_BMA	76,19%	83,87%	73,33%	100%
RS_GS	77,77%	84,84%	73,33%	66,66%

Tabela 6.9 Resultado obtido no experimento realizado usando atributos selecionados por medidas de Tsallis.

Classificador	DoS	Probing	R2L	U2R
C4.5	50,00%	73,33%	90,90%	98,65%
BCH_BMA	98,87%	73,33%	93,94%	84,61%
RS_BMA	50,00%	73,33%	87,87%	73,07%
RS_GS	50%	73,33%	87,87%	50,00%

Nos experimentos apresentados nas Tabelas 6.8 e 6.9, houve variações no desempenho observado na identificação de intrusões. Por exemplo, observou-se que a versão de AG para a determinação de tabelas de palavras código foi capaz de gerar soluções com desempenho na classificação correta (TVP) e precisão de até 100% para algumas categorias de ataques.

De maneira geral, cada estratégia usando códigos corretores de erros tende a enfatizar algumas categorias de ataques. Essa característica pode ser explorada na escolha de uma estratégia particular em domínios em que alguns tipos ou categorias de intrusão sejam considerados mais importantes do que outros. Esses resultados também demonstram um potencial para a combinação de diferentes estratégias em uma solução global. Dessa forma, é possível explorar as complementariedades encontradas na identificação de categorias de ataques em uma rede de computadores.

Em uma análise simplificada, verificou-se que as estratégias que mais se sobressaíram foram as que utilizam códigos BCH binários, que juntamente com a utilização dos métodos tradicionais de decodificação, permitiram identificar, com maior precisão, os tipos de ataques e, portanto, estes métodos podem ser considerados como uma solução viável a ser utilizada em problemas multiclasses.

Foi verificado empiricamente que o uso de decodificação por lista pode ser aplicado como um classificador binário, pois o mesmo tem a capacidade de listar como resposta os possíveis tipos de ataques que estejam ocorrendo na rede, cabendo ao sistema decidir qual o tipo de ataque. Além disso, o processo de decodificação por lista é computacionalmente demorado e, portanto, a decodificação por lista deve ser utilizada, preferencialmente, nos casos em que há falhas de decodificação pelos decodificadores tradicionais.

Com base nos resultados experimentais obtidos, os códigos BCH e Reed-Solomon possuem as seguintes características:

- Trabalham bem em grandes conjuntos de dados;
- Não possuem habilidade para trabalhar com os diferentes tipos de atributos;
- Necessidade de conhecimento do domínio para determinar a estrutura do código;
- Habilidade para lidar com dados ruidosos, devido à sua capacidade de correção de erros;
- Habilidade para trabalhar com pequenos disjuntos;
- Habilidade para trabalhar com novas entradas de dados que não foi testada, ou seja, possui capacidade de generalização;
- Indiferença na ordem de apresentação dos objetos;
- Permite o uso de decodificadores algébricos.

Embora a abordagem aqui apresentada reduza o espaço das possíveis soluções, o novo espaço de busca é mais estruturado e a distância mínima do código corretor de erros escolhido é inicialmente conhecida, não sendo necessário avaliar este parâmetro durante o processo de busca da tabela de palavras código. Além disso, outro benefício dessa abordagem, é que o

esquema adotado no AG aqui proposto reduz o custo computacional envolvido na seleção das tabelas de palavras código.

Diversos experimentos foram realizados no decorrer desta tese. Contudo, optou-se por apresentar apenas aqueles considerados mais relevantes para a demonstração das estratégias propostas. Outros resultados podem ser consultados no Apêndice J e em artigos publicados no período [16, 17, 103, 144–146, 151] e que encontram-se nos anexos desta tese.

6.3 Considerações Finais

Neste capítulo analisou-se experimentalmente as estratégias propostas nesta tese. Inicialmente investigou-se o uso de medidas de informação na determinação dos atributos que serão utilizados como pontos de observação dos detectores. A seleção de atributos trouxe benefícios para solução encontrada, uma vez que define o tamanho das palavras código e reduz o custo computacional envolvido no processo de seleção das tabelas. A saída destes detectores irão compor os símbolos da palavra recebida.

Utilizou-se uma variante de algoritmo genético para selecionar a tabela de palavras código com melhor desempenho na identificação de tipos de intrusão a uma rede de computadores. Investigou-se ainda o uso de decodificadores algébricos baseados no conceito de decodificação tradicional e baseado no conceito de decodificação por lista.

Entre os benefícios verificados, tem-se uma estrutura que permite o uso de decodificadores algébricos. Os resultados experimentais evidenciaram um melhor desempenho com o uso de códigos BCH, que utiliza decodificação tradicional. Contudo, a decodificação por lista, apesar de se ter um maior custo computacional envolvido nesse processo, pode ser uma alternativa em ambientes que possuem várias camadas de proteção.

No próximo capítulo são apresentadas as conclusões deste trabalho.

CAPÍTULO 7

Conclusões e Perspectivas

Neste capítulo são sintetizados os principais resultados obtidos e as sugestões para investigações futuras.

7.1 Principais conclusões

Cumprindo com o objetivo posto para este trabalho de tese, foi apresentado um esquema para classificação multiclasses, baseado em tabela de palavras código, que consiste basicamente em relacionar palavras código de um código BCH ou de um código Reed-Solomon a cada classe do problema analisado. Os atributos de rede, que serão monitorados pelos detectores de rede e de *host*, são determinados por meio do uso de medidas de informação de Rényi [40] e Tsallis [41,42]. Tais detectores serão responsáveis por produzir exatamente um bit dos símbolos da palavra recebida, com base em sua observação.

Conforme apresentado no Capítulo 2, os IDS devem proteger um sistema computacional contra ataques conhecidos e é desejável que o mesmo seja capaz de identificar atividades maliciosas desconhecidas. Também foi exposto que a grande maioria dos ataques a redes de computadores possui padrões de assinaturas, podendo existir assinaturas variadas, porém próximas (dada uma medida de distância) de atividades que exploram as mesmas vulnerabilidades. Nesse contexto, é necessário que sejam definidos padrões de assinatura que possam ser utilizados na identificação de ataques, classificando-os individualmente pelo seu tipo ou categoria e permitir que variações de um padrão também possam ser identificadas sem a necessidade imediata de atualização do sistema de detecção de intrusão.

Com base no que foi exposto e nos resultados dos experimentos realizados, o uso de códigos pode ser uma alternativa, visto que é possível relacionar uma ou mais palavras código a um padrão de assinatura de ataque. Dentro do limite $t \leq \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$ de correção de erros do código utilizado para esta finalidade, uma dada palavra recebida pode ser decodificada para uma palavra código associada a este padrão e assim, um ataque pode ser identificado e classificado.

Os resultados experimentais, utilizando-se uma base de dados artificial de tráfego de rede contendo variados tipos de intrusões, demonstram que o uso de árvores de decisão baseadas nas entropias de Rényi [40] e Tsallis [41, 42], podem ser utilizadas como métodos de seleção dos atributos a serem utilizados para compor as palavras recebidas pelo decodificador.

Confirmou-se nos experimentos que o algoritmo C4.5, baseado em medidas de informação de Rényi [40] e Tsallis [41, 42] podem auxiliar na escolha dos atributos a serem monitorados pelos detectores.

O conceito de decodificação por lista [44, 45] foi utilizado neste trabalho, a princípio, para minimizar as falhas de decodificação, pois algumas palavras recebidas podem ficar fora da região de decodificação. Este esquema obteve resultados compatíveis com outros métodos utilizados nos experimentos e podem ser utilizados como um classificador binário, cujo objetivo é separar o tráfego normal do tráfego suspeito.

No Apêndice A é apresentada a lista de artigos produzidos e uma cópia desses encontram-se nos anexos desta tese.

As contribuições deste trabalho de tese são:

- A implementação das medidas de informação de Rényi e Tsallis na ferramenta Weka, aplicando-as na detecção de intrusos;
- A utilização das medidas de informação de Rényi e Tsallis na seleção de atributos;
- O uso de códigos BCH e de códigos Reed-Solomon aplicados à solução de problemas multiclasses;
- A utilização de algoritmos genéticos na escolha das palavras código para compor a tabela de palavras código;
- A utilização do algoritmo de decodificação para códigos BCH e Reed-solomon, usando o algoritmo de *Berlekamp-Massey* [43] aplicados no sistema de classificação multiclasses;
- A utilização do algoritmo de decodificação por lista de Guruswami-Sudan para códigos Reed-Solomon aplicados no sistema de classificação multiclasses.

7.2 Perspectivas Futuras

O trabalho realizado dá abertura para novas investigações. Como perspectivas para futuros trabalhos advindos da pesquisa realizada, propõe-se:

- Investigar o uso do esquema proposto no tráfego de rede real;
- Analisar o uso de medidas de informação de Rényi e Tsallis em outras estratégias de seleção de atributos;

- Analisar o uso de medidas de informação de Rényi e Tsallis como método de discretização de atributos;
- Pesquisar novos tipos de códigos corretores de erros que possam ser aplicados no AG para selecionar tabelas de palavras código com melhor desempenho;
- Aplicar um algoritmo de decodificação por lista para códigos BCH binários.

Referências Bibliográficas

[1] VISA. *Estudo da Visa detalha impacto do uso de cartões na economia brasileira*. São Paulo: [s.n.], ago. 2012. Disponível em: <http://www.visa.com.br/downloads/>. Acesso em: 10 nov. 2012.

[2] VISA. *Visa registra recorde de transações pelo 18o ano consecutivo durante as compras de final de ano*. São Paulo: [s.n.], fev. 2011. Disponível em: <http://www.visa.com.br/downloads/>. Acesso em: 10 nov. 2012.

[3] CERT, Centro de Estudos, Resposta e Tratamento de Incidentes de Segurança no Brasil. Disponível em: <http://www.cert.br/stats/incidentes/>. Acesso em: 01 nov. 2012.

[4] GARFINKEL, S.; SPAFFORD, G.; SCHWARTZ, A. *Practical Unix & Internet Security, 3rd Edition*. Sebastopol: O'Reilly Media, 2003. 988 p.

[5] NAKAMURA, E. T.; GEUS, P. L. de. *Segurança de Redes em Ambientes Cooperativos*. São Paulo: Novatec, 2007. 488 p.

[6] ALLEN, J. et al. *State of the Practice of Intrusion Detection Technologies*. Pittsburgh, jan. 2000.

[7] BACE, R.; MELL, P. *Intrusion Detection Systems*. 31. ed. [S.l.]: U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2001. 43 p. (NIST special publication, v. 800).

[8] KRUEGEL, C.; VALEUR, F.; VIGNA, G. *Intrusion Detection and Correlation : Challenges and Solutions (Advances in Information Security)*. [S.l.]: Springer, 2004. Hardcover.

[9] GHORBANI, A. A.; LU, W.; TAVALLAEE, M. *Network Intrusion Detection and Prevention: Concepts and Techniques*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2009.

[10] STALLINGS, W. *Network Security Essentials: Applications and Standards*. 4th. ed. Upper Saddle River: Prentice Hall Press, 2010.

[11] KULKARNI, S. R.; LUGOSI, G.; VENTKATESH, S. S. Learning pattern classification – a survey. *IEEE Trans. Inform. Theory*, v. 44, n. 6, p. 2178–2206, out. 1998.

[12] HAN, J.; KAMBER, M. *Data Mining: Concepts and Techniques*. 2. ed. San Francisco: Morgan Kaufmann, 2006.

[13] LEE, W.; STOLFO, S.; CHAN, P. Learning patterns from unix process execution traces for intrusion detection. In: *AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management*. [S.l.: s.n.], 1997. p. 50–56.

[14] PEDDABACHIGARI, S. et al. Modeling intrusion detection system using hybrid intelligent systems. *J. Netw. Comput. Appl.*, Academic Press, London, v. 30, n. 1, p. 114–132, 2007.

[15] AMOR, N. B.; BENFERHAT, S.; ELOUEDI, Z. Naive bayes vs decision trees in intrusion detection systems. In: *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*. New York: ACM, 2004. p. 420–424.

[16] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. Decision tree based on shannon, renyi and tsallis entropies for intrusion tolerant systems. *Fifth International Conference on Internet Monitoring and Protection*, IEEE Computer Society, Barcelona, v. 0, p. 117–122, maio 2010.

[17] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. Artificial immune systems applied in intrusion tolerant systems. *Anais do Congresso Wireless Systems International Meeting - RFID: trends to the future*, Campina Grande, maio 2010.

[18] PIMENTA, E. M. C. *Abordagens para decomposição de problemas multiclasse: os códigos de correção de erros de saída*. Dissertação (Mestrado) — Faculdade de Ciência da Universidade do Porto, dez. 2004.

[19] LORENA, A. C.; CARVALHO, A. C. P. L. F. de. Evolutionary design of code-matrices for multiclass problems. *Soft Computing for Knowledge Discovery and Data Mining*, p. 153–184, 2008.

[20] VAPNIK, V. N. *The nature of statistical learning theory*. New York: Springer-Verlag, 1995.

[21] KNERR, S.; PERSONNAZ, L.; DREYFUS, G. Single-layer learning revisited: A stepwise procedure for building and training a neural network. In: SOULIÉ, F. F.; HÉRAULT, J. (Ed.). *Neurocomputing: Algorithms, Architectures and Applications*. Berlin: Springer-Verlag, 1990, (NATO ASI Series, F68). p. 41–50.

[22] HASTIE, T.; TIBSHIRANI, R. Classification by pairwise coupling. *The Annals of Statistics*, v. 26(2), p. 451–471, 1998.

[23] DIETTERICH, T. G.; BAKIRI, G. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, v. 2, p. 263–286, 1995.

[24] BLAHUT, R. E. *Theory and Practice of Error Control Codes*. Owego: Addison-Wesley Publishing Company, Inc., 1983.

[25] ESCALERA, S. et al. Subclass problem-dependent design for error-correcting output codes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, v. 30, n. 6, p. 1041–1054, 2008.

[26] GARCÍA-PEDRAJAS, N.; GARCÍA, A. de H. Output coding methods: Review and experimental comparison. In: YIN, P.-Y. (Ed.). *Pattern Recognition Techniques, Technology and Applications*. Croatia: InTech, 2008, (Artificial Intelligence). cap. 14, p. 327–344.

[27] ROCHA, A.; GOLDENSTEIN, S. Multi-class from binary - divide to conquer. In: RANCHORDAS, A.; ARAúJO, H. (Ed.). *Proceedings of the Fourth International Conference on Computer Vision Theory and Applications (VISAPP 2009)*. Lisboa: INSTICC Press, 2009. v. 1, p. 323–330.

[28] SANTOS, E. A.; ASSIS, F. M. de; GURJãO, E. C. Redes de sensores com codificação bch distribuída. *XXVII Simpósio Brasileiro de Telecomunicações - SBrT 2009*, outubro 2009.

[29] MOREIRA, L. M. *The use of boolean concepts in general classification contexts*. Tese (Doutorado) — Escola Politécnica Federal de Lausanne, Suiça, 2000.

[30] ALLWEIN, E. L.; SCHAPIRE, R. E.; SINGER, Y. Reducing multiclass to binary: a unifying approach for margin classifiers. *J. Mach. Learn. Res.*, JMLR, v. 1, p. 113–141, set. 2001.

[31] GARCIA-PEDRAJAS, N.; FYFE, C. Evolving output codes for multiclass problems. *IEEE Transactions on Evolutionary Computation*, v. 12, n. 1, p. 93–106, fev. 2008.

[32] XIE, X. et al. Network traffic classification based on error-correcting output codes and nn ensemble. *2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, Ieee, p. 475–479, 2009.

[33] KOUZANI, A.; NASIREDING, G. Multilabel classification by bch code and random forests. *International Journal of Recent Trends in Engineering (IJRTE)*, v. 2, n. 1, p. 113–116, nov. 2009.

[34] KOUZANI, A. Z. Multilabel classification using error correction codes. In: CAI, Z. et al. (Ed.). *Advances in Computation and Intelligence*. [S.l.]: Springer Berlin / Heidelberg, 2010. (Lecture Notes in Computer Science, v. 6382), p. 444–454.

[35] ALPAYDIN, E.; MAYORAZ, E. Learning error-correcting output codes from data. In: *Proceedings of the 9th International Conference on Neural Networks*. [S.l.: s.n.], 1999. p. 743–748.

[36] POLYCHRONAKIS, M.; ANAGNOSTAKIS, K. G.; MARKATOS, E. P. Real–world polymorphic attack detection. *In Proceedings of the 4th International Annual Workshop on Digital Forensics & Incident Analysis (WDFIA)*, jun. 2009.

[37] WICKER, S. B. *Error Control Systems for Digital Communication and Storage*. Upper Saddle River: Prentice-Hall, 1995. 624 p.

[38] LORENA, A. C. *Investigação de estratégias para a geração de máquinas de vetores de suporte multiclasses*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação - ICMC-USP, 2006.

[39] TANENBAUM, A. S. *Redes de Computadores*. trad. 4 ed. Rio de Janeiro: Elsevier, 2003.

[40] RÉNYI, A. On measures of entropy and information. In: *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability*. Berkeley: University of California Press, 1960. v. 1, p. 547–561.

[41] HAVRDA, M.; CHARVáT, F. Quantification method of classification processes: concept of structural alfa-entropy. *Kybernetika*, v. 3, p. 30–35, 1967.

[42] TSALLIS, C. Possible generalization of boltzmann-gibbs statistics. *Journal of Statistical Physics*, v. 52, n. 1-2, p. 479–487, 1988.

[43] MASSEY, J. L. Shift-register synthesis and bch decoding. *IEEE Transactions on Information Theory*, v. 15, n. 1, p. 122–127, jan. 1969.

[44] ELIAS, P. List decoding for noisy channels. Research Laboratory of Electronics, Massachusetts Institute of Technology, set. 1957.

[45] WOZENCRAFT, J. M. *List decoding*. Cambridge, 1958. 90–95 p. (Quarterly Progress Report, v. 48).

[46] LIMA, C. F. L. *Agentes Inteligentes para Detecção de Intrusos em Redes de Computadores*. Dissertação (Mestrado) — PPGEE/UFMA, maio 2002.

[47] ANDERSON, J. P. *Computer Security Threat Monitoring and Surveillance*. Fort Washington, abr. 1980.

[48] DENNING, D. E. An intrusion detection model. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, v. 13, n. 2, p. 222–232, fev. 1987.

[49] PORRAS, P. et al. The common intrusion detection framework. *Information Survivability Workshop*, Orlando, out. 1998. Disponível em: <http://gost.isi.edu/cidf/drafts/architecture.txt>. Acesso em: 23 ago. 2012.

[50] PTACEK, T. H.; NEWSHAM, T. N. *Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection*. Calgary, jan. 1998.

[51] KUMAR, S. *Classification and Detection of Computer Intrusions*. Tese (Doutorado) — Purdue University, 1995.

[52] JR., F. G. et al. *A real-time intrusion-detection expert system (IDES)*. Washington: SRI International, Computer Science Laboratory, 1992. 157 p.

[53] TENG, H. S.; CHEN, K.; LU, S. C.-Y. Security audit trail analysis using inductively generated predictive rules. In: *Proceedings of the sixth conference on Artificial intelligence applications*. Piscataway: IEEE Press, 1990. p. 24–29.

[54] FOX, K. L. et al. A neural network approach towards intrusion detection. In: *Proceedings of the 13th National Computer Security Conference*. [S.l.: s.n.], 1990. p. 125–134.

[55] AHMAD, I.; ABDULLAH, A. B.; ALGHAMDI, A. S. Artificial neural network approaches to intrusion detection: a review. In: *Proceedings of the 8th Wseas international conference on Telecommunications and informatics*. Stevens Point: World Scientific and Engineering Academy and Society (WSEAS), 2009. p. 200–205.

[56] BEQIRI, E.; LEE, S. W.; DRAGANOVA, C. A neural network approach for intrusion detection systems. *5th Conference in Advances in Computing and Technology*, University of East London, p. 209–217, jan. 2010.

[57] RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 2nd edition. ed. [S.l.]: Prentice-Hall, Englewood Cliffs, NJ, 2003.

[58] SNAPP, S. R.; SMAHA, S. E. Signature analysis model definition and formalism. In: *Proceedings of the Fourth Workshop on Computer Security Incident Handling*. Denver: [s.n.], 1992.

[59] LUNT, T. F. A survey of intrusion detection techniques. *Computers & Security*, v. 12, n. 4, p. 405–418, jun. 1993.

[60] GARVEY, T. D.; LUNT, T. F. Model-based intrusion detection. In: *Proceedings of the 14th National Computer Security Conference*. Washington: National Institute of Standards and Technology, National Computer Security Center, 1991. v. 1, p. 372–385.

[61] PORRAS, P. A.; KEMMERER, R. A. Penetration state transition analysis: a rule-based intrusion detection approach. In: *Eighth Annual Computer Security Applications Conference*. San Antonio: [s.n.], 1992. p. 220–229.

[62] CANSIAN, A. M. et al. Network intrusion detection using neural networks. In: *Proceedings of International Conference on Computational Inteligence and Multimedia Applications*. Gold Coast: [s.n.], 1997. p. 276–280.

[63] NOROUZIAN, M. R.; MERATI, S. Classifying attacks in a network intrusion detection system based on artificial neural networks. *Neural Networks*, ACM Press, p. 868–873, 2011.

[64] WANG, J. *Computer Network Security: Theory and Practice*. 1. ed. [S.l.]: Berlin: Springer; Beijing: Higher Education Press, 2009. 384 p.

[65] CHOU, T.-S. Cyber security threats detection using ensemble architecture. *International Journal of Security and Its Applications*, v. 5, n. 2, p. 17–31, abr. 2011.

[66] BALASUBRAMANIYAN, J. et al. An architecture for intrusion detection using autonomous agents. In: *Proceedings of the 14th Annual Computer Security App. Conference*. Los Alamitos: IEEE Computer Society, 1998. p. 13–24.

[67] BARRUS, J.; ROWE, N. C. A distributed autonomous-agent network-intrusion detection and response system. In: *Proceedings of the 1998 Command and Control Research and Technology Symposium*. Monterey, CA: [s.n.], 1998. p. 577–586.

[68] FORREST, S. et al. Self-nonself discrimination in a computer. In: *SP '94: Proceedings of the 1994 IEEE Symposium on Security and Privacy*. Washington: IEEE Computer Society, 1994. p. 202.

[69] KEPHART, J. O. A biologically inspired immune system for computers. In: *In Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. [S.l.]: MIT Press, 1994. p. 130–139.

[70] FORREST, S.; HOFMEY, S. A.; SOMAYAJI, A. Computer immunology. *Commun. ACM*, ACM, New York, v. 40, p. 88–96, out. 1997.

[71] GREENSMITH, J.; AICKELIN, U.; TWYCROSS, J. Detecting danger: Applying a novel immunological concept to intrusion detection systems. *CoRR*, abs/1002.0696, 2010.

[72] SILVA, L. N. de C. *Engenharia imunológica: desenvolvimento e aplicação de ferramentas computacionais inspiradas em sistemas imunológicos artificiais.* Tese (Doutorado) — Universidade Estadual de Campinas, São Paulo, 2001.

[73] DASGUPTA, D. Immunity-based intrusion detection systems: a general framework. In: *Twenty Second National Information Systems Security Conference*. [S.l.: s.n.], 1999. p. 147–160.

[74] DASGUPTA, D.; BRIAN, H. Mobile security agents for network traffic analysis. *DARPA Information Survivability Conference and Exposition,*, IEEE Computer Society, Los Alamitos, v. 2, p. 1332, 2001.

[75] HOFMEYR, S. A.; FORREST, S. A. Architecture for an artificial immune system. *Evol. Comput.*, MIT Press, Cambridge, v. 8, p. 443–473, dez. 2000.

[76] PAULA, F. S. de. *Uma arquitetura de segurança computacional inspirada no sistema imunológico*. Tese (Doutorado) — Universidade Estadual de Campinas, SP, 2004.

[77] CASTRO, L. N. D.; ZUBEN, F. J. V. The clonal selection algorithm with engineering applications. In: *In GECCO 2002 - Workshop Proceedings*. [S.l.]: Morgan Kaufmann, 2002. p. 36–37.

[78] BROWNLEE, J. *CLONAL SELECTION THEORY & CLONALG THE CLONAL SELECTION CLASSIFICATION ALGORITHM (CSCA)*. [S.l.], 2005.

[79] WATKINS, A.; TIMMIS, J.; BOGGESS, L. Artificial immune recognition system (airs): An immune-inspired supervised learning algorithm. *Genetic Programming and Evolvable Machines*, Kluwer Academic Publishers, Hingham, v. 5, n. 3, p. 291–317, 2004.

[80] BROWNLEE, J. *Immunos-81, The Misunderstood Artificial Immune System*. [S.l.], jan. 2005.

[81] JAIN, A. K.; DUBES, R. C. *Algorithms for clustering data*. Upper Saddle River: Prentice-Hall, Inc., 1988.

[82] KAUFMAN, L.; ROUSSEEUW, P. *Finding Groups in Data An Introduction to Cluster Analysis*. New York: Wiley Interscience, 1990.

[83] QUINLAN, J. R. *C4.5 Programs for Machine Learning*. San Diego: Morgan Kaufmann, 1993.

[84] SHANNON, C. E. A mathematical theory of communication. *Bell Systems Technical Journal*, v. 27, p. 379–423 and 623–656, 1948.

[85] MITCHELL, T. M. *Machine Learning*. 1. ed. New York: McGraw-Hill, 1997. 414 p.

[86] BREIMAN, L. et al. *Classification and Regression Trees*. New York: Chapman & Hall, 1984. 358 p.

[87] ROKACH, L.; MAIMON, O. *Data Mining with Decision Trees: Theory and Applications*. Singapore: World Scientific, 2008. 244 p. (Series in Machine Perception and Artificial Intelligence, v. 69).

[88] QUINLAN, J. R. Induction of decision trees. *Machine Learning*, Kluwer Academic Publishers, Hingham, v. 1, n. 1, p. 81–106, 1986.

[89] BRAMER, M. *Principles of data mining*. London: Springer, 2007. 354 p. (Undergraduate Topics in Computer Science).

[90] MURTHY, S. Automatic construction of decision trees from data: a multi-disciplinary survey. In: *Data Mining and Knowledge Discovery*. [S.l.]: Springer, 1998. v. 2, n. 4, p. 345–389.

[91] BEN-BASSAT, M. Use of distance measures, information measures and error bounds in feature evaluation. In: KRISHNAIAH, P.; KANAL, L. (Ed.). *Classification Pattern Recognition and Reduction of Dimensionality*. [S.l.]: Elsevier, 1982, (Handbook of Statistics, v. 2). p. 773 – 791.

[92] WANG, Q. R.; SUEN, C. Y. Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition. In: *IEEE Transaction on Pattern Analysis and Machine Intelligence*. [S.l.: s.n.], 1984. v. 6, p. 406–417.

[93] WU, X. *Knowledge Acquisition from Databases*. Norwood: Ablex Publishing Corporation, 1995.

[94] QUINLAN, J. R. Improved use of continuous attributes in c4.5. *Artificial Intelligence Research*, AI Access Foundation, USA, v. 4, n. 1, p. 77–90, mar. 1996.

[95] RISSANEN, J. A universal prior for integers and estimation by minimum description length. *The Annals of Statistics*, v. 11, n. 2, p. 416–431, 1983.

[96] QUINLAN, J. R.; RIVEST, R. L. Inferring decision trees using the minimum description length principle. *Information and Computation*, Academic Press, Duluth, MN, USA, v. 80, n. 3, p. 227–248, mar. 1989.

[97] QUINLAN, J. R. Simplifying decision trees. *International Journal of Man-Machine Studies*, Academic Press Ltda, London, v. 27, n. 3, p. 221–234, 1987.

[98] TSALLIS, C. "Nonextensive Statistics: Theoretical, Experimental and Computational Evidences and Connections". *Brazilian Journal of Physics*, v. 29, p. 1–35, March 1999.

[99] BASSAT, M.; RAVIV, J. Renyi's entropy and the probability of error. *IEEE Transactions on Information Theory*, v. 24, n. 3, p. 324–330, 1978.

[100] MELLO, C. A. B.; SCHULER, L. A. Tsallis entropy-based thresholding algorithm for images of historical documents. In: *SMC*. [S.l.]: IEEE, 2007. p. 1112–1117.

[101] WITTEN, I.; FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. 2. ed. San Francisco: Morgan Kaufmann, 2005.

[102] HUNT, E. B.; MARIN, J.; STONE, P. J. Experiments in induction. *New York: Academic Press*, 1966.

[103] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. Árvores de decisão baseadas nas entropias de shannon, rényi e tsallis para sistemas tolerantes a intrusão. *La Novena Conferencia Iberoamericana en Sistemas, Cibernética e Informática*, Orlando, p. 142–147, jun. 2010.

[104] MOREIRA, M.; MAYORAZ, E. Improved pairwise coupling classification with correcting classifiers. In: *Proceedings of the 10th European Conference on Machine Learning*. London: Springer-Verlag, 1998. p. 160–171.

[105] FURNKRANZ, J. Round robin classification. *Journal of Machine Learning Research*, v. 2, p. 721–747, 2002.

[106] KRESSEL, U. H. G. Pairwise classification and support vector machines. In: SCHöLKOPF, B.; BURGES, C. J. C.; SMOLA, A. J. (Ed.). *Advances in kernel methods*. Cambridge: MIT Press, 1999. cap. 15, p. 255–268.

[107] PLATT, J. C.; CRISTIANI, N.; SHAWE-TAYLOR, J. Large margin dags for multiclass classification. In: *Advances in Neural Information Processing Systems*. [S.l.]: The MIT Press, 2000. v. 12, p. 547–553.

[108] ESCALERA, S.; PUJOL, O.; RADEVA, P. On the decoding process in ternary error-correcting output codes. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, Washington, v. 32, p. 120–134, jan. 2010.

[109] ESCALERA, S.; PUJOL, O.; RADEVA, P. Error-correcting ouput codes library. *J. Mach. Learn. Res.*, JMLR, v. 11, p. 661–664, mar. 2010.

[110] ESCALERA, S.; PUJOL, O.; RADEVA, P. Optimal extension of error correcting output codes. In: *Proceeding of the 2006 conference on Artificial Intelligence Research and Development*. Amsterdam: IOS Press, 2006. p. 28–36.

[111] SANTOS, E. A. *Redes de Sensores com Codificação BCH Distribuída*. Dissertação (Mestrado) — Universidade Federal de Campina Grande, abr. 2009.

[112] AKYILDIZ, I. F. et al. A survey on sensor networks. *IEEE Communications Magazine*, v. 40, n. 8, p. 102–114, ago. 2002.

[113] YAO, C. et al. Performance analysis and code design for minimum hamming distance fusion in wireless sensor networks. *IEEE Transaction on Information Theory*, v. 53, n. 5, p. 1716–1734, maio 2007.

[114] WANG, T. yi et al. Distributed fault-tolerant classification in wireless sensor networks. *IEEE Journal on Selected Areas in Communications*, v. 23, p. 724–734, 2005.

[115] ASSIS, F. M. de. Genetic algorithms and packing of block codes. *International Conference on Tele-comunications, Proceedings of the ICT97*, v. 3, p. 1045–1048, 1997.

[116] ASSIS, F. M. de. Weight structure of binary codes and the performance of blind search algorithms. *Neural Networks, Brazilian Symposium on, IEEE Computer Society*, v. 0, p. 144, 2000.

[117] RAISINGHANI, V. T.; JAGANNATH, P. Evaluating multi-class classification using binary svms (it-642: Course project). 2002.

[118] KDD cup 99 Intrusion detection data set. Disponível em: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>. Acesso em: 01 mar. 2012.

[119] GORENSTEIN, D.; ZIERLER, N. A class of error-correcting codes in p^m symbols. *Journal of the Society of Industrial and Applied Mathematics (JSIAM)*, v. 9, p. 207–214, 1961.

[120] REED, I. S.; SOLOMON, G. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, SIAM, v. 8, n. 2, p. 300–304, 1960.

[121] GURUSWAMI, V.; RUDRA, A. Limits to list decoding reed-solomon codes. In: *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. New York: ACM, 2005. (STOC '05), p. 602–609.

[122] ESTéVEZ, P. A. et al. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, v. 20, n. 2, p. 189–201, fev 2009.

[123] LIU, H.; YU, L. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, IEEE Computer Society, Los Alamitos, USA, v. 17, p. 491–502, 2005.

[124] BLUM, A. L.; LANGLEY, P. Selection of relevant features and examples in machine learning. *ARTIFICIAL INTELLIGENCE*, v. 97, p. 245–271, 1997.

[125] GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. *Journal of Machine Learning Research*, v. 3, p. 1157–1182, 2003.

[126] M., D. Feature selection for dimensionality reduction. In: SAUNDERS, C. et al. (Ed.). *Subspace, Latent Structure and Feature Selection*. [S.l.]: Springer Berlin Heidelberg, 2006, (Lecture Notes in Computer Science, v. 3940). p. 84–102.

[127] YANG, Y.; PEDERSEN, J. O. A comparative study of feature selection in text categorization. In: *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. p. 412–420.

[128] HALL, M. A. *Correlation-based Feature Subset Selection for Machine Learning*. Tese (Doutorado) — University of Waikato, Hamilton, New Zealand, 1998.

[129] KOHAVI, R.; JOHN, G. H. The wrapper approach. In: LIU, H.; MOTODA, H. (Ed.). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Boston: Kluwer Academic Publishers, 1998, (The Kluwer International Series in Engineering and Computer Science, v. 453). cap. 3, p. 33–50.

[130] HOCQUENGHEM, A. Codes correcteurs d'erreurs. *Chiffres*, v. 2, p. 147–158, 1959.

[131] BOSE, R. C.; RAY-CHAUDHURI, D. K. On a class of error correcting binary group codes. *Inform. and Control*, v. 3, p. 68–79, 1960.

[132] TANOMARU, J. Motivação, fundamentos e aplicações de algoritmos genéticos. In: *Congresso Brasileiro de Redes Neurais*. Curitiba: CNRN/Copel, 1995. (Anais, 2), p. 373–403.

[133] MORELOS-ZARAGOZA, R. H. *The Art of Error Correcting Coding*. [S.l.]: John Wiley & Sons, 2006.

[134] PETERSON, W. W. Encoding and error-correction procedures for the bose-chaudhuri codes. *IRE Transactions on Information Theory*, IT-6, p. 459–470, 1960.

[135] BERLEKAMP, E. *Algebraic Coding Theory*. New York: McGraw-Hill, 1968.

[136] CHIEN, R. T. Cyclic decoding procedures for bose-chaudhurihocquenghem codes. *IEEE Transactions on Information Theory*, v. 10, n. 4, p. 357–363, 1964.

[137] FORNEY, G. D. *Concatenated codes*. [S.l.]: MIT Press, 1966.

[138] SHAFAREVICH, I. R. *Basic Algebraic Geometry*. Berlin: Springer, 1995. 328 p.

[139] WELCH, L.; BERLEKAMP, E. *Error correction for algebraic block codes*. dez. 1986. US Patent 4 633 470.

[140] SUDAN, M. Decoding of reed solomon codes beyond the error-correction bound. *J. Complex.*, Academic Press, Orlando, v. 13, p. 180–193, mar. 1997.

[141] GURUSWAMI, V.; SUDAN, M. Improved decoding of reed-solomon and algebraic geometry codes. *IEEE Trans. Information Theory*, v. 45, n. 6, p. 1755–1764, set 1999.

[142] MOON, T. K. *Error Correction Coding: Mathematical Methods and Algorithms*. [S.l.]: Wiley, 2005.

[143] COUTINHO, F. R. *Decodificação por decisão suave de códigos de Reed-Solomon na Presença de símbolos apagados*. Dissertação (Mestrado) — Universidade Federal do Paraná, 2007.

[144] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. A comparative study of use of shannon, rényi and tsallis entropy for attribute selecting in network intrusion detection. *IEEE International Workshop on Measurements and Networking*, p. 77–82, 2011.

[145] LIMA, C.; ASSIS, F. M.; SOUZA, C. P. de. A comparative study of use of shannon, rényi and tsallis entropy for attribute selecting in network intrusion detection. In: YIN, H.; COSTA, J. A. F.; BARRETO, G. (Ed.). *Intelligent Data Engineering and Automated Learning - IDEAL 2012*. [S.l.]: Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7435). p. 492–501.

[146] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. de. An empirical investigation of attribute selection techniques based on shannon, rényi and tsallis entropies for network intrusion detection. *American Journal of Intelligent Systems*, v. 2(5), p. 111–117, 2012.

[147] DOUGHERTY, J.; KOHAVI, R.; SAHAMI, M. Supervised and unsupervised discretization of continuous features. In: *Proceedings of the Twelfth International Conference*. San Francisco, USA: Morgan Kaufmann, 1995. p. 194–202.

[148] FAYYAD, U. M.; IRANI, K. B. Multi-interval discretization of continuous-valued attributes for classification learning. In: *Thirteenth International Joint Conference on Articial Intelligence*. Chambéry, France: Morgan Kaufmann, 1993. v. 2, p. 1022–1027.

[149] RISSANEN, J. Stochastic complexity and modeling. *The Annals of Statistics*, v. 14, n. 3, p. 1080–1100, 1986.

[150] DANYLUK, A. P.; PROVOST, F. Small disjuncts in action: Learning to diagnose errors in the local loop of the telephone network. In: *Proc. 10th International Conf. Machine Learning*. [S.l.]: Morgan Kaufmann, 1993. p. 81–88.

[151] LIMA, C. F. L.; ASSIS, F. M. de; SOUSA, C. P. de. Detecção de intrusos em redes de computadores com usos de códigos corretores de erros e medidas de informação. *Anais do X Congresso Brasileiro de Inteligência Computacional*, nov. 2011.

[152] KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Proceedings of the 14th international joint conference on Artificial intelligence*. San Francisco: Morgan Kaufmann, 1995. v. 2, p. 1137–1143.

[153] KOHAVI, R.; PROVOST, F. Glossary of Terms. *Machine Learning*, v. 30, n. 2/3, p. 271–274, 1998.

[154] WEISS, S. M.; KULIKOWSKI, C. A. *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems*. San Francisco, CA: Morgan Kaufmann, 1991.

[155] COHEN, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, v. 20, n. 1, p. 37–46, 1960.

[156] BATISTA, G. E. A. P. A.; PRATI, R. C.; MONARD, M. C. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explor. Newsl.*, ACM, New York, v. 6, n. 1, p. 20–29, jun 2004. ISSN 1931-0145.

[157] LANDIS, J. R.; KOCH, G. G. The measurement of observer agreement for categorical data. *Biometrics*, v. 33, n. 1, p. 159–174, mar. 1977.

[158] EGAN, J. P. *Signal detection theory and ROC analysis*. New York: Academic Press, 1975. (Series in Cognition and Perception).

[159] FAWCETT, T. An introduction to roc analysis. *Pattern Recogn. Lett.*, Elsevier Science Inc., New York, v. 27, n. 8, p. 861–874, jun. 2006.

[160] PRATI, R. C.; BATISTA; MONARD, M. C. Curvas roc para avaliação de classificadores. *Revista IEEE América Latina*, v. 6, n. 2, p. 215–222, jun 2008.

[161] FLACH, P.; WU, S. Repairing concavities in roc curves. In: *Proc. 2003 UK Workshop on Computational Intelligence*. [S.l.]: University of Bristol, 2003. p. 38–44.

[162] MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. In: REZENDE, S. O. (Ed.). *Sistemas Inteligentes - Fundamentos e Aplicações*. 1. ed. Barueri: Manole, 2003. v. 1, cap. 4, p. 89–114.

[163] COVER, T. M.; THOMAS, J. A. *Elements of information theory*. New York: Wiley-Interscience, 1991.

[164] FURUICHI, S. Information theoretical properties of tsallis entropies. *Journal of Mathematical Physics*, AIP, v. 47, n. 2, 2006.

[165] CACHIN, C. *Entropy Measures and Unconditional Security in Cryptography*. Tese (Doutorado) — Swiss Federal Institute of Technology Zürich, 1997.

[166] LIN, S.; JR., D. J. C. *Error Control Coding: Fundamentals and Applications*. [S.l.]: Prentice-Hall, 1983. TUB-HH 2413-469 3.

[167] SKLAR, B. *Digital communications: fundamentals and applications*. Upper Saddle River: Prentice-Hall, Inc., 1988.

[168] LIDL, R.; NIEDERREITER, H.; COHN, P. M. *Finite Fields*. Cambridge: Cambridge University Press, 1997.

[169] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1. ed. Boston, USA: Addison-Wesley Longman Publishing Co., 1989.

[170] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.

[171] SAIT, S. M.; YOUSSEF, H. *Iterative Computer Algorithms with Applications in Engineering: Solving Combinatorial Optimization Problems*. 1st. ed. Los Alamitos, USA: IEEE Computer Society Press, 1999.

[172] MICHALEWICZ, Z. *Genetic algorithms + data structures = evolution programs*. 3. ed. London, UK: Springer-Verlag, 1996.

[173] ZUBEN, F. V. Computação Evolutiva: uma abordagem pragmática 2003. Disponível em: <ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/tutorial/tutorialEC.pdf>. Acesso em: 10 nov. 2012.

[174] EIBEN, A. E.; SMITH, J. E. *Introduction to Evolutionary Computing*. [S.l.]: Springer, 2003.

[175] SOUZA, E. P. de. *Estudo sobre sistema de detecção de intrusão por anomalias: uma abordagem utilizando redes neurais*. Dissertação (Mestrado) — Universidade de Salvador, Salvador, 2008.

[176] ABBAS, A. K.; LICHTMAN, A. H. H.; PILLAI, S. *Imunologia celular e molecular*. 7. ed. [S.l.]: Elsevier Brasil, 2012. 560 p.

[177] CARTER, J. H. The immune system as a model for classification and pattern recognition. *Journal of the American Informatics Association*, v. 7, n. 1, p. 28–41, 2000.

[178] WATKINS, A. *AIRS: A Resource Limited Artificial Immune Classifier*. Dissertação (Mestrado) — Mississippi State University, 2001.

[179] BROWNLEE, J. *Artificial Immune Recognition System (AIRS) - A Review and Analysis*. [S.l.], jan. 2005.

[180] A., W. *Exploiting Immunological Metaphors in the Development of Serial, Parallel, and Distributed Learning Algorithms*. Tese (Doutorado) — Mississippi State University, 2005.

[181] TAVALLAEE, M. et al. A detailed analysis of the kdd cup 99 data set. In: *Proceedings of the 2009 IEEE Symposium Computational Intelligence for Security and Defense Applications (CISDA'09)*. [S.l.]: IEEE Computer Society, 2009. p. 53–58.

[182] NSL-KDD data set for network-based intrusion detection systems. Disponível em: <http://nsl.cs.unb.ca/NSL-KDD/>. Acesso em: 14 fev. 2012.

APÊNDICE A

Lista de Artigos Produzidos

Segue abaixo a lista de artigos produzidos ao longo deste trabalho de tese.

[1] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. **Decision tree based on Shannon, Renyi and Tsallis entropies for intrusion tolerant systems**. *Fifth International Conference on Internet Monitoring and Protection*, IEEE Computer Society, Barcelona, v. 0, p. 117-122, maio 2010.

[2] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. **Artificial immune systems applied in intrusion tolerant systems**. *Wireless Systems International Meeting - RFID: trends to the future*, Campina Grande, maio 2010.

[3] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. **Árvores de decisão baseadas nas entropias de Shannon, Rényi e Tsallis para sistemas tolerantes a intrusão**. *La Novena Conferencia Iberoamericana en Sistemas*, Cibernética e Informática, Orlando, jun. 2010.

[4] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. **A comparative study of use of Shannon, Rényi and Tsallis entropy for attribute selecting in network intrusion detection**. *IEEE International Workshop on Measurements and Networking*, p. 77-82, 2011.

[5] LIMA, C. F. L.; ASSIS, F. M. de; SOUSA, C. P. de. **Detecção de intrusos em redes de computadores com usos de códigos corretores de erros e medidas de informação**. X Congresso Brasileiro de Inteligência Computacional, nov. 2011.

[6] LIMA, C.; ASSIS, F. M.; SOUZA, C. P. de. **A comparative study of use of Shannon, Rényi and Tsallis entropy for attribute selecting in network intrusion detection**. In: YIN, H.; COSTA, J. A. F.; BARRETO, G. (Ed.). *Intelligent Data Engineering and Automated Learning* - IDEAL 2012. [S.l.]: Springer, 2012, (Lecture Notes in Computer Science, v. 7435). p. 492-501.

[7] LIMA, C. F. L.; ASSIS, F. M. de; SOUZA, C. P. de. **An empirical investigation of attribute selection techniques based on Shannon, Rényi and Tsallis entropies for network intrusion detection. American Journal of Intelligent Systems**, v. 2(5), p. 111-117, 2012.

Os referidos artigos encontram-se nos anexos desta tese, na ordem que foram ora apresentados.

APÊNDICE B

Materiais e Métodos

A avaliação da capacidade de generalização de um modelo é importante, pois permite uma estimativa da precisão do modelo ao classificar dados que não foram utilizados no processo de construção do modelo. Além disso, a referida estimativa permite a comparação de desempenho entre diferentes classificadores sobre uma mesma base de dados de teste. Para tal finalidade, existem diversas métricas, dentre as quais serão apresentadas neste apêndice, as utilizadas neste trabalho.

B.1 Método de Amostragem

No método de avaliação r-validação cruzada (k-*fold cross validation*) [152], a base de dados inicial é aleatoriamente dividida em r partições mutuamente exclusivas, nas quais as classes são representadas aproximadamente na mesma proporção da base de dados completa. O esquema de aprendizagem é treinado com $r-1$ *folds* e o restante é usado para teste do modelo. O teste é realizado r vezes com diferentes amostragens. A medida de desempenho corresponde à média dos percentuais de acerto das r iterações.

Segundo [101], uma grande quantidade de testes sobre diversas bases de dados têm mostrado que o valor de $r = 10$ é o mais adequado para se obter uma estimativa razoável da precisão do classificador. Portanto, na prática, a validação cruzada com dez *folds* vem sendo o método mais utilizado e foi o escolhido nos experimentos realizados nesta tese.

B.2 Estimativas de Desempenho dos Modelos de Classificação

Na predição multiclasses, é frequente que os resultados da classificação sejam expressados na forma de uma matriz de confusão, cuja dimensão ($M \times M$) é dada pelo número de classes do problema.

B.2.1 Matriz de confusão

A matriz de confusão de um classificador indica o número de classificações corretas *versus* as predições efetuadas para cada caso, sobre um conjunto de instâncias [153]. Conforme ilustrado na Tabela B.1, na matriz de confusão A, existe uma coluna j e uma linha i para cada classe $c_1, ..., c_M$, em que a linha indica a classe verdadeira da instância avaliada e a coluna indica a classe predita. A diagonal principal da matriz exibe o número de acertos para as classes analisadas, enquanto que os elementos fora da diagonal, o número de erros de classificação.

Tabela B.1 Matriz de confusão A para 5 classes.

	Predito					
Real	c_1	c_2	c_3	c_4	c_5	Total
c_1	20	1	0	0	0	21
c_2	1	8	0	0	1	10
c_3	0	2	45	0	0	47
c_4	0	0	0	3	0	3
c_5	0	0	1	2	4	7
Total	21	11	46	5	5	88

B.2.2 Medidas de desempenho

Seja a matriz de confusão, ilustrada na Tabela B.1, algumas medidas de desempenho podem ser calculadas, tais como [154]:

- O número de classificações corretas (fazendo referência aos verdadeiros positivos - VP) está representado na diagonal principal;
- Fora da diagonal principal, podem ser verificados o número de Falsos Negativos (FN) e o número de Falsos Positivos (FP) para cada classe, por intermédio das linhas e colunas, respectivamente;
- A precisão do classificador para a classe i, denotada aqui por $PR_i(\hat{f})$, representa a proporção de instâncias preditas corretamente e o número de instâncias preditas para a referida classe (somatória na coluna), dada pela Equação (B.1):

$$PR_i(\hat{f}) = \frac{a_{ii}}{\sum_{l=1}^{M} a_{li}}; \tag{B.1}$$

- A taxa de falso negativo - TFN do classificador para a classe i pode ser encontrada pela Equação (B.2):

$$TFN_i(\hat{f}) = 1 - PR_i; \tag{B.2}$$

- A taxa de verdadeiro positivo - TVP (ou revocação - *recall*) do classificador para a classe i é a razão entre o número de instâncias preditas corretamente e o total de instâncias pertencentes à referida classe (somatória na linha), dada pela Equação (B.3):

$$TVP_i(\hat{f}) = \frac{a_{ii}}{\sum_{j=1}^{M} a_{ij}}; \tag{B.3}$$

- A taxa de falso positivo - TFP do classificador para a classe i é a razão entre o número de instâncias preditas incorretamente e o total de instâncias não pertencentes à referida classe, dada pela Equação (B.4):

$$TFP_i(\hat{f}) = \frac{\sum_{j=1}^{M} a_{i\neq j}}{\sum_{l=1}^{M}\sum_{j=1}^{M} a_{lj}}, \text{ em que} l \neq i; \tag{B.4}$$

- A acurácia do classificador é a proporção do número de instâncias recuperadas em todas as classes (somatória na diagonal principal) e o número total de instâncias testadas, dada pela Equação (B.5):

$$Acc(\hat{f}) = \frac{\sum_{i=1}^{M} a_{ii}}{\sum_{i=1}^{M}\sum_{j=1}^{M} a_{ij}}; \tag{B.5}$$

- O complemento da acurácia, a taxa de erro de classificação, pode ser obtida por meio da Equação (B.6), representada como:

$$err(\hat{f}) = 1 - Acc(\hat{f}). \tag{B.6}$$

Entre as medidas apresentadas, a acurácia é a mais conhecida e talvez a mais utilizada. No entanto, a medida de acurácia não leva em consideração os acertos devido ao acaso. Portanto, será apresentada uma medida de concordância, nomeada estatística *kappa* [155], para decidir qual classificador é mais adequado para o problema proposto.

Outro aspecto relevante na avaliação de desempenho de algoritmos de classificação é com relação à quantidade de amostras e a distribuição dessas entre as classes no conjunto de treinamento. Quando o número de instâncias é representativo, mas não se encontram em equilíbrio para cada classe, principalmente em situações em que as informações associadas a determinadas classes são mais difíceis de se obter, surge o problema conhecido na literatura como problema de classes desbalanceadas [156].

B.2.3 Estatística *kappa*

A estatística *kappa*, também conhecida como índice ou coeficiente *kappa* de Cohen [155] pode ser utilizada para avaliar a concordância entre observações em uma mesma unidade amostral. É uma medida baseada na diferença entre a concordância real da classificação e a con-

cordância por puro acaso. Para estimar o coeficiente *kappa*, pode ser utilizada uma matriz de confusão (Figura B.1) e a Equação (B.7), dada por:

$$\kappa = \frac{Pr(o) - Pr(e)}{1 - Pr(e)}, \tag{B.7}$$

em que $Pr(o)$ indica a probabilidade de sucesso (ou concordância) observada e $Pr(e)$ é a probabilidade de sucesso esperado por puro acaso. Para estimar as probabilidades referentes a cada classe são utilizados os dados observados na tabela de confusão.

A probabilidade $Pr(o)$ é dada por:

$$Pr(o) = \frac{\sum_{i=1}^{M} O_{ii}}{m}, \tag{B.8}$$

em que O_{ii} corresponde ao número de acertos observados para a i-ésima classe (extraídos da diagonal principal da matriz), m é o número total de casos e M é o número de classes.

A probabilidade $Pr(e)$ é dada por:

$$Pr(e) = \frac{\sum_{i=1}^{M} \pi_i^a \pi_i^b}{m^2}, \tag{B.9}$$

em que π_i^a representa o número total de casos avaliados pertencentes à classe i (total marginal da linha i), π_i^b representa o número de classificações preditas para a referida classe i (total marginal da coluna i) e m é o número total de casos.

Por exemplo, supondo-se a avaliação de dois classificadores A e B e um conjunto de dados contendo 3 classes, com distribuição desigual, conforme ilustrado na Tabela B.2.

Tabela B.2 Matrizes de confusão de dois classificadores para um problema com 3 classes.

Classificador A

	Predito			
Real	c_1	c_2	c_3	Total
c_1	18	1	2	21
c_2	0	8	1	9
c_3	3	0	44	47
Total	21	9	47	77

Classificador B

	Predito			
Real	c_1	c_2	c_3	Total
c_1	20	0	1	21
c_2	0	6	3	9
c_3	2	1	44	47
Total	22	7	48	77

A estimativa da taxa de erro de classificação e do coeficiente κ é o seguinte:

- Classificador A
  $err(A) = \frac{7}{77} \Rightarrow err(A) = 0,0909$
  $Pr(o) = \frac{18+8+44}{77} = 0,909$
  $Pr(e) = \frac{21\times21+9\times9+47\times47}{77^2} = 0,46054$
  $\kappa_A = \frac{0,909-0,46054}{1-0,46054} \Rightarrow \kappa_A = 0,8313$

- Classificador B
  $err(B) = \frac{7}{77} \Rightarrow err(B) = 0,0909$

$Pr(o) = \frac{20+6+44}{77} = 0,909$
$Pr(e) = \frac{21\times22+9\times7+47\times48}{77^2} = 0,46905$
$\kappa_B = \frac{0,909-0,46028}{1-0,46028} \Rightarrow \kappa_B = 0,8286$

Com os valores de κ obtidos, é necessário interpretá-los. Baseado em [157], uma possível interpretação do desempenho do classificador em função do coeficiente *kappa*, pode ser obtida por meio da Tabela B.3.

Tabela B.3 Interpretação do coeficiente *kappa*

Valor de *kappa*	Interpretação
$\kappa \leq 0,00$	Péssima
$0,00 < \kappa \leq 0,20$	Ruim
$0,21 \leq \kappa \leq 0,40$	Razoável
$0,41 \leq \kappa \leq 0,60$	Boa
$0,61 \leq \kappa \leq 0,80$	Muito Boa
$0,81 \leq \kappa \leq 1,00$	Excelente

Ambos classificadores têm a mesma estimativa de taxa de erro de classificação $(0,0909)$. Entretanto, com base nos resultados obtidos para o coeficiente κ, pode-se perceber que o classificador A consegue discriminar melhor as três classes. Além disso, a interpretação a cerca dos dois classificadores é que ambos tem um desempenho de classificação excelente.

Em síntese, o coeficiente κ compensa os acertos devido ao acaso e dá uma ideia dos pontos fortes do classificador para cada classe. É uma alternativa mais robusta, especialmente para dados com uma distribuição desigual de classes, fato que ocorre na base de dados de teste utilizada nesta tese.

No aprendizado com dados desbalanceados, em geral, a tendência é induzir modelos de classificação que favorecem as classes com maior probabilidade de ocorrência, resultando em baixa precisão para as classes minoritárias. Isto pode ser indesejável quando as classes minoritárias são aquelas que possuem uma informação muito importante. Por exemplo, na detecção de intrusão em redes de computadores, o número de atividades normais é muito superior ao número de atividades suspeitas.

Ainda para contornar o problema de classes desbalanceadas, pode-se utilizar outras métricas que sejam indicadas para avaliação desses classificadores, como é o caso das métricas da análise ROC.

B.2.4 Análise ROC

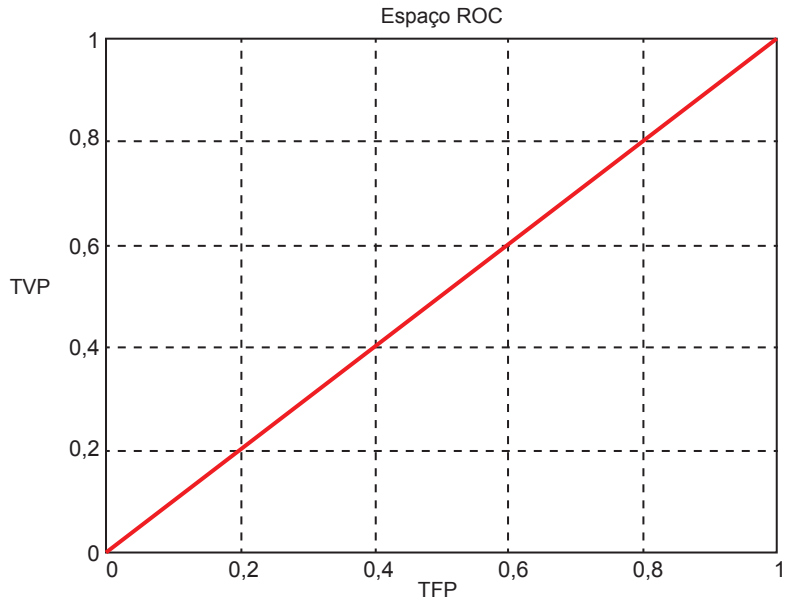
A análise ROC (*Receiver Operating Characteristic*) é uma técnica visual, que foi originalmente desenvolvida na Teoria de Detecção de Sinais [158] e, nos últimos anos, tem sido usada pelas comunidades de Aprendizado de Máquina e Mineração de Dados para avaliação,

organização e seleção de classificadores binários [159], assim como fornece métricas para avaliação sensíveis ao problema de desbalanceamento.

O gráfico ROC compara diferentes classificadores, com base nas suas respectivas taxas de verdadeiros positivos (TVP) e taxas de falsos positivos (TFP), calculadas a partir de sua matriz de confusão (vide Tabela B.1), em um espaço bidimensional. Esse espaço bidimensional é também conhecido como espaço ROC. Um classificador discreto é representado por um ponto no espaço ROC [160].

Para se construir o gráfico ROC, plota-se a TFP no eixo X e a TVP no eixo Y. Na Figura B.1 é ilustrado um gráfico ROC, identificando quatro regiões importantes, que serão descritas a seguir, e uma linha diagonal que representa classificadores aleatórios.

Figura B.1 Gráfico ROC



Considerando duas classes (positiva e negativa), classificadores representados na região próxima do ponto (0,0) do gráfico ROC tendem a rotular as instâncias como negativa. Dessa forma, o número de instâncias negativas rotuladas erroneamente, em geral, é baixo (TFP próxima de 0), assim como o número de instâncias positivas rotuladas corretamente (TVP próxima de 0).

De modo inverso, os classificadores que tendem a rotular, com maior frequência, as instâncias como positiva, são representados na região próxima do ponto (1, 1). Assim, o número de instâncias negativas rotuladas erroneamente, em geral, é elevado (TFP próxima de 1), bem como o número de instâncias positivas rotuladas corretamente (TVP próxima de 1).

O ponto (0, 1) representa o modelo perfeito, na qual todas as instâncias positivas e negativas são corretamente classificadas. Portanto, os classificadores representados na região próxima a este ponto, estão mais próximos da classificação perfeita. Em contrapartida, os clas-

sificadores, representados na região próxima ao ponto (1, 0), possuem informações com capacidade de distinguir as classes, porém não as utilizam devidamente [161], gerando um número elevado de classificações incorretas.

Em geral, os classificadores representados por pontos próximos à linha diagonal, são considerados classificadores aleatórios, ou seja, não possuem informação sobre a classe e rotulam as classes das instâncias testadas aleatoriamente.

A maneira mais utilizada para comparar curvas ROC é calcular a área abaixo da curva ROC (*Area Under Curve* ou AUC) [159]. **AUC** é uma medida escalar variável compreendida no intervalo entre 0 e 1. A partir da medida, experiências podem ser realizadas em busca do classificador que produza o maior valor de AUC. Assim, o classificador que tiver maior área é considerado o melhor.

No âmbito da análise ROC multiclasse, de acordo com [159], uma estratégia que pode ser aplicada é gerar M curvas ROC, sendo uma para cada classe. Se C é o conjunto de todas as classes, o gráfico ROC i plota a performance do classificador, usando a classe c_i como a classe positiva e todas as demais classes como a classe negativa (abordagem um-contra-todos).

Uma alternativa para o cálculo da área, é calcular a AUC para cada classe e depois somar essas áreas de forma ponderada, por meio da Equação (B.10). Entretanto, essa abordagem é sensível ao desbalanceamento entre classes [159].

$$AUC_{Total}(\hat{f}) = \sum_{c_i \in C} AUC(c_i)p(c_i), \tag{B.10}$$

em que $AUC(c_i)$ é a área da curva c_i e $p(c_i)$ é a probabilidade de um elemento ser da classe i.

Para a limitação anterior, uma alternativa é utilizar uma medida que não seja sensível à distribuição das classes, dada pela Equação (B.11):

$$AUC_{Total}(\hat{f}) = \frac{2}{|C|\,(|C|-1)} \sum_{\{c_i,c_j\} \in C} AUC(c_i, c_j), \tag{B.11}$$

em que $AUC(c_i, c_j)$ é calculado para todos os pares de classes, ou seja, $|C|\,(|C|-1)/2$ vezes.

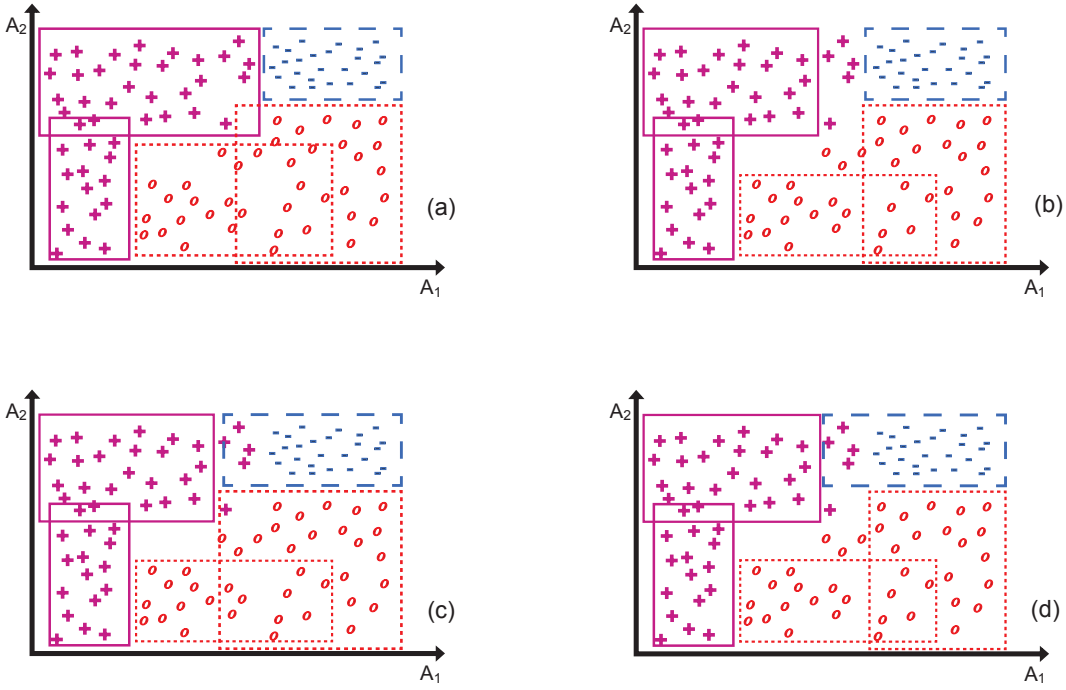
B.2.5 Análise de consistência e completude

Nos experimentos realizados usando códigos corretores de erros, ocorreram falhas de decodificação, ou seja, não foi possível classificar as instâncias completamente, bem como ocorreram classificações incorretas.

Uma hipótese $\hat{f}$ pode ser avaliada com relação à sua completude, isto é, se ela classifica todas as instâncias e com relação à sua consistência, se ela classifica corretamente as instâncias [162]. Portanto, dada uma hipótese $\hat{f}$, ela pode ser: (a) completa e consistente; (b) incompleta e consistente; (c) completa e inconsistente ou (d) incompleta e inconsistente.

Na Figura B.2 é ilustrado um exemplo dos quatro casos mencionados, considerando dois atributos A_1 e A_2, três classes (o, +, -), bem como as hipóteses induzidas para cada classe. Elas são representadas por duas regiões indicadas por linhas sólidas para a classe (+), duas regiões indicadas por linhas pontilhadas para a classe (o) e uma região indicada por linhas tracejadas para a classe (-).

Figura B.2 Relação entre completude e consistência de uma hipótese (a) completa e consistente; (b) incompleta e consistente; (c) completa e inconsistente ou (d) incompleta e inconsistente [162], adaptado pela autora.



APÊNDICE C

Medidas de Informação

Neste Apêndice, são apresentadas, de forma sucinta, as medidas de informação de Shannon [84], Renyi [40] e Tsallis [42], que podem ser utilizadas na construção de árvores de decisão e que são objeto de estudo deste trabalho de tese. Estudos mais aprofundados podem ser encontrados em [40, 42, 84, 163–165].

C.1 Medidas de Informação de Shannon

O conceito de entropia na teoria da informação foi introduzida por Claude Shannon com a publicação do artigo *The Mathematical Theory of Communication* [84] em 1948, no qual é apresentado um novo modelo matemático para o estudo dos sistemas de comunicação. Em seu modelo de comunicação (fonte de informação-canal de comunicação-receptor) [84], a quantidade de informação transmitida em uma mensagem é função da previsibilidade da mensagem [163].

Pode-se definir uma fonte de informação como sendo um modelo matemático para um sistema físico que produz uma sucessão de símbolos de maneira aleatória chamados eventos. O espaço contendo todos os eventos possíveis é usualmente chamado de alfabeto da fonte de informação, ao qual é atribuído um conjunto de probabilidades de ocorrência. Uma fonte discreta de informação gera símbolos de um alfabeto $\mathcal{A} = \{s_i, 1 \leq i \leq M\}$ com probabilidade de ocorrência p_i, tal que $\sum_{i=1}^{M} p_i = 1$. No caso particular em que a fonte é discreta sem memória, os símbolos gerados são estatisticamente independentes.

A noção de entropia está relacionada com a incerteza de se obter uma informação e com a capacidade informativa da fonte. Essa incerteza caracteriza o ganho de informação que a ocorrência de um evento pode fornecer. Em resumo, uma fonte de informação que responda com uma única mensagem a toda e qualquer pergunta não transmite informação, já que não há incerteza quanto à mensagem. Portanto, quanto maior for a incerteza, maior será o potencial de informação dessa fonte.

Dessa forma, seja a variável aleatória discreta X (equivalente a uma fonte discreta sem memória), com alfabeto $\mathcal{A}$, que pode assumir um número finito de valores possíveis $x_i \in \mathcal{A}$, com uma distribuição de probabilidade $p_i = Pr[X = x_i]$, associada a cada um destes valores, tal que $p_i = \{p_1, ..., p_M\}, \sum_{i=1}^{M} p_i = 1$.

Define-se entropia, representada por $H(X)$, a esperança matemática para a quantidade de informação contida em um evento possível qualquer, ou seja, a quantidade de informação média necessária para descrever os acontecimentos, que pode ser representada por [84]:

$$\begin{aligned} H(X) = &\ \mathbb{E}\left[\log \tfrac{1}{Pr[X]}\right] \\ = &\ -\sum_{i=1}^{M} p_i \log_2 p_i \text{ (bits/símbolo).} \end{aligned} \tag{C.1}$$

Com base na Equação (C.1), pode-se ter as seguintes interpretações para a entropia $H(X)$:

- A incerteza sobre o evento de X antes de observá-lo, isto é, em relação à ocorrência de um símbolo na saída de uma fonte discreta;
- A quantidade de informação média esperada de um evento de X que foi ganha depois de realizar a observação;
- O número médio de *bits* esperado necessário para descrever um evento de X;
- O menor número médio de perguntas binárias necessário para identificar a saída de uma fonte $(X = x_i)$.

A entropia de uma fonte de informação é sensível à quantidade de símbolos que esta é capaz de emitir. Por exemplo, para uma fonte discreta sem memória com M símbolos do alfabeto $\mathcal{A}$ e probabilidade $p_i = p_1, ..., p_M$, a entropia é limitada segundo a desigualdade a seguir:

$$0 \leq H(X) \leq \log_2 M, \tag{C.2}$$

em que o limite inferior, $H(X) = 0$, caracteriza a ausência total de incerteza sobre a saída da fonte (se e somente se $p_i = 1$ para algum i) e o limite superior, $H(X) = \log_2 M$, corresponde ao máximo de incerteza, ocorrendo a igualdade para acontecimentos equiprováveis $Pr[X = x_i] = p_i = \frac{1}{M}, \forall x_i \in \mathcal{A}$.

Entretanto, conforme pode ser observado por meio da Equação (C.1), a entropia $H(X)$ não depende dos valores que a variável aleatória X assume, nem de qualquer outra característica, a não ser das probabilidades associadas a esses valores e assim, a notação $H(p_1, ..., p_M)$ pode ser empregada para ressaltar a distribuição de probabilidades.

Shannon [84] definiu outros conceitos básicos na teoria da informação em relação ao tratamento de dados correlacionados, que serão abordados a seguir:

- Entropia Conjunta: a entropia conjunta $H(X, Y)$ das variáveis aleatórias X e Y, assumindo, respectivamente, um número finito de valores possíveis x_i e y_j, com distribuições

$p_i = Pr[X = x_i]$ e $q_j = Pr[Y = y_j]$ e com distribuição de probabilidade conjunta $p_{ij} = Pr[X = x_i, Y = y_j]$ é definida por

$$\begin{aligned} H(X,Y) = & -\sum_{i,j} p_{ij} \log_2 p_{ij} \\ = & H(X) + H(Y|X) \\ = & H(Y) + H(X|Y), \end{aligned} \tag{C.3}$$

em que $H(X|Y)$ ou $H(Y|X)$ refere-se à entropia condicional, que será definida no próximo item.

A entropia conjunta de n variáveis aleatórias é definida de forma análoga. Os limites da entropia conjunta são determinados pela expressão $0 \leq H(X,Y) \leq H(X) + H(Y)$. No caso particular das v.a X e Y serem independentes, a entropia conjunta é dada por $H(X,Y) = H(X) + H(Y)$.

- Entropia Condicional: a entropia condicional é a medida da incerteza do valor de X dado o conhecimento do valor de Y, também chamado de equivocação de X em relação a Y, é o valor médio esperado, ponderado para todas as possíveis ocorrências de Y, dada por:

$$\begin{aligned} H(X|Y) = & \sum_j H(X|Y = y_j) Pr[Y = y_j] \\ = & H(X,Y) - H(Y). \end{aligned} \tag{C.4}$$

Da mesma forma, $H(X|Y) = H(X,Y) - H(X)$. Assim, pode-se interpretar que a equivocação de X (ou Y) em relação a Y (ou X) é igual à incerteza total do espaço conjunto (X,Y), do qual é retirada a incerteza em relação a Y (ou X). Contudo, se X e Y são independentes, $H(X|Y) = H(X)$ ou $H(Y|X) = H(Y)$, ou seja, o conhecimento de Y(ou X) não traz informação sobre X (ou Y). Usando a desigualdade fundamental da teoria da informação, demonstra-se que o condicionamento somente pode reduzir a entropia, então pode-se afirmar que $H(X) \geq H(X|Y)$.

Uma outra quantidade a ser definida é a informação mútua contida no par X e Y, que mede quanta informação estas variáveis aleatórias têm em comum. Em resumo, a informação mútua mede a redução da incerteza de uma variável aleatória dado o conhecimento de outra (Equação C.5). Assim, quanto maior for a informação mútua, maior será a redução da incerteza. Se X e Y forem independentes, a informação mútua entre estas duas variáveis será igual a zero.

$$\begin{aligned} I(S;Z) = & I(Z;S) \\ = & H(X) - H(X|Y) \\ = & H(Y) - H(Y|X) \\ = & H(X) + H(Y) - H(X,Y). \end{aligned} \tag{C.5}$$

Dentro da teoria da informação, foram formuladas propostas de generalização da entropia, dentre elas, as medidas de informação de Rényi [40] e Tsallis [42] que serão brevemente descritas a seguir.

C.2 Medidas de Informação de Rényi

Em 1961, Alfred Rényi [40] evidenciou, por meio de um conjunto de postulados, que outras medidas de quantidade poderiam ajustar-se igualmente, ou talvez melhor, a uma medida de informação, surgindo a ideia de entropia generalizada. Rényi [40] propôs então uma entropia parametrizada por α ($\alpha > 0$ e $\alpha \neq 1$) que contém a entropia de Shannon [84] como caso limite, chamada entropia de Rényi [40]. Assim, a entropia de Rényi [40] de ordem α de uma variável aleatória X, com uma distribuição discreta de probabilidade p_i, é dada por:

$$R_\alpha(X) = \frac{1}{1-\alpha}\log \sum_{i=1}^{M} p_i^\alpha, \tag{C.6}$$

em que $\sum_{i=1}^{k} p_i = 1$ e $\lim_{\alpha\to 1} R_\alpha(X) = H(X)$.

Para $\alpha \in (0,1)$, a informação mútua, chamada de α-informação mútua, pode ser generalizada, usando a entropia de Rényi [40], como:

$$I_\alpha(X;Y) = R_\alpha(X) - R_\alpha(X|Y). \tag{C.7}$$

C.3 Medidas de Informação de Tsallis

Em 1988, Constantino Tsallis [42] definiu outra forma generalizada de entropia, tendo como caso particular a entropia de Shannon [84]. Para uma distribuição de probabilidade $p_i = Pr[X = x_i]$ de uma variável aleatória X, a entropia de Tsallis de uma variável aleatória X é dada por:

$$S_\alpha(X) = \frac{1}{\alpha-1}\left(1 - \sum_{i=1}^{M} p_i^\alpha\right), \tag{C.8}$$

em que $\alpha \geq 0$ e $\lim_{\alpha\to 1} S_\alpha(X) = H(X)$.

Para $\alpha > 1$, a informação mútua de Tsallis [42] entre duas variáveis aleatórias (X, Y) é definida como [164]:

$$I_\alpha(X;Y) = S_\alpha(X) - S_\alpha(X|Y). \tag{C.9}$$

APÊNDICE D

Conceitos sobre Códigos Corretores de Erros

Este apêndice tem como objetivo introduzir um conjunto mínimo de conceitos necessários para o entendimento sobre os códigos corretores de erros, com ênfase aos códigos de bloco lineares cíclicos. Também neste capítulo é apresentada uma importante família de códigos cíclicos, que será utilizada nos experimentos, chamada de códigos BCH (Bose-Chaudhuri-Hocquenghem). Porém para melhor entendimento deste apêndice, é necessário um conhecimento básico sobre álgebra abstrata e aritmética de corpo finito. Para estudos mais avançados são recomendados os livros [24, 37, 133, 166].

D.1 Códigos de Blocos Lineares

Um código corretor de erros é basicamente uma estratégia de acrescentar, de forma controlada, redundância à informação que se queira transmitir ou armazenar, gerando uma palavra código. Como esta palavra código, ao ser transmitida, pode sofrer alterações devido a ruídos e interferências no meio de transmissão, é possível, dependendo do código utilizado, detectar e até mesmo corrigir os erros e recuperar a informação original pela retirada da redundância.

Assim, a teoria de códigos passou a ser parte integrante da teoria da comunicação de dados, visto que o aumento da demanda na qualidade das comunicações e os progressos na tecnologia da informação tem impulsionado a implementação de procedimentos de correção de erro e a expansão na área de aplicação de códigos corretores de erros.

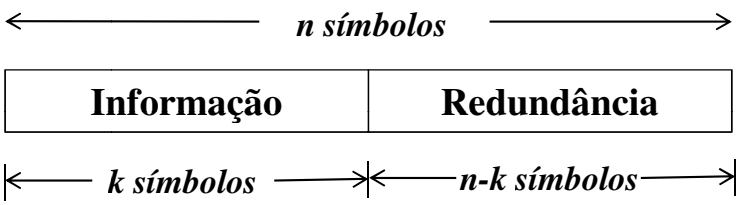
De acordo com o modo com que a redundância é adicionada às informações, os códigos corretores de erros são divididos em duas classes: os códigos de bloco e os códigos convolucionais. Um código de bloco é caracterizado geralmente na forma (n, k). Neste tipo de codificação, a informação original é fragmentada em segmentos com k símbolos. O codificador transforma esses k símbolos em um bloco codificado de n símbolos. Adicionam-se portanto $n - k$ símbolos redundantes a cada bloco de k símbolos de informação com o propósito de

corrigir, ou pelo menos detectar erros. A taxa de um código de bloco linear é definida como $R = k/n$. Este é um parâmetro importante que caracteriza a diminuição da transmissão de informação devido à redundância [84].

Considerando $GF(q)$ um corpo finito de q elementos, um código de bloco C consiste num conjunto de *n*-tuplas chamadas de palavras código (vetores) de comprimento n, com elementos em $GF(q)$. Quando $q = 2$ trata-se de um código binário, quando $q > 2$ o código é dito não binário. O espaço de *n*-tuplas de um código é constituído por q^n *n*-tuplas, das quais q^k são palavras código, ou seja, um código binário possui tamanho $|C| = 2^k$ palavras binárias de n bits [167].

Para fins de ilustração, na forma sistemática de codificação, os símbolos de redundância são anexados aos símbolos de informação (podendo ser antes ou depois), conforme a Fig. D.1, ou seja, para cada sequência de k símbolos de informação, existe uma palavra de código distinta de n símbolos.

Figura D.1 Codificação sistemática de uma informação [133]



Nos códigos convolucionais, a sequência codificada de n símbolos depende não somente da informação na entrada do codificador, mas também das informações anteriormente processadas. Assim, a sequência codificada para um conjunto de k símbolos de informação não é única.

Os códigos de bloco podem ser classificados como lineares ou não lineares. Um código de bloco $C \subset GF(q)^n$ é linear se for um subespaço vetorial de *n*-tuplas de $GF(q)^n$ [168], tal que a soma de quaisquer duas palavras código e o produto de uma delas por um elemento de $GF(q)$ tem como resultado uma palavra código. Sendo um espaço vetorial, a palavra código nula (vetor nulo) sempre pertence ao código.

D.2 Distância de *Hamming* e a Capacidade de Correção de Erros

A distância de *Hamming* fornece uma informação sobre a proximidade entre as palavras do código. Dadas as palavras código $v = (v_1, v_2, ..., v_n), u = (u_1, u_2, ..., u_n) \in C$, a distância de *Hamming* $d(v, u)$ é definida como o número de coordenadas em que o par de palavras código distintas v e u diferem. Isto é:

$$\underset{v,u\in C,v\neq u}{d(v,u)} = |\{i|v_i \neq u_i, 1 \leq i \leq n\}|, \tag{D.1}$$

em que a distância de *Hamming* entre duas palavras é sempre um número inteiro.

Algumas das propriedades da distância de *Hamming* é que esta é não negativa, simétrica e satisfaz a desigualdade triangular [24].

A distância mínima de um código de bloco linear, d_{min}, é dada pela menor distância de *Hamming* entre todos os possíveis pares distintos de palavras código em C.

$$d_{min} = \underset{v,u\in C}{min}\{d(v,u)|v \neq u\}. \tag{D.2}$$

Visto que o código de bloco linear C é um subespaço vetorial, a distância de *Hamming* entre a palavra código nula e uma palavra código qualquer é igual ao peso de *Hamming* desta última. Assim, o peso de v é dado por:

$$w(v) = d(0,v). \tag{D.3}$$

Define-se como peso do código C, o menor peso de *Hamming* de todas as palavras código não nulas do código. Isto é,

$$w_{min} = \underset{v\in C,v\neq 0}{min}\{w(v)\}. \tag{D.4}$$

Sendo o código linear, então $v-u$ é também uma palavra código que dista d da palavra código nula. Portanto, toda distância entre palavras distintas do código C é também o peso de alguma palavra do referido código, ou seja:

$$\begin{aligned} \underset{v,u\in C,v\neq u}{d(v,u)} &= |\{i|v_i \neq u_i, 1 \leq i \leq n\}| \\ &= |\{i|v_i - u_i \neq 0, 1 \leq i \leq n\}| \\ &= d(v-u,0) \\ &= w(v-u). \end{aligned} \tag{D.5}$$

Consequentemente, tem-se que:

$$d_{min} = w_{min}. \tag{D.6}$$

Dessa forma, a determinação de um código linear que corrija t erros, implica que o peso mínimo deste código seja $w_{min} \geq 2t+1$.

Seja v uma palavra código de C, uma esfera de *Hamming* $S_t(v)$, de raio t e centro em v, contém todas as palavras que recebidas devem ser decodificadas para a palavra código no centro da esfera de *Hamming* menor ou igual a t.

$$S_t(v) = \{x|d(v,x) \leq t\}. \tag{D.7}$$

Se até t erros ocorreram, então a palavra recebida está sempre naquela esfera e a decodificação será correta, presumindo que a palavra código mais próxima da palavra recebida foi a transmitida.

Nesse contexto, a capacidade de correção de erro t de um código C está relacionado ao tamanho do raio (no caso, o maior raio ℓ) da esfera de *Hamming* $S_t(v)$ em torno de todas as palavras código $v \in C$, tal que para todos os pares diferentes $v, u \in C$ as esferas de *Hamming* são disjuntas, isto é,

$$t = \underset{v,u \in C}{max}\{\ell | S_\ell(v) \cap S_\ell(u) = 0,\ v \neq u\}. \tag{D.8}$$

No entanto, como a correção de t erros só é garantida se $d_{min} \geq 2t+1$, então um código de bloco linear tem a capacidade de detectar até $d_{min} - 1$ erros e corrigir até $t \leq \left\lfloor \frac{d_{min}-1}{2} \right\rfloor$ erros.

D.3 Descrição Matricial

Considerando k a dimensão do código $C(n, k)$, então qualquer conjunto de k vetores linearmente independentes podem formar uma base para o referido código. Esses k vetores permitem expressar qualquer outro vetor do código como combinação linear dos seus elementos e, portanto, podem ser usados como linhas para formar uma matriz $G_{k \times n}$, chamada de matriz geradora do código. Assim, qualquer palavra código $v = (v_1, v_2, ..., v_n)$ pode ser representada por uma combinação linear das linhas de G. Em geral, o processo de codificação de uma palavra de informação i em uma palavra código v pode ser representada por:

$$v = iG, \tag{D.9}$$

em que i é uma *k*-tupla dos símbolos de informação a serem codificados.

Associado a cada código linear $C(n, k)$, existe um complemento ortogonal $C^\perp$ constituído por todos os vetores ortogonais a C. Esse complemento, por ser um subespaço ortogonal, chamado de código dual de C, é também um código de bloco linear.

O código dual $C^\perp$ tem dimensão $n-k$ e possui uma matriz geradora $H_{(n-k)\times n}$, chamada de matriz de teste de paridade. Cada linha dessa matriz será ortogonal a qualquer palavra código de C, o que permite escrever a seguinte relação:

$$vH^T = 0. \tag{D.10}$$

Essa relação define um teste para verificar se uma sequência de símbolos é palavra código. Considerando o recebimento de um vetor $r = e + v$ após sua transmissão, em que e é um padrão de erro causado por ruído no canal. Se rH^T for diferente de 0, é porque r não é palavra código e portanto ocorreu erro na transmissão. O resultado de rH^T é chamado de síndrome de r. Se for detectada a presença de erros, o decodificador tomará os procedimentos para

localiza-los e corrigi-los ou fará requisição para retransmissão de v. Entretanto, pode ocorrer que os erros em um vetor sejam não detectáveis, portanto existe erro, mas $vH^T = 0$. Isto ocorre quando e é idêntico a uma palavra-código.

Chama a atenção o fato de que para cada escolha de uma base para C obtém-se uma matriz geradora G diferente, de modo que esta matriz não é única. Portanto, também existe mais de uma matriz de verificação de paridade H.

Além disso, como consequência da definição de código dual, tem-se que a matriz geradora do código C é a matriz de verificação de paridade do código dual $C^{\perp}$. Portanto, o código C contém uma palavra código não nula com peso de *Hamming* w ou menor se, e somente se, existe um conjunto linearmente dependente de w colunas de H. Como consequência deste teorema, tem-se que um código C tem peso mínimo não menor do que w se, e somente se, todo conjunto de $w-1$ colunas de H é linearmente independente.

Dessa forma, para determinar um código $C(n,k)$ que corrija t erros, a matriz $H_{(n-k)\times n}$ deverá ter conjuntos de $2t$ colunas que sejam linearmente independentes.

D.3.1 Códigos sistemáticos e limite de *Singleton*

Um código é definido como sistemático se as k componentes do código são os símbolos de informação em sua ordem natural. Os símbolos restantes são chamados símbolos de redundância.

Toda matriz geradora é equivalente a uma matriz geradora na forma sistemática $G = [I : P]$, na forma escalonada das linhas, em que I é a matriz identidade $k \times k$ e P é uma matriz $k \times (n-k)$, obtida por meio de permutação de colunas e operações elementares com as linhas. Uma definição apropriada da matriz verificação de paridade sistemática é $H = [-P^T : I]$, isto porque $GH^T = 0$. Assim, pode-se afirmar que todo código de bloco linear é equivalente a um código sistemático.

Por meio da forma sistemática de um código, é possível obter um limitante da distância mínima deste código, chamado de limitante de *Singleton*, em função de seus parâmetros n e k, que satisfaça a seguinte desigualdade:

$$d_{min} \leq n - k + 1. \tag{D.11}$$

O limitante de *Singleton* estabelece que para corrigir t erros, um código deve ter $2t = n-k$ símbolos de paridade, ou seja, 2 símbolos de paridade para cada erro a ser corrigido. A maioria dos códigos, inclusive os definidos como ótimos, tem muito mais símbolos de paridade do que necessário pelo limitante de *Singleton*, mas alguns satisfazem este limitante com igualdade. Diz-se que um código é separável em máxima distância, MDS, se $d_{min} = n - k + 1$.

D.4 Códigos Modificados

Dependendo da especificidade da aplicação, existem restrições não relacionadas com o controle de erro, que levam a uma modificação no comprimento natural de um código, por meio de seus três parâmetros fundamentais: n, k e $r = n - k$. Existem seis tipos básicos de técnicas de modificações que podem ser realizadas nos parâmetros de um código [24, 37, 133], a saber:

- **Aumentar**: Fixa n; incrementa k; decrementa r.
- **Expurgar**: Fixa n; decrementa k; incrementa r.
- **Estender**: Fixa k; incrementa n; incrementa r.
- **Puncionar (ou Perfurar)**: Fixa k; decrementa n; decrementa r.
- **Prolongar**: Fixa r; incrementa n; incrementa k.
- **Encurtar**: Fixa r; decrementa n; decrementa k.

Quando um código C é aumentado, mais palavras código são adicionadas, podendo resultar em uma diminuição da distância mínima de C. Ao contrário, o processo de expurgar um código implica em retirar palavras código, podendo ocasionar um aumento na distância mínima do código. Um código linear pode ser facilmente aumentado pela adição de linhas à matriz geradora e pode ser expurgado, por meio do processo inverso de retirada de linhas.

Estender um código significa adicionar símbolos redundância extra a ele, podendo aumentar a distância mínima do código. Nesse tipo de modificação, em termos de espaço vetorial, o número de vetores que constituem a matriz geradora mantem-se e portanto a dimensão do código não se altera, porém a taxa do código é reduzida. O inverso é a punção, em que elimina-se símbolos redundância, podendo ocasionar a diminuição da distância mínima do código. Para estender um código linear, adiciona-se colunas à sua matriz geradora e para puncionar o código, o processo é inverso, ou seja, exclui-se colunas.

Ao prolongar um código C, aumenta-se o comprimento e adiciona-se palavras código a C. O processo inverso de encurtar um código envolve a retirada de palavras código e exclusão de coordenadas. Como nos códigos lineares encurtados o número de símbolos de redundância não se altera, então a capacidade de controle de erros do código será sempre igual ou superior à do código original.

D.5 Decodificação por Síndrome

Em um código linear (n, k), o processo de decodificação envolve uma decisão sobre qual palavra código foi transmitida. Isto pode ser feito particionando os q^k vetores possíveis na recepção em q^k subconjuntos disjuntos de maneira que cada subconjunto contenha somente uma palavra código. Esta distribuição é chamada de arranjo padrão.

D.5.1 Arranjo padrão e classe lateral

O arranjo padrão é uma forma de representar todas as esferas de *Hamming* centradas na palavras código. Por exemplo, a esfera de raio t tendo a palavra código nula como centro é dada por $S_0 = \{x|d(0,x) \leq t\}$. Esta esfera contém todas as palavras recebidas, x, que serão decodificadas como a palavra código toda nula. Da mesma forma, a esfera $S_v = \{x|d(v,x) \leq t\}$ contém todas as palavras que recebidas que serão decodificadas como a palavra código v.

Conforme ilustrado na Tabela D.1, cada coluna do arranjo padrão corresponde a uma esfera de *Hamming* com centro em uma palavra código. As palavras código $0, v_2, ..., v_{q^k}$ do código C são colocadas em uma linha com a palavra código nula. Cada linha do arranjo padrão corresponde a uma classe lateral. A classe lateral é um subgrupo de $GF(q)^n$ com elementos representados por:

$$r = v + e, \tag{D.12}$$

em que e representa uma *n*-tupla de $GF(q)^n$.

Existem $i = q^{n-k}$ classes laterais diferentes associadas a um código de bloco $C(n,k)$ corretor de erro linear. Sendo uma delas, geralmente adotada como primeira linha, o conjunto das palavras do código corretor de erro original. Portanto, existem $q^{n-k} - 1$ classes laterais diferentes da classe lateral que contém o vetor nulo.

Tabela D.1 Arranjo padrão de um código de bloco linear

S_0	S_{v_2}	S_{v_3}	...	$S_{v_{q^k}}$
0	v_2	v_3	...	v_{q^k}
$0+e_1$	v_2+e_1	v_3+e_1	...	$v_{q^k}+e_1$
$0+e_2$	v_2+e_2	v_3+e_2	...	$v_{q^k}+e_2$
...	...	...	...	...
$0+e_i$	v_2+e_i	v_3+e_i	...	$v_{q^k}+e_i$

Das $q^{n-k} - 1$ *n*-tuplas restantes, uma *n*-tupla e_1 não usada anteriormente é escolhida. Então se forma uma segunda linha pela adição de e_1 a cada palavra código e o resultado de cada soma é colocado na coluna da palavra código a qual se somou e_1. A terceira linha é formada pela soma de e_2, que não pertence às linhas anteriores, a cada palavra código e o resultado colocado na coluna da palavra código respectiva. Assim, as outras linhas vão sendo preenchidas até que as $q^{n-k} - 1$ *n*-uplas tiverem sido utilizadas. Como o código é um subgrupo, esse procedimento gera as classes laterais do grupo.

O vetor de peso mínimo em uma classe lateral é chamado de elemento líder dessa classe e está representado na primeira coluna da Tabela D.1. Dessa forma, o vetor nulo é o líder de classe original do código C. Em geral, todas as classes laterais de peso menor ou igual a $\left\lfloor \frac{d-1}{2} \right\rfloor$ tem um líder de classe único. Portanto, cada um desses elementos é líder de uma e somente uma classe.

O arranjo padrão tem um valor somente conceitual pois, para valores grandes de n e k, seria impraticável a construção deste arranjo.

D.5.2 Exemplo de decodificação por síndrome

Todos os elementos de uma classe lateral têm a mesma síndrome, a qual é única para esta classe lateral e os elementos de classes laterais distintas possuem síndromes distintas. Assim, a decodificação de um vetor recebido r, dada uma matriz de paridade H, consiste em três passos:

- Cálculo da síndrome: rH^T;
- Localização do líder de classe lateral e_i cuja síndrome é igual a rH^T. Portanto e_i é considerado o padrão de erro resultante do canal.
- O vetor r é decodificado com estimativa na palavra código $\hat{v} = r - e_i$.

Como quaisquer duas palavras na mesma classe lateral apresentam a mesma síndrome, então precisa-se somente conhecer as síndromes e os líderes das classes laterais. Portanto, supondo-se um código binário $C(6,3)$ cuja matriz geradora é dada por:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{D.13}$$

O arranjo padrão correspondente deste código é ilustrado na Tabela D.2:

Tabela D.2 Exemplo de arranjo padrão de um código $C(6,3)$

000000	100110	010101	001011	011110	101101	110011	111000
000001	100111	010100	001010	011111	101100	110010	111001
000010	100100	010111	001001	011100	101111	110001	111010
000100	100010	010001	001111	011010	101001	110111	111100
001000	101110	011101	000011	010110	100101	111011	110000
010000	110110	000101	011011	001110	111101	100011	101000
100000	000110	110101	101011	111110	001101	010011	011000

Para o referido código $C(6,3)$, pode-se obter uma matriz de verificação de paridade H dada por:

$$H = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{D.14}$$

Calculando as síndromes para qualquer um dos elementos de cada linha da Tabela D.2, tem-se o resultado ilustrado na Tabela D.3 em que estão relacionados os vetores de peso menor ou igual a 1 (C é um código que corrige um erro), com as suas respectivas síndromes.

Tabela D.3 Relação entre os líderes de classe e sua síndrome

Líder	Síndrome
000000	000
000001	001
000010	010
000100	100
001000	011
010000	101
100000	110

Então, supondo que foi recebida a palavra $r = (010111)$, o resultado do cálculo $rH^T = (010)$. Seguindo o procedimento de decodificação por síndromes, tem-se que $e = (000010)$ e a palavra transmitida estimada é $\hat{v} = (010101)$.

D.6 Códigos Cíclicos

Existem diversos códigos de bloco lineares conhecidos, dentre eles: o código de *Hamming*, BCH e Reed-Solomon. A seguir, os códigos cíclicos, um caso especial de códigos lineares, serão descritos, e na seção seguinte serão descritos os códigos BCH que pertencem à classe dos códigos cíclicos e que serão utilizados na implementação dos algoritmmos desta tese.

Os códigos cíclicos (n, k) em $GF(q)$ são uma subclasses dos códigos de bloco lineares, podendo ser representados por polinômios e a sua grande popularidade resulta da sua facilidade de implementação por meio de registradores de deslocamento e lógica combinacional [24] [37] [166].

Um código de bloco linear C é cíclico se e somente se cada deslocamento cíclico de uma palavra código v também é uma palavra código.

$$v = (v_0, v_1, ..., v_{n-1}) \in C \Leftrightarrow v^{(1)} = (v_{n-1}, v_0, ..., v_{n-2}) \in C, \tag{D.15}$$

em que a notação $v^{(1)}$ representa o deslocamento cíclico de 1 posição do vetor. Generalizando, tem-se $v^{(\ell)}$, representando ℓ deslocamentos cíclicos.

Para toda palavra código v, um polinômio $v(x)$ pode ser associado. Cada *n*-tupla corresponde a um polinômio de grau menor ou igual a $n - 1$.

$$v = (v_0, v_1, ..., v_{n-1}) \rightarrow v(x) = v_0 + v_1 x + ... + v_{n-1} x^{n-1}. \tag{D.16}$$

O conjunto de palavras código (polinômios) é definido com uma estrutura algébrica denominada ideal contida no anel $GF(q)[x]/(x^n - 1)$. Portanto, um código cíclico é um conjunto de todos os múltiplos de um polinômio chamado de *polinômio gerador* do código $g(x) \in GF(q)[x]/(x^n - 1)$, ou seja, o código cíclico é o conjunto de todos os produtos $u(x)g(x)$ em que $u(x)$ são polinômios de grau menor ou igual a $k - 1$.

O polinômio gerador $g(x)$ é um divisor de $(x^n - 1)$. Assim, existe um código cíclico de tamanho n com polinômio gerador $g(x)$ com grau $n - k$ se e somente se $g(x)|(x^n - 1)$.

Para encontrar o gerador polinomial, o polinômio $(x^n - 1)$ deve ser decomposto em fatores irredutíveis (polinômios primos) $\phi_j(x), j = 1, 2, ..., l$, tal que

$$(x^n - 1) = \phi_1(x)\phi_2(x)...\phi_l(x). \tag{D.17}$$

Qualquer uma das combinações de produto dos polinômios primos $\phi_j(x)$ obtidos pela fatoração podem ser utilizados para obter um polinômio gerador.

Com base no que foi exposto anteriormente, a palavra de informação representada pelo polinômio $u(x)$ com grau menor ou igual a $k - 1$ e $g(x)$ o polinômio gerador com grau igual $n - k$ pode ter regras simples de codificação dada por:

- Codificação não sistemática

$$v(x) = u(x)g(x),\ \deg u(x) \leq k - 1,\ \deg g(x) = n - k. \tag{D.18}$$

- Codificação sistemática

$$v(x) = x^{n-k}u(x) - [(x^{n-k}u(x))\ \bmod g(x)]. \tag{D.19}$$

D.7 Códigos BCH

O código BCH foi primeiramente descoberto de forma independente por A. Hocquenghem [130] em 1959 e por R. C. Bose e D. K. Ray-Chaudhuri em 1960 [131]. Um código $BCH(n, k, d_{min})$ é um subespaço vetorial linear com elementos (palavras código) de comprimento n, com dimensão k e distância de *Hamming* mínima d_{min}. A estrutura imposta pelos espaços vetoriais propiciam meios estruturados para a construção de codificadores e decodificadores.

Para qualquer inteiro $s \geq 3$, com capacidade de correção de $1 \leq t < 2^{s-1}$ erros, existe um código $BCH(n, k, d_{min})$ primitivo binário, estruturado da seguinte forma:

- Comprimento do Código: $n = 2^s - 1$.
- Números de bits de redundância: $n - k \leq st$, em que $k = n - \deg g(x)$.
- Distância mínima: $d_{min} \geq 2t + 1$.

Dado o comprimento do código $n = q^s - 1$ para um inteiro s, com capacidade de correção de t erros e sendo α um elemento primitivo de ordem n de $GF(q^s)$, em que n divide $(x^n - 1)$, o procedimento para definir o polinômio gerador de um código BCH é dado por:

1. Escolher um polinômio primo de grau s para gerar $GF(q^s)$.
2. Encontrar o polinômio mínimo $\phi_j(x)$ de α^j para $j = 1, 2, ...2t$, , ou seja, $\phi_1(x), \phi_2(x), ...,$ $\phi_{2t}(x)$ são os polinômios mínimos, respectivamente, das raízes $\alpha^1, \alpha^2, ..., \alpha^{2t}$ do polinômio $g(x)$.
3. $g(x) = MMC[\phi_1(x), \phi_2(x), ..., \phi_{2t}(x)]$, em que MMC significa mínimo múltiplo comum.

Um código BCH assim construído pode, em alguns casos, corrigir mais que t erros. Por esse motivo, $d_{min} = 2t + 1$ é chamada de distância de projeto do código, pois essa distância mínima pode ser maior. Esse limitante inferior é chamado de limitante do BCH.

D.8 Códigos Reed-Solomon

Os códigos Reed-Solomon, apresentados por Irving Reed e Gus Solomon em 1960 [120], são uma sub-classe dos códigos BCH (*Bose-Chaudhuri-Hocquenghem*) com símbolos em sequências de $s-$bits. Estes estão entre os mais poderosos códigos no que diz respeito à capacidade de correção de erro, sendo largamente aplicados nos dispositivos de armazenamento (fita magnética, CD's, DVD's), RAID (*Redundant Array of Independent Disks*), códigos de barra, comunicação móvel e sem fio, comunicação via satélite, televisão digital e aplicações no contexto espacial.

Estes códigos de tamanho n e dimensão k, também chamados de $RS(n, k)$, são códigos polinomiais cíclicos não binários construídos e decodificados em $GF(q)$, em que $q = 2^s$.

Um código $RS(n, k)$ convencional é estruturado da seguinte forma:

$$(n, k) = (2^s - 1, 2^s - 1 - 2t) \tag{D.20}$$

O espaço de *n*-tuplas é de um código $RS(n, k)$ é constituído por 2^{ns} *n*-tuplas, das quais 2^{ks} são palavras código [167].

Os códigos Reed-Solomon têm certas características ótimas. Uma das mais importantes é que satisfaz a seguinte condição:

$$d_{min} = n - k + 1 \tag{D.21}$$

Códigos deste tipo, que atingem o limitante de *Singleton* com igualdade, são chamados de códigos separáveis por máxima distância (MDS). Ele é capaz de corrigir qualquer combinação de t ou menos erros, em que t pode ser expresso como

$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor \tag{D.22}$$

Para cada erro, um símbolo redundante é usado para localizar o erro, e outro símbolo redundante é usado para encontrar o seu valor correto.

Dessa forma, uma vez que os elementos diferentes de zero em $GF(q)$ podem ser representados como $(q-1)$ potências de um elemento primitivo α, o processo de codificação Reed-Solomon consiste na geração de $2t$ símbolos de paridade, por meio de um polinômio gerador $g(x)$. O grau do polinômio gerador é igual ao número de símbolos de paridade que o código possui, no caso $2t$, o que significa que devem existir exatamente $2t$ potências de α que sejam raízes deste polinômio.

$$g(x) = \prod_{j=1}^{2t}(x - \alpha^j) \tag{D.23}$$

A codificação sistemática dos símbolos de informação $u(x)$ é dada pela equação D.24.

$$c(x) = x^{n-k}u(x) - [(x^{n-k}u(x))\ mod\ g(x)] \tag{D.24}$$

Nesta pesquisa, a decodificação de códigos BCH e códigos Reed-Solomon será realizada usando o algoritmo de *Berlekamp-Massey* [43]. Considera-se como entrada no decodificador as palavras recebidas na forma sistemática.

D.9 Decodificação de Códigos BCH e Reed Solomon

Segundo o critério da distância mínima, decodificar é encontrar a palavra código mais próxima de uma palavra recebida $r(x)$. Seja $v(x) = v_0 + v_1x + ... + v_{n-1}x^{n-1}$ a palavra transmitida e $e(x) = e_0 + e_1x + ... + e_{n-1}x^{n-1}$ um padrão de erro possível de ser inserido pelo meio de transmissão, o polinômio recebido na entrada do decodificador é:

$$r(x) = v(x) + e(x) \in GF(q)[x]. \tag{D.25}$$

Avaliando-se $r(x)$ em $GF(q^s)$ para o conjunto dos zeros de $g(x)$, $\{\gamma_1, ..., \gamma_r\}$ tem-se que:

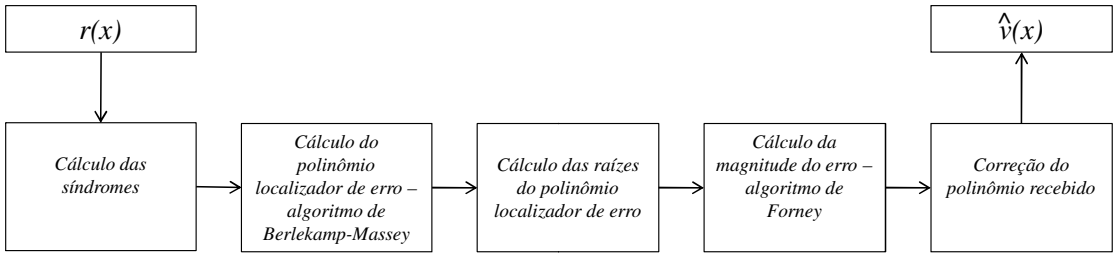
$$r(\gamma_j) = v(\gamma_j) + e(\gamma_j) = e(\gamma_j), \tag{D.26}$$

ou seja,

$$r(\gamma_j) = \sum_{i=1}^{n-1} e_i \gamma_j^i,\ j = 1, 2, ..., r. \tag{D.27}$$

Esse conjunto de r equações pode ser resolvido para e_i e assim o padrão de erro pode ser determinado. O processo de decodificação de um código BCH (binário e não binário) possui diversas etapas, tais como ilustrado na Figura D.2, iniciando por meio do cálculo da síndrome de erros para o vetor recebido $r(x)$.

Figura D.2 Decodificação de um código BCH [133]



A síndrome é o resultado da verificação de paridade executada em $r(x)$ para determinar se $r(x)$ é um membro válido do conjunto de palavras código por meio da equação D.28. O polinômio $r(x)$ será uma palavra código se e somente se $r(\alpha) = 0$, ou seja, se de fato $r(x)$ é um membro, então a síndrome S tem o valor 0, caso contrário, qualquer valor de S diferente de zero indica a presença de erros. Assim, cada componente da síndrome S_j pode ser obtido diretamente pela determinação de $r(x)$ com $x = \alpha_j$, de forma que $S = (S_1, S_2, ...S_{2t})$.

$$S_j \triangleq r(\alpha^j) = \sum_{i=0}^{n-1} r_i \alpha^{ij},\ j = 1, 2, ..., 2t. \tag{D.28}$$

Uma vez que $g(x)$ divide $v(x)$ e $v(\alpha^j) = 0$ para qualquer palavra código, então cada síndrome depende apenas do polinômio de erro, conforme a equação D.29.

$$S_j = v(\alpha^j) + e(\alpha^j) = e(\alpha^j). \tag{D.29}$$

Suponha-se que $e(x)$ tem ν erros, $0 \leq \nu \leq t$, nas localizações $i_1, i_2, ..., i_\nu$, então o polinômio de erro pode ser escrito como

$$e(x) = e_{i_1} x^{i_1} + e_{i_2} x^{i_2} + ... + e_{i_\nu} x^{i_\nu}, \tag{D.30}$$

em que e_{i_l} é o valor do l-ésimo erro.

Para um código BCH binário, o valor do erro $e_i \in \{0, 1\}$ e o conjunto $\alpha^{i_1}, \alpha^{i_2}, \alpha^{i_\nu}$ são as localizações dos erros, sendo que $\alpha \in GF(2^m)$. Portanto, tendo-se o valor do erro $Y_l = e_{i_l}$ e o localizador do erro $X_l = \alpha^{i_l}$, as síndromes S_j são definidas como

$$S_j = \sum_{l=1}^{\nu} e_{i_l}(\alpha^j)^{i_l} = \sum_{l=1}^{\nu} e_{i_l}(\alpha^{i_l})^j = \sum_{l=1}^{\nu} Y_l X_l^j. \tag{D.31}$$

O polinômio localizador de erro, cujas raízes fornecem a localização do erro, é definido como

$$\Lambda(x) = \prod_{l=1}^{\nu}(1 - xX_l) = 1 + \Lambda_1 x + ... + \Lambda_\nu x^\nu. \tag{D.32}$$

em que $\Lambda_1, \Lambda_2, ..., \Lambda_\nu$ são os coeficientes do polinômio localizador do erro.

O polinômio $\Lambda(x)$ é o polinômio de grau mínimo tal que as raízes deste polinômio são valores $X_1^{-1}, X_2^{-1}, ..., X_\nu^{-1}$, isto é, são as inversas das localizações dos erros. Por esta definição, se $x = X_l^{-1}$, então $\Lambda(x) = 0$.

Existe uma relação linear entre as síndromes e os coeficientes do polinômio localizador de erro, conforme a Equação D.33. Esta relação é descrita pela identidade de Newton, que se aplica em qualquer campo.

A Equação (D.33) descreve a saída de um registrador de deslocamento com realimentação linear (LFSR) com coeficientes $\Lambda_1, \Lambda_2, ..., \Lambda_\nu$. Para que essa equação se aplique, os coeficientes Λ_l devem ser obtidos de forma que o LFSR gere a sequência conhecida de síndromes $S_1, S_2, ..., S_{2t}$.

$$S_j = -\sum_{l=1}^{\nu} \Lambda_l S_{j-l},\ j = \nu + 1, \nu + 2, ..., 2t. \tag{D.33}$$

Existem várias maneiras de encontrar o polinômio localizador [24, 37, 133]. Assim, com base nas síndromes, o algoritmo de *Berlekamp-Massey* (BMA) [43, 135], ilustrado na Figura D.3, é um método computacionalmente eficiente para encontrar o polinômio mínimo que produz toda a sequência de síndromes $S_1, S_2, ..., S_{2t}$ por meio de sucessivas iterações, modificando o LFSR atual. O LFSR $\Lambda(x)$ resultante do BMA, corresponde ao polinômio localizador de erro. O grau do polinômio localizador de erros deve ser equivalente ao número de erros ν, e seus coeficientes devem ser condizentes com as síndromes observadas.

A ideia deste algoritmo é que o polinômio $\Lambda(x)$ é calculado com a ajuda das aproximações sequenciais $\Lambda^{(0)}(x), \Lambda^{(1)}(x), ...$ até $\Lambda^{(2t)}(x)$, dada por

$$S_j = -\sum_{l=1}^{L} \Lambda_l^{(j)} S_{j-l}, j = 1, .., 2t, \tag{D.34}$$

em que L é o grau do polinômio $\Lambda^{(j)}(x)$ e $\Lambda_l^{(j)}$ são os coeficientes desse polinômio.

Figura D.3 Algoritmo de Berlekamp-Massey [133]

ENTRADA: $S_1, S_2, \cdots, S_{2t}$
SAÍDA: $\Lambda(x)$
INICIALIZAÇÃO:
$\Lambda(x) \leftarrow 1$: Polinômio localizador de erro
$L \leftarrow 0$: Grau do polinômio $\Lambda(x)$ atual
$T(x), B(x) \leftarrow 1$: Polinômios Temporários
$j \leftarrow 0$: Variável contadora
Δ_j : Discrepância
ITERAÇÃO:
Passo 1. $j \leftarrow j + 1$
Passo 2. $\Delta_j \leftarrow S_j + \sum_{l=1}^{L} \Lambda_l S_{j-l}$
Passo 3. Se $\Delta_j = 0$, então vá para passo 8
Passo 4. $T(x) \leftarrow \Lambda(x) - \Delta_j x B(x)$
Passo 5. Se $2L > j - 1$, então vá para passo 7
Passo 6. Calcule:
$B(x) \leftarrow \Delta_j^{-1} \Lambda(x)$
$\Lambda(x) \leftarrow T(x)$
$L \leftarrow j - L$
Vá para o passo 9
Passo 7. $\Lambda(x) \leftarrow T(x)$
Passo 8. $B(x) \leftarrow xB(x)$
Passo 9. Se $j < 2t$, então vá para passo 1
Passo 10. Se deg $\Lambda(x) = L$, então vá para passo 11
Passo 11. Pare BMA. Vá para o próximo estágio de decodificação.

A busca iterativa de $\Lambda(x)$ inicia-se com um polinômio que pode produzir apenas S_1 e verifica-se se este polinômio é capaz de produzir $S1, S2$ e caso seja capaz de fazê-lo, não é necessário modificá-lo. Se o polinômio que produz S_1 não pode produzir uma sequência mais longa, é necessário determinar os novos coeficientes que produzirão esta sequência. Procedendo-se desta maneira, determina-se um polinômio que seja capaz de produzir a sequência $S1, S2, ..., S_{j-1}$ e busca-se determinar se este polinômio também é capaz de produzir $S1, S2, ..., S_{j-1}$.

A cada estágio, as modificações são feitas no polinômio localizador de erros de maneira que ele tenha o menor grau possível. O processo é repetido até que o polinômio de menor grau que gera todos os componentes da síndrome é encontrado, ou quando o padrão de erro incorrigível é detectado.

Uma vez que o polinômio localizador de erro foi encontrado, as suas raízes são avaliadas a fim de encontrar a localização dos erros. O método *Chien search* [136] pode ser usado para avaliar as raízes. É um procedimento de tentativa e erro para encontrar os zeros de $\Lambda(x)$ em cada elemento diferente de zero $x = X_l$ de $GF(2^m)$. Com base na Equação D.32, se $\Lambda(x) = 0$ então X_l^{-1} é uma raiz.

Agora, tendo encontrado o polinômio localizador de erro e suas raízes, os valores de erro também devem ser determinados para os códigos BCH não binários, sendo desnecessário para um BCH binário, visto que nesse caso, após localizar o erro, basta somente trocar o valor do bit. Em geral, isso é feito usando o algoritmo de *Forney* [137] (equação D.37). Este algoritmo calcula o valor do erro por meio do polinômio avaliador de erro $\Omega(x)$ (Equação D.35) com base no polinômio localizador de erro (Equação D.32).

$$\Omega(x) = S(x)\Lambda(x) \ \ mod\ x^{2t}, \tag{D.35}$$

em que a síndrome polinomial $S(x)$ é definida como

$$S(x) = \sum_{j=1}^{2t} S_j x^{j-1} = \sum_{j=1}^{2t} \sum_{l=1}^{\nu} Y_l X_l^j x^{j-1}. \tag{D.36}$$

Supondo-se que $\Lambda(x)$ tem grau ν, o valor do erro Y_l pode ser calculado como segue:

$$Y_l = -\frac{\Omega(X_l^{-1})}{\Lambda'(X_l^{-1})}, \tag{D.37}$$

em que $\Lambda'(x)$ é a derivada formal de $\Lambda(x)$.

APÊNDICE E

Algoritmo de Decodificação por Lista

Neste apêndice, descreve-se de forma sucinta, o algoritmo de decodificação por lista de Guruswami-Sudan, a partir da descrição de algoritmos baseados em interpolação. Para estudos mais avançados são recomendadas as referências [141–143].

E.1 Algoritmos de decodificação baseados em interpolação

Diferentes algoritmos de descodificação são baseados na interpolação de duas variáveis. Por exemplo os algoritmos de Welch-Berlekamp [139], Sudan [140] e Guruswami-Sudan [141] são baseados nesta abordagem.

Definição E.1 (códigos Reed-Solomon)**.** *Seja $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ com n elementos distintos de $\mathbb{F}_q = GF(q)$. Um código Reed-Solomon de dimensão k e conjunto suporte (α_i) é dado por:*

$$\mathrm{RS}(\alpha, k) = \{(p(\alpha_1), \ldots, p(\alpha_n)) : p \in \mathbb{F}_q[x]_{<k}\}\,.$$

O polinômio interpolante é construído, utilizando-se um polinômio de duas variáveis, $Q(x, y)$, que satisfaz a condição $Q(x_i, y_i) = 0$. Além de simplesmente interpolar em cada ponto, um número inteiro s, denominado multiplicidade de interpolação, é introduzido para definir a ordem de interpolação para cada ponto. Isto equivale a dizer que o valor da função naquele ponto e de suas $s_i - 1$ derivadas. Esta multiplicidade de interpolação irá aumentar a capacidade de correção para todas as taxas de código.

A partir do polinômio de duas variáveis $Q(x, y)$, os polinômios $p(x)$ são obtidos por meio da fatoração, e irão satisfazer a condição $p(x_i) = y_i$ para uma quantidade suficiente de pontos (x_i, y_i). Assim, cada polinômio $p(x)$ representa uma possível palavra transmitida, e o conjunto de polinômios é uma lista de possíveis palavras código decodificadas.

Os algoritmos Welch-Berlekamp [139], Sudan [140] e Guruswami-Sudan [141] são baseados no princípio supracitado.

As diferenças entre os três algoritmos de decodificação são os parâmetros da interpolação de duas variáveis e representam o maior custo na complexidade destes métodos. No que se segue, apresenta-se resumidamente os principais algoritmos de decodificação para códigos Reed-Solomon.

E.1.1 Algoritmo Welch-Berlekamp

O algoritmo Welch-Berlekamp é um algoritmo de decodificação tradicional para códigos Reed-Solomon. Porém, alguns algoritmos de decodificação por lista são baseados nesse método, que consiste no cálculo do polinômio de duas variáveis, por meio da interpolação que satisfaz as seguintes condições:

$$(\mathbb{IP}_{WB}) \triangleq \begin{cases} 0 \neq Q(x,y) \triangleq Q_0(x) + YQ_1(x), \\ Q(\alpha_i, y_i) = 0,\ \forall i \in \{1,\ldots,n\}, \\ \deg Q_0 \leq n-t-1, \\ \deg Q_1 \leq n-t-k, \end{cases}$$

em que $t = \left\lfloor \frac{n-k}{2} \right\rfloor$ é a capacidade de correção do código. O algoritmo Welch-Berlekamp consiste em:

Tabela E.1 Algoritmo de Welch-Berlekamp.

ENTRADA: A palavra recebida $r \in \mathbb{F}_q^n$ e o código Reed-Solomon $\mathcal{C}$
SAÍDA: A palavra código $c \in \mathcal{C}$, se existir, tal que $d(c,y) \leq t = \left\lfloor \frac{n-k}{2} \right\rfloor$, na forma polinomial.
INICIO
$Q(x,y) \leftarrow$ Interpolação($\mathbb{IP}_{WB}, \mathcal{C}$)
Retorna $-\frac{Q_0(x)}{Q_1(x)}$.
FIM

E.1.2 Algoritmo de Sudan

Sudan observou que, para corrigir mais erros, com o algoritmo Welch-Berlekamp, pode acontecer que existam diferentes raízes Y do polinômio de duas variáveis que satisfaçam a condição. Então, Sudan propôs modificar o problema de interpolação como segue:

$$(\mathbb{IP}_S) \triangleq \begin{cases} 0 \neq Q(x,y) \triangleq \sum_{i=0}^{\ell} Q_i(x) y^i, \\ Q(\alpha_i, y_i) = 0,\ \forall i \in \{1,\ldots,n\}, \\ \deg Q_j \leq n-T-1-j(k-1),\ \forall j \in \{0,\ldots,\ell\}. \end{cases}$$

O método de Sudan pode ser resumido da seguinte forma:

Tabela E.2 Algoritmo de Sudan.

ENTRADA: A palavra recebida $r \in \mathbb{F}_q^n$ e o código Reed-Solomon $\mathcal{C}$.
SAÍDA: A lista de palavras código c_i de $\mathcal{C}$, tal que $\forall i,\ d(v, c_i) \leq \tau$.
INICIO
$Q(x, y) \leftarrow$ Interpolação$(\mathbb{IP}_S, \mathcal{C})$
$(p_1, \ldots, p_\ell) \leftarrow$ y-Raízes$(Q(x, y))$
Candidato $\leftarrow \{\}$
Para $i \in \{1, \ldots, \ell\}$
Se $d\left(p_i(\alpha), r\right) \leq \tau$
Candidato $\leftarrow$ *Candidato* $\cup \{p_i(\alpha)\}$
Retorna *Candidato*.
FIM

E.1.3 Algoritmo de Guruswami-Sudan

O algoritmo de Guruswami-Sudan, utilizado neste trabalho de tese, introduz a noção de raízes com multiplicidade no Algoritmo de Sudan.

Definição da derivada de Hasse:

Definição E.2 (Derivada Hasse)**.** *Seja $Q(x, y) \in \mathbb{F}_q[x, y]$ ser um polinômio de duas variáveis e a, b serem dois inteiros positivos. O (a, b)-ésima derivada Hasse de Q é*

$$Q^{[a,b]}(x, y) \triangleq \sum_{i=a}^{\deg_x(Q)} \sum_{j=b}^{\deg_y(Q)} \binom{i}{a}\binom{j}{b} q_{i,j} x^{i-a} y^{j-b}.$$

A definição de multiplicidade maior que um:

Definição E.3 (Raízes com multiplicidade)**.** *Seja $Q(x, y) \in \mathbb{F}_q[x, y]$ um polinômio de duas variáveis e $(\alpha, \beta) \in (\mathbb{F}_q)^2$ ser um ponto. O ponto (α, β) é uma raiz com multiplicidade $s \in \mathbb{N}$ se e somente se s é o maior inteiro tal que para todo $i + j < s$,*

$$Q^{[i,j]}(\alpha, \beta) = 0.$$

Guruswami e Sudan notaram que para dois polinômios p_{i_0}, p_{j_0}, têm-se $r_{k_0} = p_{i_0}(\alpha_{k_0}) = p_{j_0}(\alpha_{k_0})$ e, portanto, o ponto (α_{k_0}, y_{k_0}) é uma raiz de Q com multiplicidade de pelo menos 2. Então Guruswami e Sudan propuseram adicionar a restrição de multiplicidade durante o processo de interpolação.

$$(\mathbb{IP}_{GS}) \triangleq \begin{cases} 0 \neq Q(x, y) \triangleq \sum_{i=0}^{\ell} Q_i(x) Y^i, \\ Q(\alpha_i, y_i) = 0, \text{ com multiplicidade } s,\ \forall i \in \{1, \ldots, n\}, \\ \deg(Q_j) \leq s(n - \tau) - 1 - j(k - 1),\ \forall j \in \{0, \ldots, \ell\}. \end{cases}$$

O algoritmo de Guruswami-Sudan é o seguinte:

Tabela E.3 Algoritmo de Guruswami-Sudan.

ENTRADA: A palavra recebida $r \in \mathbb{F}_q^n$ e o código Reed-Solomon $\mathcal{C}$.
SAÍDA: A lista de palavras código c_i de $\mathcal{C}$, tal que $\forall i,\ d(v, c_i) \leq \tau$.
INICIO
$Q(x, y) \leftarrow$ Interpolação$(\mathbb{IP}_{GS}, \mathcal{C})$
$(p_1, \ldots, p_\ell) \leftarrow$ y-Raízes$(Q(x, y))$
Candidato $\leftarrow \{\}$
Para $i \in \{1, \ldots, \ell\}$
Se $d\left(p_i(\alpha), r\right) \leq \tau$
Candidato $\leftarrow$ *Candidato* $\cup \{p_i(\alpha)\}$
Retorna *Candidato*.
FIM

Uma vez obtido o polinômio $Q(x, y)$ que interpola o conjunto de pontos $(x_i, y_i), i = 1, 2, ..., n$, o próximo passo no algoritmo de Guruswami-Sudan é a determinação de todos os fatores, na forma $y - p(x)$, em que $p(x)$ é um polinômio de grau $\leq v$, em que v é um inteiro não negativo que representa o peso do expoente da incógnita y, ou seja, $(y - p(x))|Q(x, y)$.

Seja $p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_v x^v$ uma raiz de y de $Q(x, y)$, então o algoritmo de Rock-Ruckenstein, descrito a seguir, irá determinar os coeficientes de $p(x)$, um de cada vez.

Tabela E.4 Algoritmo de Fatoração de *Rock-Ruckenstein*.

ENTRADA: $Q(x, y)$ e k.
SAÍDA: A lista de polinômios $p(x)$ de grau $\leq k$.
INICIO
Faça $p(x) = 0, u =$ grau$(p) = -1$
Cria uma lista encadeada para receber os polinômios
Faça $v = 0$
Chama *rothrucktree*$(Q(x, y), u, p)$
FIM

O algoritmo de Guruswami-Sudan utiliza o paradigma de decodificação por lista, portanto sua saída é uma lista de polinômios $L = p_1(x), p_2(x), ..., p_L(x)$. A palavra código transmitida estará em L se o número de erros inseridos pelo canal transmitida for menor ou igual a τ.

Como $Q(x, y)$ é um polinômio de grau ponderado e $p(x)$ são polinômios que satisfazem $(y - p(x))|Q(x, y)$, então o número máximo de polinômios retornados é o grau de y de $Q(x, y)$. Para saber o grau de $Q(x, y)$, basta saber o grau de $Q(1, y)$. Como limitante superior, tem-se que:

Tabela E.5 Algoritmo de Fatoração de *Rock-Ruckenstein* - Função *Rothrucktree*.

ENTRADA: $Q(x, y)$, p(x) e u.
SAÍDA: A lista de polinômios $p(x)$ de grau $\leq k$.
INICIO
$v = v + 1$
Se $Q(x, 0)$ ENTÃO
  Adiciona (x) na lista de saída.
Senão
  Se $u < k$ ENTÃO
    $R =$ lista de raízes de $Q(0, y)$.
    Para cada raiz α em R FAÇA
      $Q_{new}(x, y) = Q(x, xy + \alpha)$
      $p_{u+1} = \alpha$
      Chama rothrucktree$(\langle\langle Q_{new}(x, y)\rangle\rangle, u + 1, p)$
    FimPara
  Senão
    Não gera saída
  FimSe
FimSe
FIM

$$Lm < (m + \frac{1}{2})\sqrt{\frac{n}{v}}, \tag{E.1}$$

APÊNDICE F

Conceitos Básicos sobre Algoritmo Genético

Esta apêndice tem como objetivo apresentar os Algoritmos Genéticos como técnica de busca e otimização. Os tópicos tratados abordam os conceitos básicos para entender o funcionamento básico de um AG.

F.1 Definição

Algoritmos genéticos (AG) são métodos probabilísticos de busca e otimização, inspirados no princípio Darwiniano de seleção natural das espécies (princípio de sobrevivência dos mais aptos) e reprodução genética, para fazer evoluir um conjunto de soluções para um determinado problema [169].

Originado de estudos sobre autômatos celulares por Holland [170], os AG empregam uma estratégia de busca paralela e adaptativa, embora aleatória, direcionada à busca de soluções (sujeita a restrições). Apesar de aleatórios, os AG não realizam buscas aleatórias não-direcionadas, pois exploram informações históricas com o objetivo de encontrar novos pontos de busca que propiciem soluções com melhores desempenhos [169].

Há diversos problemas que envolvem encontrar a melhor (ou a mais satisfatória) solução em um conjunto de todas as possíveis soluções, aqui chamado de espaço de busca. Neste contexto, é que surgem os algoritmos genéticos, como uma maneira de solucionar problemas de busca adaptativa, nos quais o conhecimento para controlar a busca é obtido dinamicamente. Na maioria dos problemas de busca ocorre também uma otimização.

F.2 Terminologia Básica

Uma analogia entre a terminologia necessária para o estudo de Algoritmos Genéticos, adaptada de [171], e o sistema natural é representada por meio da Tabela F.1. Os principais termos encontram-se sintetizados na Figura F.1.

Tabela F.1 Analogia entre Algoritmos Genéticos e o Sistema Natural

Natureza	**Algoritmos Genéticos**
Cromossomo	Palavra, *string*, vetor
Gene	Característica do problema
Alelo	Valor da característica
Locus	Posição na palavra, *string*, vetor
Genótipo	Estrutura da palavra, *string*, vetor
Fenótipo	Estrutura real do problema
Indivíduo	Solução
Geração	Ciclo

Uma possível solução para um problema é representada como sendo um indivíduo. Em geral, o indivíduo não é representado diretamente pelos valores numéricos verdadeiros do problema a ser otimizado. A este conjunto de valores do mundo real, o qual representa a informação numérica do problema, é chamado de fenótipo.

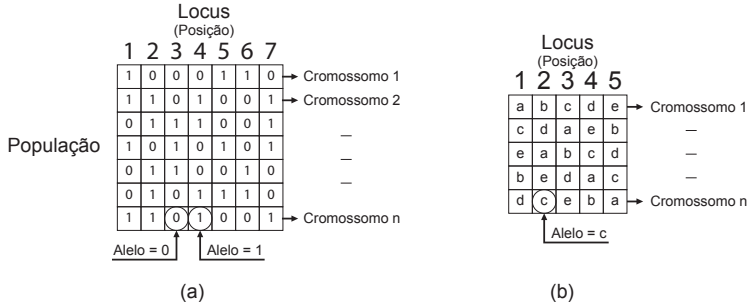
A representação do fenótipo de um indivíduo é feita por meio de uma palavra (vetor ou *string*) de símbolos, chamada de cromossomo (representação cromossômica), em que cada símbolo (ou conjunto de símbolos) é comumente chamado de gene.

Um possível alfabeto que pode ser usado, no caso mais simples, para codificar um indivíduo é o alfabeto binário $A = \{0, 1\}$. Mas, no caso geral, o alfabeto genético vai depender de cada problema. Como cada gene pode assumir qualquer valor do alfabeto A, então cada elemento é equivalente a um alelo, ou seja, um valor possível para um dado gene.

A posição de um gene em um cromossomo corresponde a um *locus* gênico (ver Figura F.1). O genótipo corresponde à estrutura que expressa um valor específico de uma palavra. Por exemplo, um indivíduo codificado por meio de genótipos pode ser uma palavra de 7 bits (ver Figura F.1 (a)), sendo cada bit representando um atributo utilizado para identificar um tipo de ataque em uma rede de computadores.

Com o uso dos processos de seleção, reprodução e substituição, análogos aos que ocorrem com os seres vivos, o AG leva a indivíduos gradualmente mais adaptados ao seu ambiente com o passar das gerações.

Figura F.1 Terminologia usada em Algoritmo Genético: (a) cromossomo com 7 genes e alelos correspondendo a valores binários; (b) cromossomo com 5 genes e alelos correspondendo a valores discretos.



F.3 Um Algoritmo Genético Simples

Um funcionamento básico de um AG, segundo uma interpretação feita a partir de [169] e apresentado na Figura F.2, pode ser o que se segue:

Passo 1: um conjunto de indivíduos é gerado aleatoriamente (ou a partir de algum critério inicial), formando o que se chama de população. Normalmente, esta população tem um tamanho fixo.

Passo 2: uma avaliação inicial de cada indivíduo da população é feita de acordo com uma função objetivo. Quanto melhor o desempenho na função objetivo, maiores serão as notas dos indivíduos em tal avaliação. Dá-se o nome de aptidão (do inglês *fitness*) a esta medida e este cálculo normalmente é feito em função do cromossomo de cada indivíduo.

Passo 3: é aplicado sobre a população um mecanismo de seleção, visando escolher um individuo (reprodução assexuada) ou um par de indivíduos (reprodução sexuada) para gerar descendência.

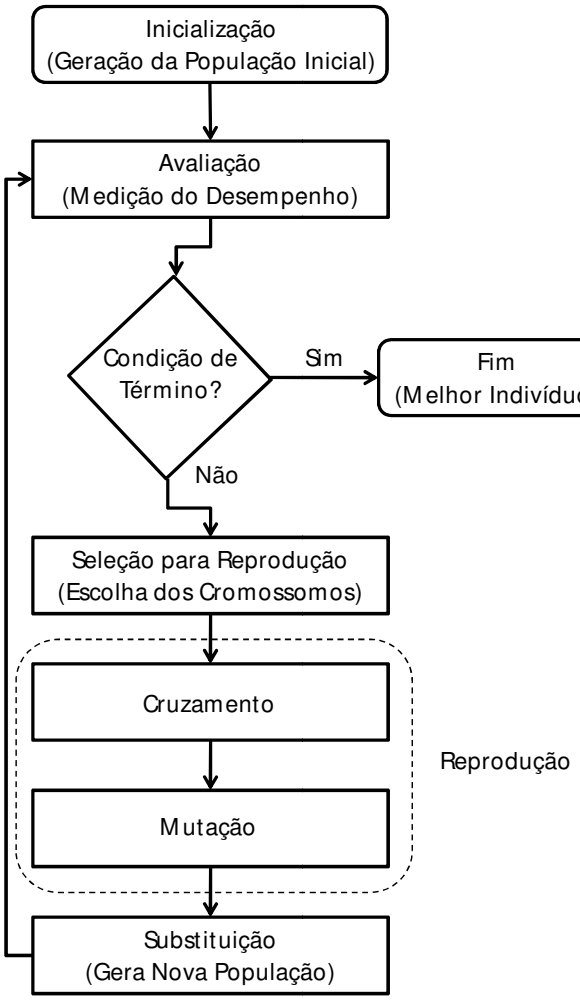
Passo 4: é aplicado, com uma probabilidade p_{cross}, um cruzamento (do inglês *crossover*) sobre uma dupla de indivíduos (ou pais) selecionados no passo 3, ou seja, tal dupla poderá ou não realizar o cruzamento, que consiste na geração de dois indivíduos, aqui chamados de descendentes (ou filhos), com base na troca de material genético entre os indivíduos pais.

Passo 5: os descendentes são submetidos ao processo de mutação, com uma probabilidade *pmutation*. A mutação consiste basicamente em ser aplicada uma mudança aleatória de um ou mais *locus* do cromossomo.

Passo 6: concluídos os passos 3, 4 e 5, os descendentes gerados são destinados à nova população. Os indivíduos desta população irão passar neste estágio por uma avaliação.

Chama-se de geração (ou iteração) ao ciclo compreendido entre os passos 3 a 5. Concluída uma geração, é feito um teste se a condição de término foi atendida. A condição de término pode ser um valor buscado de aptidão, um determinado número máximo de iterações que se deseja que o algoritmo seja executado, um intervalo de tempo ou a convergência, esta última considerada quando há um certo número de gerações sem melhora na solução.

Figura F.2 Fluxograma básico de um Algoritmo Genético.



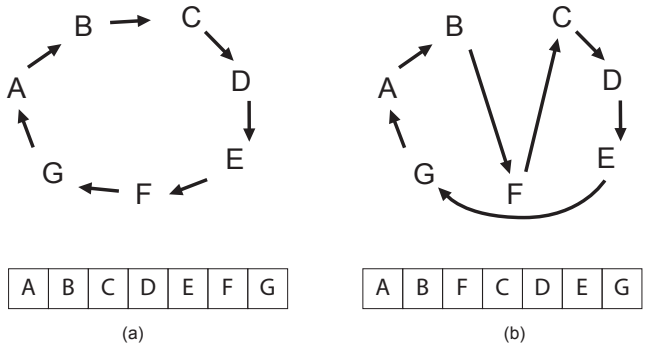
Passo 7: se a condição de término não foi atendida, há um retorno à etapa que corresponde ao passo 3 e um novo ciclo de geração é executado.

Passo 8: se a condição de término foi atendida, há a passagem para o próximo bloco, que é a determinação do indivíduo pertencente à última geração tenha o maior aptidão.

Há variações do modelo de AG ora apresentado. Por exemplo, há problemas de AG que não enfocam o valor do gene e sim seu possível locus dentro do cromossomo, isto é, os genes permanecem com seus alelos fixos. Tais problemas são chamados de problemas de permutação, ou de ordenação. Um exemplo de problema deste tipo é o do caixeiro viajante [132], em que o cromossomo representa a ordem das cidades percorridas (ver Figura F.3). Um problema de ordenação também é objeto de estudo deste trabalho de tese.

A seguir serão vistos com mais detalhes os operadores genéticos utilizados pelos AG.

Figura F.3 Representação cromossômica do problema do caixeiro viajante



F.4 Operadores Genéticos

Com base na Figura F.2, um AG simples é composto por três operadores principais:

- Seleção;
- Cruzamento;
- Mutação.

F.4.1 Seleção

O processo de seleção em AG seleciona indivíduos para a reprodução. O AG trabalha com um número fixo de indivíduos na população ao longo das gerações. Então, a cada nova geração, deve-se selecionar quais indivíduos terão cópias e quais desaparecerão. O mecanismo de seleção em AG emula os processos de reprodução assexuada e seleção natural, ou seja, a seleção é baseada na aptidão dos indivíduos: indivíduos mais aptos têm maior probabilidade de serem escolhidos para reprodução.

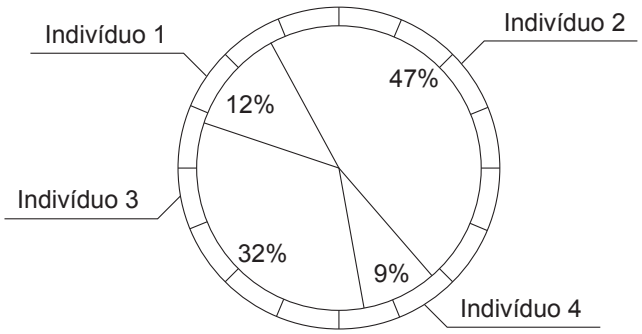
No GA proposto por [169], o esquema de seleção, denominado **seleção por roleta**, é do tipo roleta de cassinos, como pode ser visualizado na Figura F.4. No esquema de seleção por roleta, se $f_\imath$ é a avaliação do indivíduo $\imath$ na população corrente, a probabilidade $p_\imath$ do indivíduo $\imath$ ser selecionado e passar para a próxima geração é proporcional a:

$$p_\imath = \frac{f_\imath}{\sum_{j=1}^{NI} f_j}, \tag{F.1}$$

em que NI é o número de indivíduos da população.

Os indivíduos com elevada aptidão receberão um intervalo maior na roleta, enquanto aqueles que têm mais baixa aptidão receberão menor intervalo na roleta. Para determinar os descendentes dos indivíduos da população, roda-se a roleta polarizada, de acordo com o tamanho da população. Estes descendentes são cópias exatas dos indivíduos sorteados. Com base no

Figura F.4 Exemplo de uma roleta polarizada, com as probabilidades de seleção para 4 indivíduos.



exemplo ilustrado na Figura F.4, girando a roleta quatro vezes, poder-se-ia ter a seguinte escolha:

- 1º giro - indivíduo 2;
- 2º giro - indivíduo 3;
- 3º giro - indivíduo 2;
- 4º giro - indivíduo 1.

Com base nos resultados obtidos, usando a roleta, o indivíduo 2 teria duas cópias, os indivíduos 3 e 1 uma cópia enquanto que o indivíduo 4 desapareceria.

A seleção de indivíduos por roleta pode fazer com que o melhor indivíduo da população seja perdido. Uma alternativa é forçar a escolha do melhor indivíduo encontrado em todas as gerações do algoritmo. Outra opção é utilizar **seleção elitista** que consiste em simplesmente manter sempre o melhor indivíduo da geração atual na geração seguinte [172].

Outro exemplo de mecanismo de seleção é a **seleção baseada em *rank*** [172]. Esta estratégia utiliza as posições dos indivíduos quando ordenados de acordo com a aptidão para determinar a probabilidade de seleção. Podem ser usados mapeamentos lineares ou não lineares para determinar a probabilidade de seleção. O número de cópias dos indivíduos é, em geral, proporcional ao valor de *ranking*. Uma forma de implementação da seleção por *rank* é simplesmente passar os melhores indivíduos para a próxima geração [173].

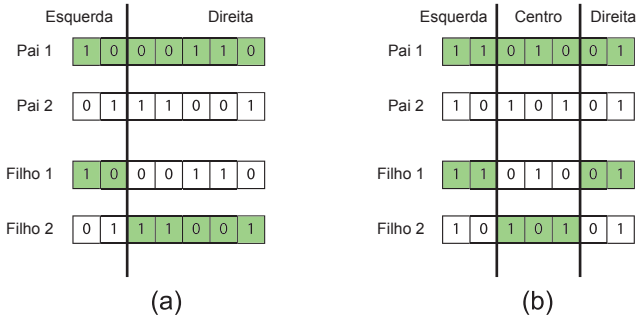
Existe ainda a **seleção por torneio**, pela qual um subconjunto da população com indivíduos é escolhido aleatoriamente e os melhores indivíduos desse grupo são selecionados para processamento genético futuro. Em geral, utiliza-se torneio de 2, isto é, dois indivíduos competem entre si e o ganhador (o de melhor aptidão) torna-se um dos pais. Uma importante propriedade da seleção por torneio é que esta não depende de um conhecimento global da população. Além disso, a seleção por torneio não leva em consideração o *rank* que o indivíduo ocupa na população, permitindo uma seleção com menos tendências [174].

F.4.2 Cruzamento

O operador **cruzamento**, ou recombinação, é utilizado após o de seleção. Esta etapa é marcada pela troca ou combinação de material genético por dois ou mais indivíduos selecionados. É o principal operador genético, pois é responsável pela busca local no espaço de busca. O operador de cruzamento, em geral, é aplicado com probabilidade dada pela taxa de cruzamento p_c, que deve ser maior que a taxa de mutação.

Um tipo bastante comum de cruzamento é a de **um ponto** [169]. Nessa operação, seleciona-se aleatoriamente um ponto de corte nos cromossomos. É a partir desse ponto que se realiza a troca de material cromossômico entre os dois indivíduos, como é ilustrado na Figura F.5 (a), dividindo este em uma partição à direita e outra à esquerda do corte. Cada descendente é composto pela junção da partição à esquerda (direita) de um pai com a partição à direita (esquerda) do outro pai.

Figura F.5 Tipos de cruzamento: (a) cruzamento de um ponto (b) cruzamento de dois pontos



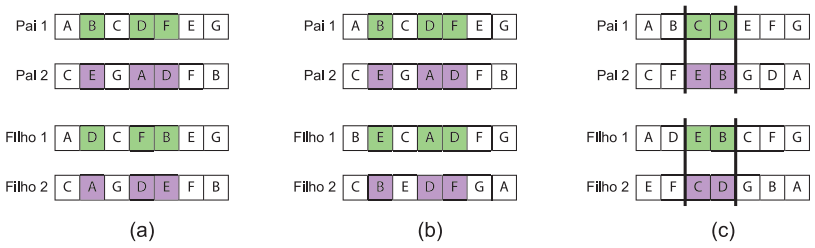
Outro cruzamento, o **multipontos** [169], divide os cromossomos em partições, as quais são recombinadas. A Figura F.5 (b) ilustra um exemplo com 2 pontos de corte. Neste caso, o Filho 1 (Filho 2) recebe a partição central do Pai 2 (Pai 1) e as partições à esquerda e à direita dos corte do Pai 1 (Pai 2).

Além dos dois métodos de cruzamento citados, existem outros tipos de cruzamento que podem ser consultados em [169, 172].

Conforme já mencionado, uma classe importante de problemas do mundo real consiste em decidir a ordem na qual uma sequência de eventos deve ocorrer [174]. Uma forma bastante natural de representar tais problemas é utilizando permutações e, nesses casos, deve-se evitar que o cruzamento introduza repetição de gene em um mesmo indivíduo. Essa restrição inviabiliza os procedimentos de cruzamentos descritos anteriormente. Uma consequência imediata é a necessidade de encontrar novos operadores. Como exemplos de operadores que são aplicados aos problemas de ordenação, aqui chamados de operadores de permutação, têm-se [169, 172], conforme ilustrado na Figura F.6:

- OBX (*Order Based Crossover*): Os genes são selecionadas aleatoriamente. É imposta uma ordem nos genes selecionados do Pai 1 igual a ordem dos respectivos genes em Pai 2;
- PBX (*Position-Based Crossover*): Os genes são selecionadas aleatoriamente e o alelo dos genes selecionados no Pai 2 é imposto ao Pai 1;
- PMX (*Partially Mapped Crossover*): Dados dois cromossomos Pai 1 e Pai 2, realizam-se dois pontos de corte aleatórios sobre os mesmos. As subcadeias de genes encontradas entre estes cortes serão herdadas integralmente pelos indivíduos Filho 1 e Filho 2. Estas subcadeias também determinam um mapeamento de relacionamento entre os genes dos pais afim de tratar a inviabilidade (cromossomos inválidos) ocasionada pelo processo [172].

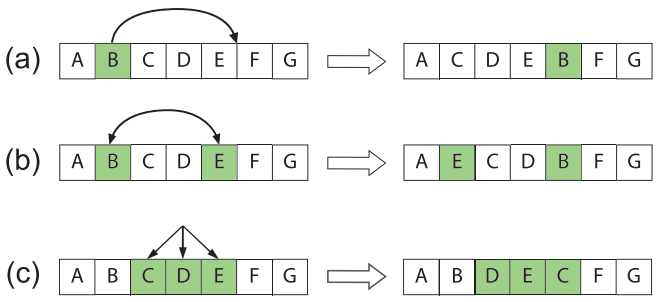
Figura F.6 Operadores de permutação: (a) OBX (b) PBX (c) PMX



F.4.3 Mutação

O operador de mutação modifica aleatoriamente um ou mais genes de um cromossomo. A probabilidade de ocorrência de mutação em um gene é denominada taxa de mutação p_m. Usualmente, são atribuídos valores pequenos para a taxa de mutação, uma vez que esse operador pode gerar um indivíduo potencialmente pior que o original.

Figura F.7 Operadores de mutação: (a) *Position Based Mutation* (b) *Order Based Mutation* (c) *Scramble Mutation*



Considerando a codificação binária, o operador de mutação padrão simplesmente troca o valor de um gene em um cromossomo [169]. Assim, se o alelo de um gene selecionado é 1, o seu valor passará a ser 0 após a aplicação da mutação.

No caso de problemas de permutação, têm-se os seguintes operadores de mutação, ilustrado na Figura F.7:

- *Position Based Mutation*: retira o alelo da posição i e insere na posição j (ver Figura F.7 (a));
- *Order Based Mutation*: troca o alelo da posição i com o alelo na posição j (ver Figura F.7 (b));
- *Scramble Mutation*: uma sublista, aleatoriamente selecionada, é embalharada (ver Figura F.7 (c)).

APÊNDICE G

Arquitetura TCP/IP

Existem diversos tipos de ataques de rede envolvendo diferentes protocolos de comunicação. Contudo, a popularização da Internet fez surgir um novo padrão de transferência de dados, conhecido como a arquitetura TCP/IP e, portanto, torna o estudo dos incidentes de segurança que utilizam esta arquitetura de extrema importância. Neste contexto, a comunicação entre as máquinas utiliza troca de pacotes através da rede. Logo, as técnicas de ataque também usam, de alguma maneira, essa unidade de intercâmbio de informações. Neste apêndice é feita uma descrição resumida da arquitetura TCP/IP, com base em [39].

A arquitetura TCP/IP é baseada em quatro camadas, que interagem entre si para fornecer as funcionalidades do modelo de comunicação como um todo. De fato, a obrigação maior de cada um destes níveis é oferecer determinados serviços para as camadas superiores, escondendo os detalhes de implementação. As camadas são descritas do nível mais baixo até o maior grau de abstração: enlace de dados, rede, transporte e aplicação [39].

G.1 Camadas da arquitetura TCP/IP

A camada de enlace de dados não é propriamente especificada no modelo de referência TCP/IP. Assim, sua implementação vai depender de vários aspectos técnicos relacionados ao tipo da rede empregada. Entretanto, pode-se dizer que ela é responsável pelo fornecimento de uma interface de serviço à camada de rede sendo implementada através de protocolos de conexão.

A segunda camada, a de rede, tem como função básica garantir que um conjunto de informações (encapsuladas em uma estrutura denominada pacote) seja enviado através da sub-rede de comunicações, da melhor forma possível, em um processo conhecido como roteamento. O nome oficial do protocolo para o envio de dados desta camada é IP (*Internet Protocol*). Existem ainda outros protocolos de controle nesta camada, dentre os quais o ICMP (*Internet Control Message Protocol*).

O terceiro nível, o de transporte, tem por finalidade habilitar a comunicação ponto a ponto entre processos executados em máquinas separadas. Para isso, dois protocolos são especificados: TCP (*Transmission Control Protocol*) e UDP (*User Datagram Protocol*).

A última camada é a de aplicativos. Nela, os usuários podem executar programas que acessam serviços disponíveis através da interligação em redes TCP/IP. Um aplicativo interage com um dos protocolos do nível de transporte para enviar ou receber dados, tais como: os programas de FTP (transferência de arquivos), HTTP (páginas da Internet), DNS (resolvedor de nomes) e TELNET (emulador de terminal).

G.2 Protocolo IP

O IP (*Internet Protocol*) é protocolo padrão para a comunicação de dados pela Internet. Ele é caracterizado por fornecer um serviço sem conexão e não-confiável, baseado em *datagramas*. Um *datagrama* IP é composto de um cabeçalho, contendo informações de controle e um campo de dados para encapsular o protocolo de transporte (TCP), e correspondem à unidade básica de transferência do protocolo IP.

O serviço é tido como não-confiável porque a entrega não é garantida. Todo o possível é feito para que o pacote seja recebido pelo seu destinatário, entretanto, em alguns casos, como falta de recursos ou falhas na rede, ele pode simplesmente ser perdido, atrasar-se ou ser entregue com problemas, sem que notificações de erro sejam dadas.

No IP, cada pacote é independente um do outro, ou seja, os caminhos por onde os *datagramas* trafegam são determinados dinamicamente, através dos algoritmos de roteamento deste protocolo. Sendo assim, dois pacotes de uma mesma origem e destinados a um *host* comum podem seguir rotas distintas.

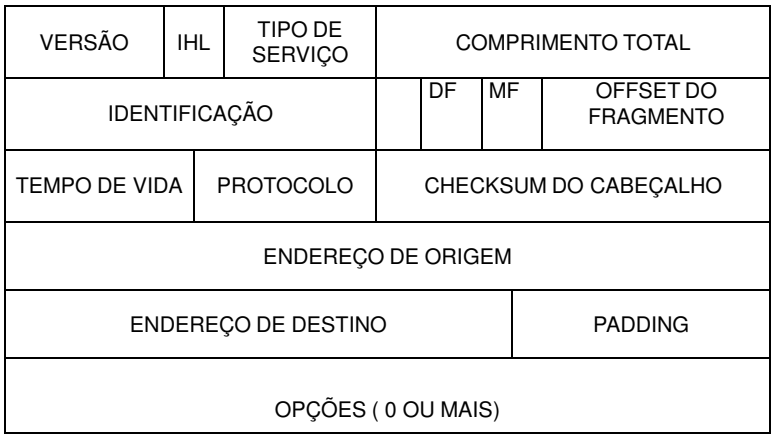
Para um maior esclarecimento sobre o funcionamento do protocolo IP, é preciso conhecer o formato de seu pacote, mais especificamente de seu cabeçalho. A Figura G.1 ilustra o cabeçalho IP.

O cabeçalho do datagrama tem uma parte fixa de vinte bytes e uma parte variável, destinada a possíveis expansões no protocolo. Os campos da parte fixa do cabeçalho mais importantes para este trabalho serão explicados em seguida.

O campo IDENTIFICAÇÃO é um número inteiro de dezesseis bits utilizado para identificar um *datagrama*. DF significa 'DO NOT FRAGMENT' e é uma ordem expressa para que o pacote não seja fragmentado. MF é o acrônimo de 'MORE FRAGMENTS' e indica que há mais fragmentos para o pacote. OFFSET DO FRAGMENTO informa a que ponto do *datagrama* o fragmento pertence.

Os quatro campos acima trabalham em conjunto para controlar a fragmentação e a remontagem do *datagrama*. A fragmentação acontece devido ao pacote, durante o seu percurso, trafegar por diversos tipos de rede, com tamanhos de blocos de transmissão diferentes. Assim, é possível que ele passe por redes onde o MTU (tamanho máximo de unidade) seja tal que não

Figura G.1 Cabeçalho IP [39].



consiga suportar o seu tamanho. Para resolver este problema, o *datagrama* será fragmentado em pacotes menores e independentes que serão enviados ao destino.

O campo TEMPO DE VIDA especifica o tempo, em segundos, que um *datagrama* pode continuar existindo. Isto previne situações de 'vida eterna' de *datagramas* devido a erros de rotas ou mesmo *loops*.

O campo PROTOCOLO (8 bits) indica o protocolo usuário do IP, cujos dados são transportados no campo de dados. Pode ser TCP (6), UDP (17) ou ICMP (1). Os últimos campos relevantes são os ENDEREÇO DE ORIGEM (32 bits) e ENDEREÇO DE DESTINO (32 bits).

G.3 Protocolo TCP

Este é um dos protocolos da arquitetura TCP/IP que trabalha na camada de transporte, especificado na RFC 793. Seu principal objetivo é fornecer um serviço fim a fim confiável entre processos localizados em hosts distintos, independente de quão imperfeitas sejam as redes utilizadas. Para isso, ele busca garantir um fluxo de bytes livre de erros entre as entidades TCP, uma em cada computador.

Para se utilizar este protocolo, é preciso a criação de pontos terminais, conhecidos como *sockets*, entre os pares envolvidos na comunicação. Os sockets são formados pela união de um endereço IP e uma porta. Porta é uma abstração usada para definir um determinado serviço dentre os vários que podem rodar em um host. Um serviço de FTP, por exemplo, é identificado pela porta 21 e para ser acessado, é preciso que se conheça também o endereço de rede da máquina (onde o serviço está).

A comunicação entre as entidades TCP transmissora e receptora se dá por meio de unidades denominadas segmentos. A Figura G.2 ilustra o formato do cabeçalho de um segmento TCP.

Figura G.2 Cabeçalho TCP [39].

<table>
<tr><td colspan="3">PORTA ORIGEM</td><td>PORTA DESTINO</td></tr>
<tr><td colspan="4">NÚMERO DE SEQUÊNCIA</td></tr>
<tr><td colspan="4">NÚMERO ACK</td></tr>
<tr><td>Hlen</td><td>RESERV</td><td>BITS DE CÓDIGO</td><td>TAMANHO DA JANELA</td></tr>
<tr><td colspan="3">CHECKSUM</td><td>PONTEIRO DE URGÊNCIA</td></tr>
<tr><td colspan="4">OPÇÕES (O OU MAIS)</td></tr>
</table>

O cabeçalho do segmento tem uma parte fixa de vinte bytes e uma parte variável, destinada a possíveis expansões no protocolo. Os campos da parte fixa do cabeçalho relevantes ao presente propósito serão explicados em seguida.

Os campos PORTA ORIGEM e PORTA DESTINO contêm os números da portas TCP associadas às aplicações em cada ponto da conexão.

O campo NÚMERO DE SEQUÊNCIA identifica o número inteiro sequencial que identifica um segmento.

O campo NÚMERO ACK (*acknowledgement number*) identifica o número do próximo segmento que a origem espera receber. É interessante observar que o número de sequência faz o controle do fluxo de dados na transmissão, enquanto o número de ACK faz o controle na direção oposta, ou seja, de recepção.

O campo PONTEIRO DE URGÊNCIA é usado para identificar dados urgentes.

O campo BITS DE CÓDIGO (code bits) contém 6 bits que devem ser usados para determinar o propósito e o conteúdo do segmento. Eles mostram como interpretar os outros campos no cabeçalho. São eles, com suas respectivas funções, caso tenham valor positivo: URG (PONTEIRO DE URGÊNCIA é válido), ACK (NÚMERO ACK é válido), PSH (o segmento deve ser transmitido logo), RST (reiniciar a conexão), SYN (estabelecer conexões) e FIN (finalizar conexões).

O protocolo TCP é orientado a conexão. Isto implica na necessidade de uma etapa de negociação antes que as informações úteis sejam de fato transmitidas. Este processo se dá em 3 passos, e por isso é conhecido como *Three Way Handshake*. Eis a explicação: uma máquina A que deseja se comunicar com uma máquina B envia para esta um pacote com o campo SYN ligado. Quando B recebe este pacote, ele envia de volta um datagrama de reconhecimento com os campos SYN e ACK ligados, em resposta ao primeiro pacote. Por fim, A envia para B outro pacote de reconhecimento com o ACK ligado e a negociação se completa.

G.4 Protocolo UDP

O outro protocolo da camada de transporte da arquitetura TCP/IP é o UDP (*User Datagram Protocol*), especificado na RFC 768. Ele oferece um serviço não orientado a conexão

e não confiável para a comunicação entre dois processos em máquinas distintas. Este protocolo parte do pressuposto que é da aplicação a responsabilidade de fornecer uma transmissão de dados de forma correta. Assim, ele não se preocupa com confirmações de dados recebidos, com o ordenamento das mensagens ou com o controle da velocidade de transmissão entre as máquinas.

O cabeçalho do segmento UDP é retratado na Figura G.3:

Figura G.3 Cabeçalho UDP [39].

PORTA DE ORIGEM	PORTA DE DESTINO
COMPRIMENTO DA MENSAGEM	SOMA DE VERIFICAÇÃO

Os campos relevantes para o entendimento deste trabalho são as PORTA DE ORIGEM e PORTA DE DESTINO do pacote.

G.5 Protocolo ICMP

O ICMP (*Internet Control Message Protocol*) é responsável pelo transporte de mensagens de erro e de controle entre os roteadores ou estações. Ele possibilita a comunicação entre o software IP de uma máquina e o software IP de outra para a transmissão de notificações sobre algum tipo de mau funcionamento de um ponto da rede. Para isso, a mensagem ICMP é encapsulada em um pacote IP e é enviada normalmente.

Este protocolo disponibiliza diversas mensagens de controle. As mais importantes são mostradas na Tabela G.1, com uma breve explicação.

Tabela G.1 Principais mensagens do protocolo ICMP [39].

Tipo de Mensagem	Explicação
Destinatário inacessível	O pacote não pode ser entregue
Tempo excedido	TEMPO DE VIDA de um pacote expira
Problema de parâmetro	Há erros no cabeçalho do pacote IP
Ajuste de fonte	Pacote que regula a taxa de envio do emissor
Redirecionamento	Mostrar a rota correta de um pacote
Requisição de eco (*echo request*)	Verifica se uma máquina está disponível
Resposta ao eco (*echo reply*)	Confirmar que a máquina está disponível
Marca de tempo	Usado para medir o atraso no envio de pacotes
Resposta a marca do tempo	Usado para medir o atraso no envio de pacotes

O formato de cada mensagem ICMP varia mas existem três campos em comum no cabeçalho:

- Tipo: define o tipo da mensagem (significado e formato), mostrados na Tabela G.1;

- Código: prevê um detalhamento maior sobre o tipo da mensagem;
- Checksum: idêntico ao algoritmo utilizado pelo IP.

APÊNDICE H

Padrões de Ataques no KDDCUP 99

Existem 22 diferentes padrões de ataque na base de dados KDDCUP 99, utilizada neste trabalho. Uma breve descrição da operação de cada um destes ataques, por categoria, será apresentada neste apêndice, com base em [9, 175]. Para um melhor entendimento deste apêndice, é necessário uma breve leitura do Apêndice G.

H.1 Negação de serviço (DoS)

A negação de serviço (DoS) é a categoria de ataque, cujo objetivo do invasor é causar a indisponibilidade do serviço alvo, seja por meio do consumo de recursos computacionais, tais como memória, processador ou rede. Seis tipos de ataques DoS foram utilizados nos experimentos, a saber:

- ***Back***

Direcionado contra servidores Web Apache, o ataque *Back* consiste no envio de solicitação HTTP (*HyperText Transfer Protocol*) contendo URL (*Uniform Resource Locator*) com um elevado número de caracteres '/' (em geral, mais de 100). Ao receber este tipo de solicitação, os servidores Web Apache2 aumentam sua utilização de UCP, afetando o seu desempenho geral. O sistema consegue se recuperar após o termino do ataque.

- ***Land***

O padrão de ataque *Land* tem, como característica, pacotes que possuem os endereços IP/Porta de origem e destino iguais. Esse tipo de ataque pode colocar um sistema em *loop* infinito. Algumas versões de sistemas operacionais, como o *Windows* 95 e *Windows NT Workstation* com *Service Pack* 3 têm suas máquinas travadas quando sofrem esse tipo de ataque pela porta 139 e as distribuições mais antigas do Linux apresentam queda de desempenho quando várias cópias desse ataque são disparadas contra ele.

- ***Neptune***

Ataque *Neptune*, também conhecido como *SynFlood*, utiliza uma técnica denominada *IP Spoofing* que pode ser entendida como uma espécie de falsidade ideológica em relação ao endereço IP do host do atacante, ou seja, o hacker manipula o cabeçalho do pacote alterando o IP origem para outro diferente do verdadeiro.

No ataque SynFlood, o invasor envia inúmeros pacotes com o *flag SYN* ativado (ver Apêndice G). Tais pacotes têm o seu endereço de origem alterado para números IPs falsos (*ip spoofing*). Desta forma, o processo de *handshake* não é completado e o computador alvo do ataque manterá os recursos alocados para estas conexões, até que todos os seus recursos sejam consumidos, e o mesmo recuse as requisições dos usuários reais.

- ***Ping of Death (Pod)***

O ataque *ping of death* (Pod) consiste no envio de pacotes ICMP *echo request* (gerados normalmente através do utilitário 'ping' para diagnóstico) com tamanhos superiores a 64000 bytes. Como os pacotes ICMP *echo request* são normalmente bem inferiores a este valor, diversos sistemas operacionais e implementações de TCP/IP disponíveis não conseguiam tratar esta anomalia, e reagiam de forma inesperada, cujas reações mais comuns eram a reinicialização do computador alvo ou total paralisação de todas as operações do sistema.

- ***Smurf***

Em um ataque do tipo *Smurf*, o invasor envia pacotes ICMP *echo request*, com endereço de destino de 'broadcast', ou seja, a requisição irá para todas as máquinas em uma determinada rede, e endereço de origem igual ao do sistema alvo do ataque. O sistema vítima terá uma sobrecarga, ao receber milhares de respostas de solicitações (pacotes ICMP *echo reply*) que o mesmo não solicitou.

- ***Teardrop***

O ataque *Teardrop* consiste em enviar fragmentos IP que não podem ser reagrupados porque o valor do *offset* do pacote foi adulterado. Este ataque causava a reinicialização do sistema operacional, ou congelamento do mesmo, se uma situação não esperada, como esta, fosse encontrada.

H.2 Reconhecimento *(Probing)*

Na categoria de ataque do tipo *Probing*, um intruso varre uma rede em busca de informação, ou para encontrar vulnerabilidades conhecidas para explorar. Quatro tipos de ataques *Probing* foram utilizados nos experimentos, a saber:

- ***IPSweep***

IPSweep é um ataque utilizado na etapa de reconhecimento e seleção de alvos vulneráveis. Consiste em uma varredura ou busca por computadores e sistemas acessíveis pela rede e que possam ser atacados por outros métodos em uma próxima etapa. O ataque *IPSweep* consiste em enviar pacotes ICMP *echo request* (utilitário ping) para todos os endereços de uma determinada rede e constrói uma relação de respostas recebidas (pacotes ICMP *echo reply*).

- ***PortSweep***

O *PortSweep* é um ataque da família das varredores de portas (*Port Scanners*). Este ataque consiste no envio de pacotes destinados a diferentes de portas de serviço oriundas de um mesmo destinatário e detectável no pequeno intervalo de segundos. A estratégia no uso da varredura de portas objetiva identificar os serviços disponíveis servidores, e assim explorar as possíveis vulnerabilidades, presentes nas diferentes implementações que suportam as facilidades oferecidas.

- ***Nmap***

Nmap é uma ferramenta de varredura e reconhecimento capaz de realizar busca por computadores, serviços e vulnerabilidades, utilizando diversos mecanismos. Sua principal função é identificar quais portas TCP ou UDP estão ativas em um determinado computador ou rede de computadores, e, consequentemente, determinar quais serviços estão ativos. Além disso, a ferramenta permite a identificação exata da versão dos softwares que implementam os serviços descobertos. Esta informação é muito útil para um invasor determinar quais vulnerabilidades podem ser exploradas, objetivando ter sucesso no ataque à rede. Um aspecto da ferramenta Nmap que pode dificultar sua detecção é a capacidade de configurar diversos aspectos relativos à velocidade e a periodicidade com que as varreduras são realizadas.

- ***Satan***

Security Administrator Tool for Analyzing Networks (Satan) é uma ferramenta desenvolvida para administradores de redes e segurança da informação, que tem como objetivo obter o maior número de informações sobre serviços ativos em computadores remotos. Apesar de não ter sido desenvolvida com a intenção de ser utilizada para efetuar ataques, os tipos de informações levantadas por esta ferramenta pode ser útil, também, para um invasor.

H.3 Remoto para Local (R2L)

A categoria de ataque do tipo *R2L* é identificada quando uma pessoa, em uma máquina remota, tenta obter acesso ao servidor local. Tem-se oito tipos de ataques dessa categoria na base de dados KDDCPU 99, a saber:

- ***Ftp–write***

Ftp–write é o padrão de ataque que explora uma falha de configuração em serviços FTP, que fornecem acesso para usuários não autenticados (anônimos). Se o diretório raiz de um servidor FTP tiver como proprietário o usuário 'ftp' ou seu proprietário estiver no mesmo grupo do usuário 'ftp', é possível que um usuário anônimo tenha acesso, com privilégio de escrita em arquivos, no servidor FTP, e até mesmo obtenha acesso remoto ao servidor.

- ***Guess–passwd***

A forma clássica de impor segurança a um determinado sistema ou serviço é através da autenticação, utilizando o par usuário/senha. O padrão de ataque *guess–passwd* consiste na obtenção de acesso não autorizado a um determinado computador ou serviço através de inúmeras tentativas de descobrir a conta e senha de um usuário válido. Nesse contexto, percebe-se que a principal vulnerabilidade explorada por este ataque é humana, e reside no fato de que os usuários tipicamente utilizam senhas fracas, passíveis de serem descobertas por métodos automatizados de adivinhação de senhas, permitindo que intrusos tenham, pelo menos, o acesso remoto inicial.

- ***Imap***

Imap é um ataque que explora uma falha de estouro de *buffer* (*buffer overflow*) no serviço *Imap* de servidores *RedHat Linux* 4.2 e que permite ao invasor executar um código arbitrário no servidor afetado.

- ***Phf***

Phf é um *script* CGI (*Common Gateway Interface*) que permite executar comando num servidor Web Apache mal configurado. Por exemplo, um invasor copiar o arquivo 'passwd', que contém os usuários e suas respectivas senhas, de um sistema Unix vulnerável.

- ***Multihop***

No ataque *multihop*, um invasor primeiro acessa uma máquina e, em seguida, a utiliza como trampolim para atacar outras máquinas.

- ***Spy***

No tipo de ataque *Spy*, o invasor instala uma ferramenta, em uma máquina que obteve acesso, capaz de capturar os pacotes que estão trafegando na rede (*packet sniffing*), por meio da interface de rede, em busca de senhas e outras informações de usuários, antes de penetrar no sistema alvo.

- ***Warezmaster***

O ataque *warezmaster* ocorre por meio de uma conexão FTP anônima, em que o invasor faz *upload* de cópias ilegais de software (*Warez*) em um servidor FTP.

- ***Warezclient***

No ataque *warezclient*, invasores fazem *download* de software ilegal, publicado via FTP anônima por *warezmaster*.

H.4 Usuário para Superusuário (U2R)

Na categoria de ataque do tipo U2R, quando um invasor, que possui acesso autorizado como um usuário normal, tenta obter acesso como superusuário. Quatro tipos de ataques *U2R* foram utilizados nos experimentos, a saber:

- ***Buffer overflow***

Estouro de *buffer* (*buffer overflow*) acontece quando um invasor envia uma quantidade maior de dados que o *buffer* pode armazenar, e isso sem a checagem dos limites da estrutura, causando o estouro do buffer. Trata-se de uma técnica mais refinada de invasão, em que o invasor manipula o conteúdo do *buffer* de um programa para obter acesso privilegiado no sistema alvo.

- ***Loadmodule***

Ataque contra o sistema operacional SunOS 4.1, que utiliza o *xnews windows system*. O programa *xnews* utiliza o módulo *loadmodule* para carregar alguns dispositivos em memória. Devido a um erro de programação no módulo *loadmodule*, um invasor pode obter privilégios de superusuário.

- ***Perl***

O ataque *perl* explora falha em algumas versões da linguagem de *scripts Perl*. Um módulo denominado *suidperl*, ou *sperl*, programa que dá suporte ao salvamento do ID do usuário e do grupo, ao apresentar erro de programação, permite que um invasor ou um usuário qualquer, com conta ativa no sistema alvo, obtenha privilégios de superusuário.

- ***Rootkit***

Rootkits são programas modificados que podem substituir aplicativos importantes do sistema operacional, como também podem omitir processos, conexões, arquivos e arquivos de auditoria manipulados pelo invasor.

APÊNDICE I

Exemplo de Construção de Árvore de Decisão

Tomando como exemplo ilustrativo o problema de classificar uma atividade do sistema como um ataque a uma rede de computadores ou como um tráfego normal, tem-se um conjunto de treinamento contendo os atributos: Protocolo, Usuário, Horário e Permissão, conforme ilustrado na Tabela I.1.

Tabela I.1 Exemplo - Verificar se uma atividade do sistema é um ataque.

ATRIBUTOS				**CLASSE**
Protocolo	Usuário	Horário	Permissão	Ataque?
http	A	Noite	Falso	**Sim**
http	B	Noite	verdade	**Sim**
http	C	Dia	Falso	Não
http	C	Dia	verdade	Não
http	B	Dia	Falso	Não
smtp	C	Dia	Falso	**Sim**
smtp	A	Noite	verdade	**Sim**
smtp	B	Dia	verdade	**Sim**
smtp	C	Noite	Falso	**Sim**
https	B	Dia	Falso	**Sim**
https	A	Noite	Falso	**Sim**
https	B	Noite	Falso	**Sim**
https	A	Noite	verdade	Não
https	B	Dia	verdade	Não

Sejam C e A duas variáveis aleatórias (definidas no Capítulo 3), então, utilizando os dados da Tabela I.1 e a Equação (C.1), tem-se que:

$$\begin{aligned} H(C) = & \ -\tfrac{9}{14}\log_2\tfrac{9}{14} - \tfrac{5}{14}\log_2\tfrac{5}{14}. \\ = & \ 0,940 \text{ bits.} \end{aligned} \tag{I.1}$$

Com base na Equação (C.5), será necessário calcular a entropia condicional de todos os atributos do conjunto de instâncias T, para posteriormente escolher o melhor atributo a ser utilizado como nó raiz da árvore. O atributo A_j usado como nó raiz da árvore de decisão irá dividir o conjunto de treinamento T em η partições.

A estimativa de probabilidades, extraída da Tabela I.1, é apresentada na Tabela I.2:

Tabela I.2 Distribuição de probabilidades.

Classe	**Protocolo**			**Usuário**			**Horário**		**Permissão**	
	http	**smtp**	**https**	**A**	**B**	**C**	**Noite**	**Dia**	**Falso**	**Verdade**
Sim	2/5	4/4	3/5	3/4	4/6	2/4	6/7	3/7	6/8	3/6
Não	3/5	0	2/5	1/4	2/6	2/4	1/7	4/7	2/8	3/6

Com base na Tabela I.2 e Equação C.4, seguem os cálculos:

- Para o atributo A_1, Protocolo, tendo $V_1 = \{$http, https, smtp$\}$:

$$\begin{aligned} H(C|A) = &\ \tfrac{5}{14}(-\tfrac{2}{5}\log_2\tfrac{2}{5} - \tfrac{3}{5}\log_2\tfrac{3}{5}) + \tfrac{4}{14}(-\tfrac{4}{4}\log_2\tfrac{4}{4} - 0) + \tfrac{5}{14}(-\tfrac{3}{5}\log_2\tfrac{3}{5} - \tfrac{2}{5}\log_2\tfrac{2}{5}) \\ = &\ 0,694. \end{aligned} \tag{I.2}$$

- Para o atributo A_2, Usuário, tendo $V_2 = \{$A, B, C$\}$:

$$\begin{aligned} H(C|A) = &\ \tfrac{4}{14}(-\tfrac{3}{4}\log_2\tfrac{3}{4} - \tfrac{1}{4}\log_2\tfrac{1}{4}) + \tfrac{6}{14}(-\tfrac{4}{6}\log_2\tfrac{4}{6} - \tfrac{2}{6}\log_2\tfrac{2}{6}) + \tfrac{4}{14}(-\tfrac{2}{4}\log_2\tfrac{2}{4} - \tfrac{2}{4}\log_2\tfrac{2}{4}) \\ = &\ 0,911. \end{aligned} \tag{I.3}$$

- Para o atributo A_3, Horário, tendo $V_3 = \{$Noite, Dia$\}$:

$$\begin{aligned} H(C|A) = &\ \tfrac{7}{14}(-\tfrac{6}{7}\log_2\tfrac{6}{7} - \tfrac{1}{7}\log_2\tfrac{1}{7}) + \tfrac{7}{14}(-\tfrac{3}{7}\log_2\tfrac{3}{7} - \tfrac{4}{7}\log_2\tfrac{4}{7}) \\ = &\ 0,788. \end{aligned} \tag{I.4}$$

- Para o atributo A_4, Permissão, tendo $V_4 = \{$falso, verdade$\}$:

$$\begin{aligned} H(C|A) = &\ \tfrac{8}{14}(-\tfrac{6}{8}\log_2\tfrac{6}{8} - \tfrac{2}{8}\log_2\tfrac{2}{8}) + \tfrac{6}{14}(-\tfrac{3}{6}\log_2\tfrac{3}{6} - \tfrac{3}{6}\log_2\tfrac{3}{6}) \\ = &\ 0,892. \end{aligned} \tag{I.5}$$

Seguindo o exemplo da Tabela I.1, o cálculo da informação mútua para os atributos $A_1 =$ Protocolo, $A_2 =$ Usuário, $A_3 =$ Horário e $A_4 =$ Permissão será realizado a seguir:

- Para o atributo A_1 (Protocolo):

$$\begin{aligned} I(C;A) = &\ H(C) - H(C|A) \\ = &\ 0,940 - 0,694 \\ = &\ 0,246. \end{aligned} \tag{I.6}$$

- Para o atributo A_2 (Usuário):

$$\begin{aligned} I(C;A) &= H(C) - H(C|A) \\ &= 0,940 - 0,911 \\ &= 0,029. \end{aligned} \tag{I.7}$$

- Para o atributo A_3 (Horário):

$$\begin{aligned} I(C;A) &= H(C) - H(C|A) \\ &= 0,940 - 0,788 \\ &= 0,152. \end{aligned} \tag{I.8}$$

- Para o atributo A_4 (Permissão):

$$\begin{aligned} I(C;A) &= H(C) - H(C|A) \\ &= 0,940 - 0,892 \\ &= 0,048. \end{aligned} \tag{I.9}$$

Após o cálculo da informação mútua, calcula-se a razão do ganho. Então:

- Para o atributo A_1, Protocolo, tendo $V_1 = \{$http, https, smtp$\}$:

$$\begin{aligned} H(A) &= -\tfrac{5}{14}\log_2\tfrac{5}{14} - \tfrac{4}{14}\log_2\tfrac{4}{14} - \tfrac{5}{14}\log_2\tfrac{5}{14} \\ &= 1,5774\text{bits}. \end{aligned} \tag{I.10}$$

Calculando $\frac{I(C;A)}{H(A)} = 0,156$.

- Para o atributo A_2, Usuário, tendo $V_2 = \{$A, B, C$\}$:

$$\begin{aligned} H(A) &= -\tfrac{4}{14}\log_2\tfrac{4}{14} - \tfrac{6}{14}\log_2\tfrac{6}{14} - \tfrac{4}{14}\log_2\tfrac{4}{14} \\ &= 1,5567\text{bits}. \end{aligned} \tag{I.11}$$

Calculando $\frac{I(C;A)}{H(A)} = 0,0186$.

- Para o atributo A_3, Horário, tendo $V_3 = \{$Noite, Dia$\}$:

$$\begin{aligned} H(A) &= -\tfrac{7}{14}\log_2\tfrac{7}{14} - \tfrac{7}{14}\log_2\tfrac{7}{14} \\ &= 1\text{bit}. \end{aligned} \tag{I.12}$$

Calculando $\frac{I(C;A)}{H(A)} = 0,151$.

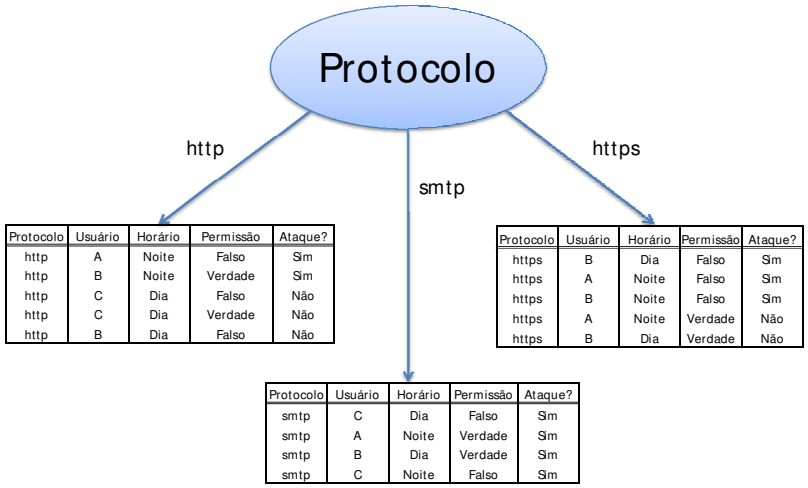
- Para o atributo A_4, Permissão, tendo V_4 = {falso, verdade}:

$$\begin{aligned} H(A) = & -\frac{8}{14}\log_2\frac{8}{14} - \frac{6}{14}\log_2\frac{6}{14} \\ = & \ 0,9852\text{bit}. \end{aligned} \tag{I.13}$$

Calculando $\frac{I(C;A)}{H(A)} = 0,0487$.

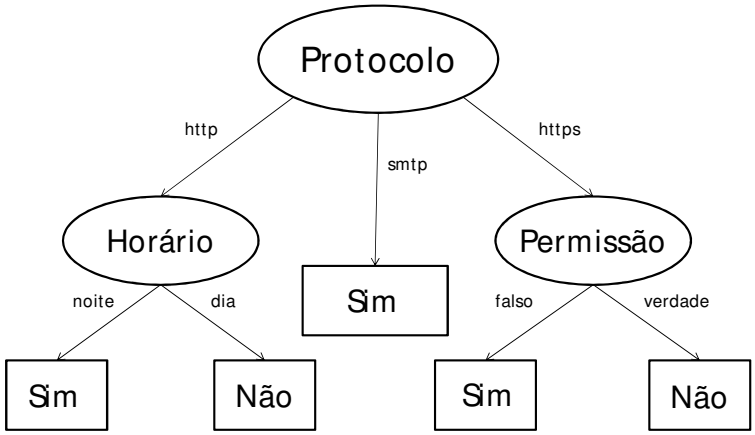
Portanto, o atributo Protocolo será escolhido como nó raiz. Como este atributo possui três valores discretos, então a base de dados será subdividida em 3 partições, conforme ilustrado na Figura I.1, e a construção da árvore continua, em um processo recursivo, para os subconjuntos residuais $T_1, ..., T_\nu$.

Figura I.1 Processo de construção de uma árvore de decisão.



A árvore de decisão C4.5, gerada a partir do exemplo da Tabela I.1, é ilustrada na Figura I.2.

Figura I.2 Árvore de decisão criada a partir de um conjunto de treinamento T.



APÊNDICE J

Resultados Adicionais

Neste apêndice são reunidas informações e resultados adicionais dos experimentos executados nesta Tese.

J.1 Estudo Comparativo entre Árvore de Decisão C4.5 e Sistemas Imunológicos Artificiais

A construção de sistemas seguros e invioláveis depende diretamente do mecanismo de identificação e tratamento de falhas empregado. Árvores de Decisão e Sistemas Imunológicos Artificiais têm sido utilizados em modelos de detecção e classificação de problemas relacionados à detecção de intrusos em redes de computadores. Eles apresentam bons resultados na detecção de diferentes tipos de ataques. Entretanto, ainda geram alarmes falsos ou ignoram ataques conhecidos e que são interpretados como execução normal no sistema. Portanto, neste apêndice, apresenta-se um estudo comparativo de tais modelos com o objetivo de encontrar alternativas mais eficientes na construção de um Sistema Tolerante a Intrusões.

O objetivo principal desta pesquisa foi realizar um estudo comparativo entre os modelos baseados em Árvore de Decisão C4.5 [83], descrita no Capítulo 3, e os Sistemas Imunológicos Artificiais [72, 73] aplicados na detecção de intrusos, visando encontrar alternativas mais eficientes na construção de um Sistema Tolerante a Intrusões.

Dessa forma, pretende-se aplicar o conhecimento adquirido, e combinar os resultados individuais de cada modelo em um resultado global de melhor qualidade, tentando minimizar o problema dos alarmes falsos e ataques legítimos ignorados, a fim de serem aplicados na detecção de intrusos em redes de computadores.

J.1.1 Sistema Imunológico e Sistemas Imunológicos Artificiais

O sistema imunológico dos vertebrados é um sistema complexo, composto por uma grande variedade de componentes celulares e moleculares espalhados por todo o corpo. A

tarefa principal deste sistema é a de proteger o organismo contra os organismos ou substâncias estranhas.

O sistema imunológico proporciona uma arquitetura tolerante a falhas distribuída dentro do corpo. Pode ser visto como um mecanismo de proteção de múltiplas camadas, composto por duas subdivisões principais: o sistema inato e o sistema adaptativo [176].

O sistema inato é a primeira linha de defesa do organismo, composto por elementos, tais como a pele e as barreiras naturais. É um sistema não específico e reage igualmente e consegue combater uma variedade de organismos estranhos.

O sistema imune adaptativo age como uma segunda linha de defesa e também proporciona proteção contra a exposição subsequente ao mesmo organismo estranho. É constituído por vários tipos de células, tais como, linfócitos B (células B) e linfócitos T (células T), a fim de determinar a especificidade da resposta imunológica contra os antígenos do organismo. Estas duas células receberam seus nomes de acordo com o órgão onde ocorre a sua maturação: a medula óssea e o timo, respectivamente. Os precursores das células T também são produzidos na medula óssea, mas depois que as células T são geradas, elas migram para o timo, onde são submetidas à maturação.

O timo desempenha um papel crucial na maturação das células T. Durante o processo de maturação, todas as células T que reconhecem auto-antígenos (auto-reatividade) são excluídas da população de células T por meio de um processo chamado de seleção negativa. Após isto, as células-T maturadas saem do timo, e, como resultado, o corpo é capaz de realizar a discriminação próprio e não próprio ao organismo.

As células B, que circulam no sangue e proporcionam imunidade humoral, produzem os anticorpos (proteínas) que são capazes de reagir a alguns antígenos específicos. Quando uma célula B encontra um antígeno com afinidade suficiente (grau de similaridade entre as células e o reconhecimento do antígeno), este prolifera e produz linfoblastos B, que são diferenciados em células tanto de plasma, que produzem anticorpos ou células de memória (clones utilizados no caso de reaparecimento do antígeno), através de um processo denominado seleção clonal. Estes clones são mutados, usando um processo de mutação somática e as células sobreviventes são selecionadas pelo processo de seleção clonal.

Estes mecanismos são realmente interessantes não só do ponto de vista biológico, mas também sob uma perspectiva computacional. Nessa perspectiva, um sistema imunológico artificial (AIS) é uma classe de algoritmo computacional adaptável, que emula os processos e mecanismos de inspiração biológicas, provenientes de sistemas imunológicos. Esses algoritmos utilizam a aprendizagem, memória e capacidades de otimização do sistema imunológico para desenvolver ferramentas computacionais para otimização, classificação, reconhecimento de padrões, controle de processo, dentre outros, bem como são usados para desenvolver sistemas adaptativos, capazes de resolver problemas em domínios diferentes.

Nas seções seguintes, apresenta-se brevemente os algoritmos CLONALG, CSCA, IMMUNOS e AIRS, e suas habilidades para categorizar invasão de rede são avaliadas e comparadas.

Estes algoritmos simulam o processo de reconhecimento do antígeno-anticorpo pela evolução de uma população de células B que aprende a reconhecer antígenos (padrões a partir de um conjunto de treinamento).

J.1.2 Algoritmo CLONALG

O algoritmo de seleção clonal CLONALG [77] foi inicialmente desenvolvido para resolver problemas de aprendizado de máquina e de reconhecimento de padrões, sendo depois estendido para problemas de otimização. Este algoritmo consiste em submeter uma população de anticorpos a uma exposição repetida a um conjunto de antígenos, por meio de um determinado número de gerações, com o objetivo de desenvolver uma população de anticorpos que seja mais sensível ao estímulo dos antígenos. O algoritmo básico é o seguinte:

1. **Inicialização**. O algoritmo CLONALG inicia pela geração aleatória de uma população inicial Ab de τ anticorpos. Estes anticorpos são subdivididos em: memória população de anticorpos Ab_m, de tamanho m, que no final do processo de treinamento, representa a solução do modelo CLONALG, e Ab_r, de tamanho r, usado para adicionar diversidade na fase de aprendizagem. A inicialização é realizada de forma que os conjuntos Ab_m e Ab_r tenham a mesma distribuição de anticorpos entre as classes e o conjunto Ag de antígenos. O conjunto de anticorpos Ab é definido como $Ab = Ab_m \cup Ab_r (r = \tau - m)$.

2. **Train antibodies**. O treinamento de anticorpos Ab é um processo interativo de exposição do sistema ao conjunto Ag de M antígenos presentes no conjunto de treinamento durante G gerações (definido pelo usuário). Em cada geração, um antígeno é selecionado para o treinamento apenas uma vez. Os passos seguintes são realizados para todos os M antígenos em Ag:

   - **Seleção de Antígeno**. Um antígeno ag_i (em que $1 \leq i \leq M$) é selecionado aleatoriamente sem reposição da população Ag.
   - **Exposição**. Para todos τ anticorpos em Ab, o valor da afinidade entre ag_i e os τ anticorpos é calculada, usando uma dada função de afinidade antigênica. É comum usar a distância de *Hamming*.
   - **Seleção**. Um conjunto Ab_s de s anticorpos (definido pelo usuário) que tiverem maior afinidade antigênica com ag_i são selecionados da população Ab.
   - **Clonagem**. Os s anticorpos do conjunto Ab_s são clonados independentemente e proporcionalmente à suas afinidades, gerando C clones. O número de clones criados de cada anticorpo ab_j é calculado, usando a Equação (J.1).

$$numClones_j = \left\lfloor \frac{\beta\tau}{i} + 0.5 \right\rfloor, \tag{J.1}$$

em que β é o fator clonal ($0 < \beta \leq 1$), e i ($1 \leq i \leq s$) é o índice na lista de anticorpos do conjunto Ab ordenado de forma decrescente pelo valor de afinidade.

O número total de clones $N_c = |C|$ dado por cada antígeno exposto é então calculado usando a Equação (J.2).

$$N_c = \sum_{j=1}^{s} numClones_j. \tag{J.2}$$

- **Mutação**. Os C clones são submetidos ao processo de maturação de afinidade inversamente proporcional à suas afinidades antigêncas, gerando a população C^* de clones maturados. Quanto maior a afinidade, menor será a taxa de mutação.
- **Avaliação dos clones**. Cada clone de C^* é exposto ao antígeno e sua afinidade é calculada.
- **Seleção dos candidatos**. Os anticorpos com mais alta afinidade são selecionados para substituírem os anticorpos de Ab_m que têm baixa afinidade.
- **Substituição**. Os v anticorpos (em que $0 \leq v \leq r$ é definido pelo usuário) com baixa afinidade na população Ab_r são substituídos por anticorpos aleatórios.

3. **Finalização**. Após o treinamento do sistema por G gerações, o grupo Ab_m de anticorpos representa a solução do classificador CLONALG.

Este classificador é então usado para fazer predições de novos ataques que não foram usados no processo de treinamento do modelo.

J.1.3 Algoritmo CSCA

O sistema de classificação clonal CSCA foi desenvolvido por [78] e é formulado como uma função de otimização que maximiza o número de padrões corretamente classificados e minimiza o número de padrões incorretamente classificados. O algoritmo CSCA é treinado por várias gerações e, durante cada geração, o conjunto de entrada de anticorpos é exposto a todos os antígenos. O algoritmo CSCA consiste nos seguintes passos:

1. **Inicialização**. O algoritmo CSCA começa gerando aleatoriamente a população inicial de S anticorpos (Ab).

2. **Treinamento**. Consiste em um processo de expor todo o grupo de anticorpos Ab ao grupo de antígenos Ag, da base de dados de treinamento, durante G gerações.

   - **Seleção e Remoção**. O grupo de anticorpos Ab é exposto ao conjunto de antígenos Ag e uma pontuação de aptidão (*fitness score*) é calculada para cada anticorpo, usando a seguinte Equação (J.3).

$$f = max\left(\frac{correct}{incorrect}\right). \tag{J.3}$$

Os anticorpos serão selecionados após a aplicação das regras de avaliação a seguir:

(a) Anticorpos que têm classificação correta igual a zero e erro de classificação maior que zero são atribuídos à classe majoritária e a aptidão é recalculada.

(b) Os anticorpos com aptidão menor que um limiar ($\mathcal{E}$), serão removidos da população de anticorpos testada.

(c) Todos os anticorpos com zero erro de classificação não participarão na clonagem e maturação.

- **Clonagem e mutação**. O conjunto selecionado de anticorpos é clonado e mutado, proporcionalmente à sua aptidão para assegurar que aqueles com aptidão pobres receberão mais atenção. O número de clones criados, a partir de cada anticorpo, é calculado por meio da Equação (J.4).

$$numClones_i = \left(\frac{f_i}{\sum_{j=1}^{S} f_j} \right) (n\alpha). \tag{J.4}$$

em que f_i é a aptidão do antígeno em questão, n é o número de total de antígenos selecionados após o passo **Seleção e Remoção**, e α é um fator de escala opcional utilizado para aumentar ou diminuir o número de clones produzidos a cada geração. O valor padrão é 1.

- **Inserção**. Os clones gerados são inseridos na população principal de anticorpos. Um número n de antígenos selecionados aleatoriamente do conjunto de antígeno é inserido na população principal.

3. **Eliminação final**. A população de anticorpos é exposta à população de antígenos, a aptidão é calculada para cada anticorpo e a eliminação de anticorpos é realizada como descrito no passo **Seleção e Remoção**.

4. **Finalização**. A população final de anticorpos representa o classificador CSCA.

Para classificar novos objetos, os anticorpos de classificação são expostos a eles e os l anticorpos mais similares (com maior afinidade) são selecionados e o voto majoritário classifica o objeto.

J.1.4 Algoritmo Immunos

O sistema imunológico artificial IMMUNOS-81, proposto por [177], é um algoritmo para reconhecimento de padrão e classificação. Brownlee [80] melhorou este algoritmo por meio da incorporação de elementos de outros classificadores baseados na imunologia artificial, tais como o processo de seleção clonal e hipermutação, dando origem ao algoritmo IMMUNOS-99. Uma breve descrição do IMMUNOS-99 consiste nos seguintes passos:

1. **Inicialização**. A população de antígenos (Ag) é dividida em grupos Ag_i, com base no rótulo das classes. Então, o grupo Ag_i é constituído de antígenos da classes c_i.

2. **Treinamento dos anticorpos**. Para cada grupo de antígenos é preparada uma população de anticorpos (células B). Durante G gerações, os seguintes passos serão executados:

   - **Seleção**. Cada anticorpo é exposto ao grupo específico de antígenos, e a sua aptidão é calculada pela Equação (J.5), indicando assim o quanto os anticorpos identificam os antígenos, que pertencem à mesma classe.

   $$fitness = \frac{correta}{incorreta}, \tag{J.5}$$

   em que a pontuação *correta* é a soma das pontuações para os antígenos do mesmo grupo, e a pontuação *incorreta* é a soma das pontuações pontuação para os antígenos das outras classes. Assim, as células B que reconhecem melhor os antígenos da mesma classe tem sua pontuação de aptidão aumentada enquanto que as células B que reconhecem antígenos de outras classes tem sua aptidão diminuída, dada por:

   $$score = \frac{i}{\nu}, \tag{J.6}$$

   em que ν é o número total de anticorpos na população ordenados da mesma classe, e i é o índice do i-ésimo anticorpo da população.

   - **Eliminação**. Um parâmetro definido pelo usuário, $\mathcal{E} \in [0, 1]$, configura a pontuação de aptidão mínima de uma célula B. Todas as células B abaixo deste valor são removidas da população.

   - **Clonagem e Mutação**. Após a eliminação, a população de célula B contém somente células que podem identificar antígenos da mesma classe. Para melhorar a habilidade de reconhecimento da célula B, o sistema submete-se a um processo de maturação baseada na clonagem e hipermutação. Cada anticorpo é clonado proporcionalmente, com base na Equação (J.7).

   $$rr_i = \frac{rank}{S}, \tag{J.7}$$

   em que rr_i é a taxa de *rank* do i-ésimo anticorpo, $rank \in [1, S]$ é o índice atual do anticorpo na sequência ordenada, S é o número total de anticorpos ab_i na população da classe c_i.

   Para cada um dos anticorpos são criados um número de clones idênticos, com base na Equação (J.8).

   $$numClones_i = \left\lfloor \frac{r_i}{\sum_{j=1}^{S} r_j} N + 0.5 \right\rfloor. \tag{J.8}$$

Clones criados a partir de anticorpos com alto valor de taxa de *rank*, têm baixa taxa de mutação, em contraste com os anticorpos criados a partir de clones com baixo valor de taxa de *rank*, que são mais afetados pela mutação.

- **Inserção**. Um número n de antígenos são aleatoriamente selecionados a partir do conjunto de antígenos Ag e são inseridos na população de anticorpos principal Ab_i, em que n é o número de anticorpos eliminado no passo **Eliminação**.

3. **Eliminação Final**. Cada população de anticorpos é exposta a todos os antígenos e apenas o melhor anticorpo recebe uma pontuação. Similar ao passo **Eliminação**, todas as células com baixa pontuação de aptidão são removidas da população.

4. **Finalização**. A população que sobreviver à última eliminação representa o classificador IMMUNOS-99 para novos e desconhecidos antígenos.

Durante o processo de classificação, cada célula B associada a uma classe é exposta a um antígeno desconhecido e um índice de avidez é calculado. Então a população de célula B compete pelo antígeno desconhecido que recebe o rótulo da classe da população de células B com índice de avidez maior.

J.1.5 Algoritmo AIRS

O sistema de reconhecimento imunológico artificial (AIRS) é um algoritmo de aprendizagem supervisionada, que é usado para problemas de classificação [178]. A função do algoritmo AIRS, proposto por [79] é um processo de seleção clonal, inspirada na clonagem e hipermutação somática para a preparação de um banco de células de memória ou reconhecimento, que são representativos dos dados de treinamento. O algoritmo utiliza uma única iteração sobre um conjunto de dados de treinamento.

No algoritmo AIRS, uma célula B é chamada de esfera de reconhecimento artificial (ARB) que consiste em um anticorpo, um número de recursos (parâmetro que pode ser definido inicialmente pelo usuário) e um valor de estimulação (definido como a similaridade entre o ARB e o antígeno).

A população ARB é treinada durante vários ciclos de competição por recursos limitados. O melhor ARB recebe o maior número de recursos e os ARBs sem recursos são eliminados da população de células. A cada ciclo de treinamento, o melhor classificador ARB gera clones mutados que melhorem o processo de reconhecimento do antígeno, enquanto que os ARBs com recursos insuficientes são removidos da população. Após o treinamento, os melhores classificadores ARB são selecionados como células de memória. Estas células de memória são usadas para classificar novos antígenos.

Baseado em [179, 180], o ciclo de vida do sistema AIRS é descrito a seguir:

1. **Inicialização**. O sistema é preparado para o processo de aprendizagem. Os dados de treinamento são normalizados para $[0, 1]$. A afinidade é calculada para todos os pares de antígenos e então a referência inicial de afinidade é determinada como a afinidade média para todos os antígenos do conjunto de treinamento. Os antígenos são selecionados aleatoriamente e inseridos na população de células de memória.

   O *Limiar de Afinidade (AT)*, dado pela Equação (J.9), é determinado como uma média de afinidades para todos os pares de antígenos na base de dados de treinamento. Os valores de afinidade para a população de célula de memória são calculados pela interação com todos os antígenos de treinamento. As células de memória com maior estimulação são selecionadas para gerar os clones mutados que serão adicionados na população ARB.

$$AT = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} affinity(ag_i, ag_j)}{\frac{n(n-1)}{2}}, \tag{J.9}$$

   em que n é o número de antígenos ag na base de dados de treinamento. ag_i and ag_j são o i-ésimo e $j$ésimo antígeno de treinamento, e $affinity(x, y)$ retorna a distância Euclidiana entre dois antígenos.

2. **Antigen Training**. Cada instância de treinamento é apresentada à população de memória ($MC_{ag.c}$) de cada vez, em que MC representa o conjunto de células de memória e mc representa um membro individual deste conjunto e $ag.c$ representa a classe de um dado antígeno ag ($ag.c \in C = 1, 2, ...nc$ e nc é o número de classes na base de dados). O primeiro passo deste estágio do algoritmo é a identificação das células de memória e a geração de ARB. Dado um específico antígeno de treinamento (ag), encontra-se a célula de memória (mc_{match}) que tem a propriedade descrita na Equação (J.10). As células de memória com maior estímulo, $stim(ag, mc_{match})$, são selecionados para gerarem clones mutados, baseados na Equação (J.11), que são adicionados na população ARB.

$$mc_{match} = argmax_{mc \in MC_{ag*c}} stim(ag, mc). \tag{J.10}$$

$$nClones = stim(ag, mc_{match}) * cRate * hRate, \tag{J.11}$$

   em que $stim(x, y)$ é definido na Equação (J.12). a taxa clonal $(cRate)$ e a taxa de hipermutação $(hRate)$ são definidos pelo usuário.

$$stim(x, y) = 1 - affinity(x, y). \tag{J.12}$$

3. **Competição por Recursos Limitados**. Esta fase é usada para controlar o tamanho da população ARB, para desenvolver uma célula de memória, que é o candidato mais bem sucedido em classificar corretamente um dado antígeno (ag), e alocar, de forma otimizada,

recursos para os ARBs com as melhores capacidades de reconhecimento. Cada antígeno treina apenas as ARBs da mesma classe.

Primeiro, os recursos são alocados para um dado ARB (ab), baseado no seu valor de estímulo normalizado (ver Equação (J.13)), o qual é usado como uma indicação de sua aptidão em reconhecer ag.

$$resource = normStim(ag, ab) * cRate. \tag{J.13}$$

O número máximo de recursos que podem ser alocados pelo ARB é um parâmetro definido pelo usuário. A população ARB é então ordenada por recursos alocados na ordem descendente e os recursos são removidos dos ARBs situados no fim da lista até que a soma de todos os recursos alocados seja menor que o número total de recursos e, em seguida, os ARBs sem recursos são removidos da população de ARB. A competição para o processo de recursos finaliza quando a estimulação média normalizada é maior do que o limiar de estimulação definido pelo usuário. No passo final, cada ARB presente na população de ARB gera um número de clones mutados, usando a Equação (J.14).

$$ARBClones = stim(ag, ab) * cRate. \tag{J.14}$$

4. **Seleção de Células de Memória**. O estágio final do processo de treinamento é a inserção de células de memórias candidatas, $mc_{candidate}$, na população de memória existente (MC). É durante este estágio que o limiar de afinidade, calculado durante a inicialização torana-se crítico, uma vez que determina se o $mc_{candidate}$ substitui o mc_{match} que foi previamente identificado. Assim, o ARB é copiado para a população de células de memória se o valor de estímulo para o $mc_{candidate}$ é melhor do que a melhor célula de memória existente. Isso ocorre se a afinidade entre o $mc_{candidate}$ e o mc_{match} é menor que o produto entre o limiar de afinidade - AT e o limiar escalar de afinidade - ATS (ver Equação (J.15)).

$$cutOff = AT * ATS, \tag{J.15}$$

em que ATS é um parâmetro definido pelo usuário.

5. **Finalização**. O processo de treinamento é finalizado e o grupo de células de memória forma o classificador AIRS.

A classificação de antígenos novos é realizada através do método *k-nearest neighbor* [85]. As melhores l células de memória são identificadas e a classe predita é determinada com um voto majoritário. O parâmetro l pode ser otimizado para maximizar o desempenho da predição.

O algoritmo AIRS1, utilizado nos experimentos, é a primeira versão do algoritmo e realiza generalização via redução de dados. Isto significa que ele não utiliza todos os dados

de formação de generalização, e o classificador resultante produzido pelo algoritmo representa os dados de treinamento com um número reduzido ou mínimo de ocorrências. Outras versões foram desenvolvidas (AIRS2 e paralelo AIRS2), mas estes não foram testados neste trabalho, devido ao volume do conjunto de dados, que gera um elevado aumento de tempo de execução geral.

J.1.6 Resultados Obtidos

Nos experimentos realizados, foi utilizada a ferramenta WEKA (*Waikato Environment for Knowledge Analysis*) para testar o algoritmo de árvore de decisão *C4.5* [83] e os algoritmos *Clonal Selection Algorithm* (CLONALG) [77], *Clonal Selection Classification System* (CSCA) [78], o *Artificial Immune Recognition System* (AIRS) [79] e o *Artificial Immune System* (IMMUNOS-99) [80]. O algoritmo C4.5 *Release* 8 é implementado nesta ferramenta por meio do algoritmo J48.

Na Tabela J.2 são apresentados os resultados gerais obtidos no estudo comparativo entre o algoritmo de árvore de decisão C4.5 e os algoritmos *Clonal Selection Algorithm* (CLONALG) [77], *Clonal Selection Classification System* (CSCA) [78], o *Artificial Immune Recognition System* (AIRS) [79] e o *Artificial Immune System* (IMMUNOS-99) [80].

Um subconjunto de dados selecionados aleatoriamente da base de dados KDD Cup 99 [118] para o treinamento e teste dos algoritmos contém 2.351 registros de conexões indicando tráfego de rede normal e 20.302 registros contendo 22 tipos de ataques, os quais se enquadram em uma das quatro categorias principais de ataques (DoS, Probing, R2L e U2R). A distribuição dos tipos de ataques pode ser visualizada na Tabela J.1. Uma explicação sobre a base de dados KDD Cup 99 [118] pode ser encontrada no Capítulo 6.

Tabela J.1 Relação de ataques organizada em categorias.

DoS	PROBING	R2L	U2R
back (1026)	ipsweep (586)	ftp-write (8)	buffer-overflow (21)
land (11)	nmap (151)	guess-passwd (53)	loadmodule (10)
neptune (10401)	portsweep (155)	imap (11)	perl (3)
pod (69)	satan (16)	multihop (11)	rootkit (7)
smurf (7669)		phf (5)	
teardrop (15)		warezclient (60)	
		warezmaster (20)	
		spy (4)	

O método de validação cruzada com 10 *folds* foi utilizado nesse experimento para encontrar a estimativa de erro na classificação das instâncias de teste (ver Apêndice B).

O percentual de falso positivo foi calculado levando em consideração o número de erros de classificação quando o tráfego é normal e foi considerado como suspeito (independente do tipo de ataque envolvido) e no percentual de falso negativo foi considerado o número de erros

Tabela J.2 Resultado geral sobre comparativo entre os algoritmos de classificação.

Algoritmo	Acc	TFP	TFN	*kappa*
C4.5	99,64%	0,89%	1,45%	0,994
CLONALG	98,35%	4,04%	0,77%	0,975
CSCA	98,18%	4,17%	1,04%	0,972
AIRS1	98,21%	11,19%	0,27%	0,973
Immunos99	91,97%	67,97%	0,00%	0,879

Acc - Acurácia do Classificador, TFP - Taxa de Falso Positivo, Taxa de FN - Falso Negativo, Coeficiente *kappa.*

quando um tráfego suspeito foi considerado normal. O caso em que houve erro de classificação com relação ao tipo de ataque previsto foi avaliado somente como classificação incorreta.

A partir dos resultados obtidos neste experimento (ver Tabela J.2), que aponta árvore de decisão com melhor desempenho na classificação de tipos de ataques, surgiu a motivação para realizar um estudo comparativo do uso de medidas usadas na teoria da informação [163] aplicadas na construção de árvores de decisão C4.5.

J.2 Estudo Comparativo entre as Medidas de Informação de Shannon, Rényi e Tsallis na Construção de Árvores de Decisão C4.5

Os conceitos abordados no Capítulo 3 e Apêndice C servem de embasamento para esta seção, que tem como objetivo principal apresentar e analisar alguns resultados obtidos nos experimentos realizados, utilizando árvore de decisão, comparando os esquemas propostos com as medidas de informação de Shannon [84].

Como as entropias de Rényi [40] e Tsallis [42] oferecem diferentes compromissos entre a puridade dos nós e a informação mútua, buscou-se encontrar coeficientes ótimos para o problema de detectar intrusões em redes de computadores.

J.2.1 Resultados Obtidos

Nas Tabelas J.3, J.4 e J.5 são apresentados alguns dos resultados comparativos obtidos com os respectivos percentuais de instâncias classificadas corretamente (verdadeiros positivos), o percentual de falsos positivos (dados normais classificados erroneamente como suspeitos) e falsos negativos (dados suspeitos classificados erroneamente como normais), o índice *kappa*, o número de folhas e a profundidade da árvore para os valores de α testados. As medidas de desempenho ora utilizadas estão descritas no Apêndice B.

Foram realizados experimentos com a base de dados KDD Cup 99 [118], subdividida em categorias de ataques (DoS, Probing, R2L e U2R), conforme Tabela J.1. Outros experimentos foram realizados, inclusive utilizando a base de dados *NSL-KDD Data Set for Network-Based*

Tabela J.3 Resultado Geral usando as medidas de informação de Rényi.

α	Acurácia	Taxa de Falso Positivo	Taxa de Falso Negativo	Estatística *kappa*	Número de Folhas	Profundidade da Árvore
0,1	99,687%	0,7656%	0,1330%	0,9953	391	456
0,2	99,6601%	0,9783%	0,1281%	0,9949	453	517
0,3	99,6468%	1,0634%	0,1281%	0,9947	454	519
0,4	99,6204%	1,1910%	0,1428%	0,9943	273	339
0,5	99,691%	0,7656%	0,0837%	0,9953	189	244
0,6	99,6822%	0,7231%	0,0985%	0,9952	128	186
0,7	99,6424%	0,8507%	0,1478%	0,9946	259	313
0,8	99,691%	0,7656%	0,1133%	0,9953	380	434
0,9	99,607%	1,1910%	0,1822%	0,994	450	509

Tabela J.4 Resultado Geral usando as medidas de informação de Shannon.

-	Acurácia	Taxa de Falso Positivo	Taxa de Falso Negativo	Estatística *kappa*	Número de Folhas	Profundidade da Árvore
	99.638%	0.8932%	0,1675%	0,9945	533	593

Tabela J.5 Resultado Geral usando as medidas de informação de Tsallis.

α	Acurácia	Taxa de Falso Positivo	Taxa de Falso Negativo	Estatística *kappa*	Número de Folhas	Profundidade da Árvore
1,1	99,6248%	0,6806%	0,2217%	0,9943	393	443
1,2	99,6601%	0,8082%	0,1872%	0,9949	509	557
1,3	99,6292%	0,8932%	0,1872%	0,9944	571	617
1,4	99,638%	0,7231%	0,1872%	0,9945	572	619
1,5	99,638%	0,6806%	0,1872%	0,9945	571	617
1,6	99,691%	0,5104%	0,5530%	0,9953	170	219
1,7	99,7042%	0,5104%	0,0690%	0,9955	166	211
1,8	99,7131%	0,4679%	0,0690%	0,9957	167	213
1,9	99,6998%	0,5104%	0,0739%	0,9955	167	213
2	99,691%	0,5104%	0,0837%	0,9953	167	213
2,1	99,6866%	0,5104%	0,0837%	0,9953	167	213
3	99,6733%	0,5104%	0,0985%	0,9951	168	215

Intrusion Detection Systems [181, 182], também utilizada em testes de SDI. Os resultados obtidos nesses experimentos foram publicados em [16, 103] e encontram-se nos anexos desta tese.

Para o conjunto de treinamento e testes adotado nesta pesquisa (ver Tabela J.1), os melhores resultados experimentais (usando a validação cruzada 10-*folds*), dentre todos os algoritmos testados com seus respectivos valores de α e tomando como referência apenas a classificação correta das conexões normais ou suspeitas (independente do tipo de intrusão e se foram corretamente classificadas ou não pelo seu tipo específico) foram obtidos com a entropia de Tsallis, com apenas $0,2869\%$ de erro na classificação das instâncias testadas, com $0,4679\%$ de taxa de falso positivo e uma taxa de falso negativo na faixa de $0,0690\%$, com uma considerada redução no tamanho e número de folhas da árvore, para o valor de $\alpha = 1,8$. Além disso, das 65 instâncias classificadas erroneamente, 41 foram identificadas como suspeitas, o que pode ser considerado positivo em sistemas que apenas classificam conexões como normais ou suspeitas.

O uso da entropia de Rényi [40] também apresentou melhores resultados do que os testes realizados usando a entropia de Shannon [84], por exemplo, ao ser utilizado $\alpha = 0,5$. Entretanto, os resultados encontrados com as entropias de Rényi [40] e Tsallis [42] foram mais significativos considerando-se a redução no tamanho e número de folhas da árvore.

Um exemplo simplificado e meramente ilustrativo de árvore de decisão que pode ser gerado pelo modelos simulados, a partir do conjunto de instâncias ilustrado na Tabela J.1, é mostrado na Figura J.1. Nesse exemplo é possível visualizar um modelo aplicado na detecção de ataques do tipo **DOS**.

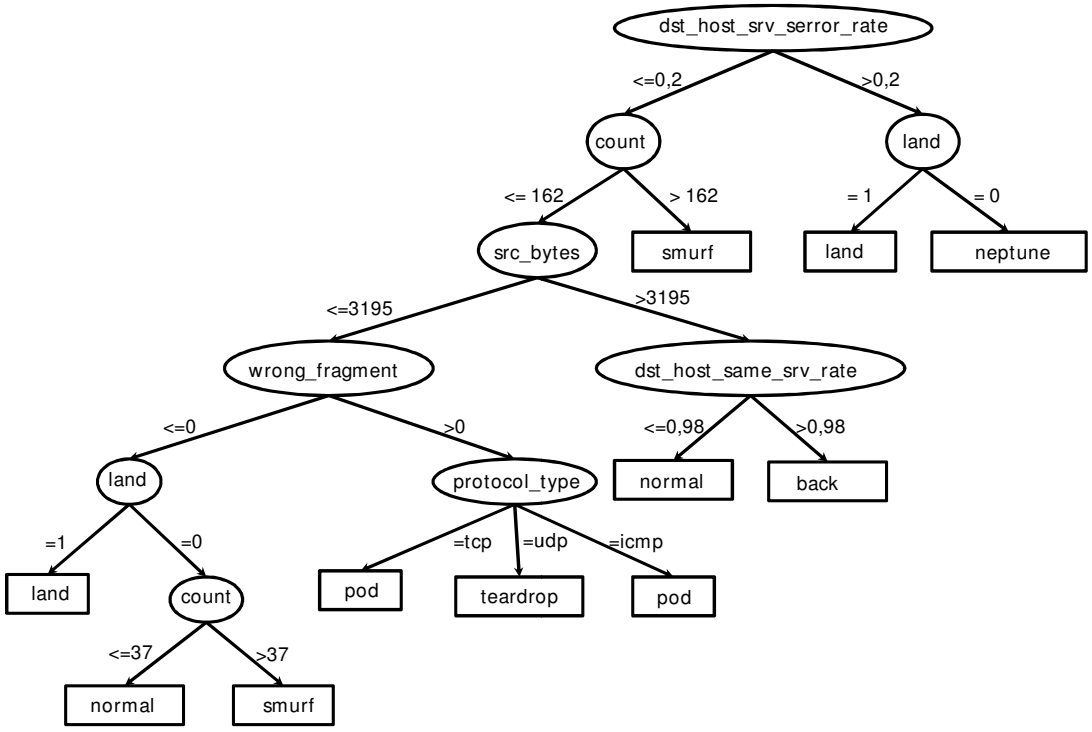


Figura J.1 Exemplo de árvore de decisão C4.5 aplicado na detecção de ataques da categoria DOS.

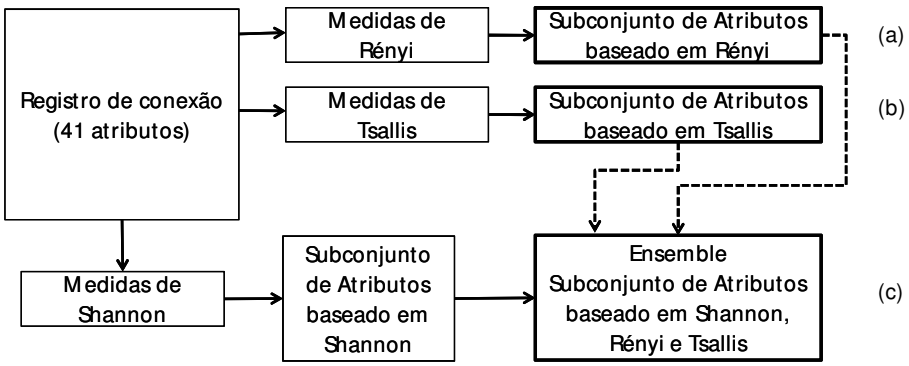
Com base nos tipos de ataques apresentados na Tabela J.1, é possível verificar, por exemplo, que o ataque do tipo *land* (ver explicação sobre este tipo de ataque no Apêndice H) foi modelado contendo dois caminhos diferentes a serem percorridos na árvore gerada. Isso significa que, para o mesmo ataque, pode-se ter diferentes assinaturas, gerando árvores com número de folhas superior ao número de classes. Assim, o número de nós da árvore depende do domínio de aplicação e da base de dados de treinamento.

J.3 Estudo Comparativo entre as Medidas de Informação de Shannon, Rényi e Tsallis Aplicadas na Seleção de Atributos

Como parte integrante desta tese, apresenta-se um estudo comparativo entre o esquema que utiliza árvore de decisão C4.5, com base nas medidas de informação de Shannon [83] e três esquemas de seleção de atributos, ilustrados na Figura J.2: (a) C4.5 baseada em medidas de Rényi [40]; (b) C4.5 baseada em Tsallis [42]; (c) um esquema que reúne os atributos selecionados pelas medidas de informação de Shannon [84], Rényi [40] and Tsallis [42]. Uma explicação sobre Seleção de Atributos pode ser encontrada no Capítulo 5.

Os esquemas utilizados neste experimento são ilustrados na Figura 5.8.

Figura J.2 Esquemas para selecionar atributos: (a) usando as medidas de Rényi; (b) usando as medidas de Tsallis; (c) abordagem *ensemble*.



A ideia de usar a abordagem *ensemble*, em que a informação de diferentes esquemas de seleção de atributos individuais são reunidos, é porque trata-se de uma abordagem que pode aumentar a capacidade de generalização de soluções baseadas em aprendizado de máquina e, portanto, fará parte do experimento apresentado.

J.3.1 Resultados Obtidos

Foram conduzidos quatro experimentos para investigar qual subconjunto de atributos é mais adequado para a detecção de uma determinada categoria de ataque. Considerando os resultados obtidos nos experimentos realizados e publicados em [16] (nos anexos deste trabalho), foi escolhido para ser analisado o melhor modelo de árvore de decisão C4.5 construído para cada categoria de ataque (DoS, Probing, R2L e U2R), segundo estimativas de desempenho Acc, TFN, TFP, índice *kappa* e profundidade da árvore (ver Apêndice B). Por exemplo, para a categoria DoS (negação de serviço), foi selecionado o modelo de árvore de decisão construído usando as medidas de Rényi com $\alpha = 0,5$, e pelas medidas de Tsallis com $\alpha = 1,2$.

Com base nos resultados apresentados na Tabela J.6, têm-se os atributos selecionados, descritos pelo seu número correspondente na base de dados (ver Tabela 6.1, Capítulo 5), pelos três esquemas propostos. Para efeito de comparação, o subconjunto de atributos selecionados, utilizando-se as medidas de informação de Shannon é também apresentado.

Tabela J.6 Atributos selecionados usando árvore de decisão baseada nas medidas de informação de Shannon, Rényi e Tsallis.

Categoria	Esquema	Atributos Selecionados
DoS	Shannon	2, 5, 7, 8, 23, 34, 36, 39
	Rényi	2, 5, 7, 8, 23, 32, 35, 36, 39
	Tsallis	2, 5, 7, 8, 23, 26, 34, 39
	Ensemble	2, 5, 7, 8, 23, 26, 32, 34, 35, 36, 39
Probing	Shannon	1, 2, 4, 5, 6, 23, 30, 33, 37, 38, 40
	Rényi	1, 2, 5, 6, 25, 30, 32, 33, 37, 38, 40
	Tsallis	1, 2, 4, 6, 23, 30, 31, 33, 37, 38, 40
	Ensemble	1, 2, 4, 5, 6, 23, 25, 30, 31, 32, 33, 37, 38, 40
R2L	Shannon	1, 3, 5, 6, 9, 10, 11, 17, 19, 22, 32, 33, 35
	Rényi	2, 5, 6, 10, 11, 12, 19, 33, 35, 37, 38, 39
	Tsallis	1, 3, 5, 6, 10, 11, 17, 19, 22, 37, 38
	Ensemble	1, 2, 3, 5, 6, 9, 10, 11, 12, 17, 19, 22, 32, 33, 35, 37, 38, 39
U2R	Shannon	13, 16, 17, 18, 32, 33
	Rényi	13, 18, 32, 33, 36
	Tsallis	13, 16, 18, 32, 33
	Ensemble	13, 16, 17, 18, 32, 33, 36

A fim de avaliar o desempenho da classificação sobre os subconjuntos de atributos selecionados, usando os quatro esquemas, foram utilizados três modelos de classificação: o algoritmo de seleção clonal (CLONALG) [77], o sistema de classificação clonal seleção (CSCA) [78] e o sistema de reconhecimento imunológico artificial (AIRS) [79].

Nos procedimentos experimentais, inicialmente os três modelos de classificação foram aplicados nos conjuntos de dados originais (contendo 41 atributos), para obter-se o desempenho de classificação. Os resultados desses algoritmos de classificação foram utilizados como valores de referência na comparação de desempenho dos mesmos classificadores quando utilizados os

atributos selecionados pelos três esquemas de seleção de atributos. Os critérios para avaliar o desempenho dos classificadores são o índice *kappa* e AUC (ver descrição destas medidas no Apêndice B). O resultado é apresentado na Tabela J.7.

Tabela J.7 Resultado Experimental.

Categoria	Esquema	41 atributos		Usando Shannon		Usando Rényi		Usado Tsallis		Usando ensemble	
		kappa	AUC	*kappa*	AUC	*kappa*	AUC	*kappa*	AUC	*kappa*	AUC
DoS	AIRS1	0.9745	0.959	0.9195	0.944	0.9209	0.868	0.9234	0.972	0.949	0.916
	Clonalg	0.9929	0.9875	0.9943	0.995	0.9919	0.9904	0.9943	0.995	0.9916	0.9914
	CSCA	0.9948	0.9935	0.9956	0.994	0.9946	0.993	0.9954	0.995	0.9948	0.9915
Probing	AIRS1	0.9384	0.9775	0.9056	0.959	0.922	0.9655	0.9361	0.9735	0.9144	0.9795
	Clonalg	0.9461	0.9855	0.9409	0.985	0.9344	0.9845	0.874	0.940	0.9388	0.9845
	CSCA	0.9195	0.961	0.9138	0.952	0.9009	0.9485	0.883	0.9275	0.898	0.9455
R2L	AIRS1	0.8885	0.9825	0.8612	0.949	0.8602	0.9495	0.3085	0.846	0.8735	0.9655
	Clonalg	0.9116	0.9425	0.9119	0.942	0.9051	0.9425	0.9116	0.939	0.9121	0.945
	CSCA	0.8755	0.9155	0.8829	0.928	0.8867	0.948	0.8744	0.9235	0.8833	0.9275
U2R	AIRS1	0.7667	0.865	0.7331	0.9	0.7002	0.876	0.6741	0.9015	0.7338	0.961
	Clonalg	0.7857	0.841	0.8699	0.9735	0.8789	0.962	0.8803	0.974	0.8699	0.9735
	CSCA	0.7682	0.829	0.8547	0.962	0.8512	0.95	0.8285	0.9495	0.8699	0.9865

Os experimentos foram realizados utilizando a validação cruzada 10-*folds* para controlar a validade dos mesmos. Os resultados experimentais foram obtidos, considerando a base de dados descrita na Tabela 6.2, Capítulo 6.

Na análise experimental sobre o desempenho dos esquemas de seleção de atributos, os resultados são significativamente diferentes, se forem maiores ou iguais a 1%. Além disso, os resultados variam de acordo com os classificadores e a métrica de desempenho utilizada para avaliar os modelos.

A partir da Tabela J.7, os resultados de detecção, no conjunto de dados KDD 99, indicam que o desempenho permanece quase o mesmo ou até torna-se melhor para os classificadores CLONALG e CSCA para as categorias DoS, R2L e U2R por qualquer esquema de seleção de atributos em comparação com o uso do conjunto de dados completo (com 41 atributos).

Em particular, as medidas de Tsallis para seleção de atributos não trazem qualquer melhoria no desempenho (ver valores *kappa* e AUC na Tabela J.7) para os algoritmos CLONALG e CSCA na categoria *Probing*, e AIRS1 na categorias R2L. Contudo, obteve o melhor desempenho quando utilizado o algoritmo CLONALG na categoria U2R.

Para as categorias de ataques DoS, R2L e U2R sobre o algoritmo AIRS1, o desempenho da classificação, em termos de índice *kappa* e considerando os atributos selecionados pelos quatro esquemas de seleção de atributos, obteve os resultados significativamente piores em comparação com o conjunto de dados contendo 41 atributos.

Os resultados menos significativos, foram para as categorias do tipo U2R. Isto é consistente com os resultados de muitas pesquisas, pois o padrão das assinaturas dos ataques da categoria U2R é muito parecido com o padrão das operações normais do sistema.

J.4 Resultados obtidos nos experimentos realizados com AG baseado em códigos BCH

No experimento realizado foi utilizado um AG para encontrar as palavras código que melhor se adequassem ao problema proposto. A tabela de palavras código gerada pelo AG, nesse experimento, permite que mais de uma palavra código seja associada a cada classe. Em todas as buscas realizadas pelo referido algoritmo, utilizou-se uma taxa de cruzamento de $0,6$ e uma taxa de mutação de $0,001$. A condição de parada foi o número máximo de 50 iterações. O procedimento de codificação e decodificação do código BCH implementado nesse algoritmo é uma versão modificada do código fonte bch3.c (em linguagem de programação C) disponibilizado por [133].

Optou-se por utilizar, nesse experimento, um código BCH primitivo. Portanto, foram feitas simulações com AG, usando os códigos $BCH(31, 16, 7)$ e $BCH(31, 21, 5)$. Para realizar a avaliação de cada indivíduo, foi utilizada a função objetivo representada na Equação 5.1, Capítulo 5. Verificou-se o poder preditivo das palavras código selecionadas, na tarefa de detectar tipos de ataques, bem como identificar o tráfego normal numa rede de computadores.

Para obter os resultados, foi utilizada uma base de dados para cada categoria de ataque, extraída da base de dados KDD Cup 99. O número de instâncias por tipo de ataque, bem como as instâncias contendo tráfego normal estão relacionados na Tabela 6.2, Capítulo 6.

Como os códigos BCH não são capazes de processar dados em todos os formatos e os atributos na base de dados KDD Cup 99 [118] contêm dados de diversos tipos, foi necessário realizar uma etapa de pré-processamento para transformar os dados em um formato aceitável pelo código. Assim, os atributos contínuos foram discretizados, utilizando-se os algoritmos para discretização de atributos *Minimum Description Lenght* (MDL), proposto por Fayyad e Irani [148], o qual usa a informação mútua de Shannon [84] como critério para encontrar o ponto onde há alternância de classe.

Após o processo de discretização, os atributos foram mapeados para valores binários. Por fim, foi investigada a importância desses atributos em relação a cada tipo de ataque. Então, com base nos resultados obtidos usando árvore de decisão C4.5 baseada em medidas de informação de Rényi e Tsallis, os atributos mais relevantes foram determinados, num total de 31 atributos binários. Esses atributos serão observados pelos detectores, cujas regras de decisão são utilizados para compor as palavras recebidas.

Em relação à fase de decodificação, levando em consideração que o número de classes ou tipos de ataques em uma rede de computadores pode ser bastante elevado, foi utilizado nesse experimento o algoritmo de decodificação algébrica [24], baseado no algoritmo de *Berlekamp-Massey* (BMA) [133] descrito no Apêndice D.

Conforme ilustrado nas Tabelas J.8, J.9, J.10 e J.11, utilizou-se nesse experimento o algoritmo de árvore de decisão C4.5 padrão como algoritmo para soluções multiclasses com o

objetivo de compará-lo com a estratégia proposta. Assim, dados os 31 atributos selecionados, estes foram utilizados como entradas nos três algoritmos avaliados: C4.5, $BCH(31, 16, 7)$ e $BCH(31, 21, 5)$.

Na avaliação dos resultados pode-se observar que a versão de AG para a determinação de tabelas de palavras código foi capaz de gerar soluções com bom desempenho, com percentual de classificação correta (VP) e precisão de até 100% para alguns tipos de ataques, tais como ataques do tipo *buffer-overflow*, além de ataques de negação de serviço, cuja importância se dá ao fato desses ataques serem responsáveis por ocasionar um elevado índice de incidentes de intrusão.

Tabela J.8 Resultado para categoria DoS

Classe	**C4.5**		**BCH(31, 16, 7)**		**BCH(31, 21, 5)**	
	VP	**Precisão**	**VP**	**Precisão**	**VP**	**Precisão**
back	0,996	0,999	0,993	1	0,998	0,994
land	0,636	0,875	0,909	1	0,727	1
neptune	1	1	0,977	0,999	0,999	0,997
pod	1	1	1	1	1	1
smurf	1	1	1	1	1	1
teardrop	1	1	0,867	1	1	1
Normal	0,999	0,998	0,813	1	0,544	0,997

Tabela J.9 Resultado para categoria Probing

Classe	**C4.5**		**BCH(31, 16, 7)**		**BCH(31, 21, 5)**	
	VP	**Precisão**	**VP**	**Precisão**	**VP**	**Precisão**
ipsweep	0,998	1	1	0,983	1	1
nmap	0,993	0,987	0,881	1	1	0,981
portsweep	0,987	0,987	0,826	0,970	1	1
satan	0,750	0,923	0,750	0,923	1	0,889
spy	1	1	1	0,121	1	1
Normal	1	0,998	0,377	1	0,750	1

Tabela J.10 Resultado para categoria R2L

Classe	**C4.5**		**BCH(31, 16, 7)**		**BCH(31, 21, 5)**	
	VP	**Precisão**	**VP**	**Precisão**	**VP**	**Precisão**
ftp	0,250	1	0,50	1	1	1
guess-pwd	0,981	1	0,943	1	1	1
imap	0,818	1	0,091	1	0,909	1
multihop	0,727	0,667	0,818	1	1	1
phf	0	0	0	0	0	0
warezclient	1	0,966	0,817	1	0,967	1
warezmaster	1	1	1	1	1	1
Normal	0,996	0,991	0,131	0,944	0,993	0,997

Tabela J.11 Resultado para categoria U2R

Classe	C4.5		BCH(31, 16, 7)		BCH(31, 21, 5)	
	VP	**Precisão**	**VP**	**Precisão**	**VP**	**Precisão**
buffer_overflow	1	0,913	1	1	1	1
loadmodule	0,4	0,667	0,90	1	1	1
perl	1	1	1	1	1	1
rootkit	0,741	0,833	0,571	1	1	1
Normal	0,998	0,996	0,955	1	0,993	1

Embora geralmente as duas estruturas de código BCH tenham alcançado resultados semelhantes, em alguns casos verificou-se uma maior adequabilidade de uma sobre a outra. Confirmou-se nos experimentos que o algoritmo C4.5, baseado em medidas de informação de Rényi [40] e Tsallis [41, 42] pode auxiliar na escolha dos atributos a serem monitorados pelos sensores.

APÊNDICE K

Código Fonte da Implementação das Entropias de Rényi e Tsallis no Weka

Neste apêndice, tem-se a listagem dos códigos fontes usados nas simulações com árvore de decisão C4.5, modificadas para trabalharem com as entropias de Rényi e Tsallis.

K.1 Código fonte com a entropia de Rényi implementada

```
/*
*    This program is free software; you can redistribute it and/or modify
*    it under the terms of the GNU General Public License as published by
*    the Free Software Foundation; either version 2 of the License, or
*    (at your option) any later version.
*
*    This program is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*    GNU General Public License for more details.
*
*    You should have received a copy of the GNU General Public License
*    along with this program; if not, write to the Free Software
*    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

/*
*    EntropyBasedSplitCrit.java
*    Copyright (C) 1999 University of Waikato, Hamilton, New Zealand
*
*/

   package weka.classifiers.trees.j48;
    /**
     * "Abstract" class for computing splitting criteria
     * based on the entropy of a class distribution.
     *
     * @author Eibe Frank (eibe@cs.waikato.ac.nz)
     * @version $Revision: 1.7 $
     */
    public abstract class EntropyBasedSplitCrit extends SplitCriterion {
```

```
      /** for serialization */
      private static final long serialVersionUID = -26189914397916530 56L;

      /** The log of 2. */
      protected static double log2 = Math.log(2);
      protected static double alfa = 0.7;

      /**
      * Help method for computing entropy.
      */
       public final double logFunc(double num) {

          // Constant hard coded for efficiency reasons
          if (num < 1e-6)
             return 0;
          else
             if (alfa!=1.0)
              return (Math.log(num) / log2);
              else
                return num * Math.log(num) / log2;
        }

        /**
        * Computes entropy of distribution before splitting.
        */
        public final double oldEnt(Distribution bags) {
            double num=0;
            double returnValue = 0;
            int j;

            for (j = 0; j < bags.numClasses(); j++)
               if (alfa==1.0)
                   returnValue = returnValue + logFunc(bags.perClass(j));
                else
                num=num+Math.pow(bags.perClass(j),alfa);
               if(alfa==1.0)
                 return logFunc(bags.total()) - returnValue;
               else{
                returnValue = returnValue + logFunc(num);
                return  (1/(1-alfa))*(returnValue -
logFunc(Math.pow(bags.total(), alfa)));
              }
        }

       /**
       * Computes entropy of distribution after splitting.
       */
       public final double newEnt(Distribution bags) {
           double num1 = 0;
           double num2 = 0;
            double returnValue = 0;
            int i, j;

            for (i = 0; i < bags.numBags(); i++) {
                num1=0;
                for (j = 0; j < bags.numClasses(); j++)
                    if (alfa==1.0)
                         returnValue = returnValue
```

```
                        + logFunc(bags.perClassPerBag(i, j));
                    else
                     num1 = num1+Math.pow(bags.perClassPerBag(i, j),alfa);
                    if(alfa==1.0){
                        returnValue = returnValue - logFunc(bags.perBag(i));
                    }
                    else{
                       num2 = num2+(bags.perBag(i)/(1-alfa))*(logFunc(num1) -
 logFunc(Math.pow(bags.perBag(i),alfa)));
                      }
                }
                if (alfa!=1.0){
                   return num2/bags.total();
                }
                else
                    return -returnValue;
            }

            /**
             * Computes entropy after splitting without considering the
             * class values.
             */
            public final double splitEnt(Distribution bags) {

                double returnValue = 0;
                double num = 0;
                int i;

                for (i = 0; i < bags.numBags(); i++)
                 if (alfa==1.0)
                        returnValue = returnValue + logFunc(bags.perBag(i));
                 else
                 num = num + Math.pow(bags.perBag(i), alfa);
                    if (alfa == 1.0)
                        return logFunc(bags.total()) - returnValue;
                    else{
                     returnValue = returnValue + logFunc(num);
                     return (1/(1-alfa))*(returnValue-
logFunc(Math.pow(bags.total(), alfa)));
                    }
            }
        }


/*
*    This program is free software; you can redistribute it and/or modify
*    it under the terms of the GNU General Public License as published by
*    the Free Software Foundation; either version 2 of the License, or
*    (at your option) any later version.
*
*    This program is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*    GNU General Public License for more details.
*
*    You should have received a copy of the GNU General Public License
*    along with this program; if not, write to the Free Software
*    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/
```

```
/*
*    GainRatioSplitCrit.java
*    Copyright (C) 1999 University of Waikato, Hamilton, New Zealand
*
*/

 package weka.classifiers.trees.j48;

 import weka.core.Utils;

 /**
 * Class for computing the gain ratio for a given distribution.
 *
 * @author Eibe Frank (eibe@cs.waikato.ac.nz)
 * @version $Revision: 1.7 $
 */
    public final class GainRatioSplitCrit extends EntropyBasedSplitCrit {

    /** for serialization */
    private static final long serialVersionUID = -433336694718670930L;
    /**
    * This method is a straightforward implementation of the gain
    * ratio criterion for the given distribution.
    */
    public final double splitCritValue(Distribution bags) {

          double numerator;
          double denumerator;
          numerator = oldEnt(bags) - newEnt(bags);

         // Splits with no gain are useless.
         if (Utils.eq(numerator, 0))
              return Double.MAX_VALUE;
         denumerator = splitEnt(bags);

         // Test if split is trivial.
         if (Utils.eq(denumerator, 0))
         return Double.MAX_VALUE;

         //  We take the reciprocal value because we want to minimize the
         // splitting criterion's value.
          return denumerator / numerator;
         }

        /**
       * This method computes the gain ratio in the same way C4.5 does.
       *
       * @param bags the distribution
       * @param totalnoInst the weight of ALL instances
       * @param numerator the info gain
       */
          public final double splitCritValue(Distribution bags,
                 double totalnoInst, double numerator) {
                 double denumerator;
                 double noUnknown;
                 double unknownRate;
                 int i;

                 // Compute split info.
```

```
            denumerator = splitEnt(bags, totalnoInst);

            // Test if split is trivial.
            if (Utils.eq(denumerator, 0))
                return 0;
            return numerator / denumerator;
        }

        /**
         * Help method for computing the split entropy.
         */
        private final double splitEnt(Distribution bags, double totalnoInst) {

            double returnValue = 0;
            double noUnknown;
            double alfa=0.7;
            double num=0;
            int i;

            noUnknown = totalnoInst - bags.total();
            if (Utils.gr(bags.total(), 0)) {
                for (i = 0; i < bags.numBags(); i++)
                 if (alfa==1)
                 returnValue = returnValue - logFunc(bags.perBag(i));
                 else{
                 num=num+Math.pow(bags.perBag(i),alfa);
                 }
                if(alfa!=1)
                 returnValue = logFunc(num)- returnValue;
                returnValue = returnValue + logFunc(Math.pow(noUnknown, alfa));
                returnValue = returnValue - logFunc(Math.pow(totalnoInst, alfa));
            }
            return (1/(1-alfa)*returnValue);
        }
    }
```

K.2 Código fonte com a entropia de Tsallis implementada

```
/*
*    This program is free software; you can redistribute it and/or modify
*    it under the terms of the GNU General Public License as published by
*    the Free Software Foundation; either version 2 of the License, or
*    (at your option) any later version.
*
*    This program is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*    GNU General Public License for more details.
*
*    You should have received a copy of the GNU General Public License
*    along with this program; if not, write to the Free Software
*    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

/*
*    EntropyBasedSplitCrit.java
*    Copyright (C) 1999 University of Waikato, Hamilton, New Zealand
```

```
*
*/

package weka.classifiers.trees.j48;
/**
* "Abstract" class for computing splitting criteria
* based on the entropy of a class distribution.
*
* @author Eibe Frank (eibe@cs.waikato.ac.nz)
 * @version $Revision: 1.7 $
 */
   public abstract class EntropyBasedSplitCrit extends SplitCriterion {

      /** for serialization */
      private static final long serialVersionUID = -2618691439791653056L;

     /** The log of 2. */
        protected static double log2 = Math.log(2);
        protected static double alfa = 2;


            /**
             * Help method for computing entropy.
             */
        public final double logFunc(double num) {

        // Constant hard coded for efficiency reasons
             if (num < 1e-6)
                  return 0;
              else
                 if (alfa!=1.0)
                 return (Math.log(num) / log2);
                 else
                 return num * Math.log(num) / log2;
            }

 /**
 * Computes entropy of distribution before splitting.
 */
    public final double oldEnt(Distribution bags) {
          double num=0;
           double returnValue = 0;
           int j;

           for (j = 0; j < bags.numClasses(); j++)
               if (alfa==1.0)
                    returnValue = returnValue + logFunc(bags.perClass(j));
               else
                 num=num+
(Math.pow(bags.perClass(j),alfa)/Math.pow(bags.total(), alfa));
               if(alfa==1.0)
                 return logFunc(bags.total()) - returnValue;
               else{
                 returnValue = returnValue + (1-num);
                 return  (1/(alfa-1))*returnValue;
                }
            }

  /**
```

```
  * Computes entropy of distribution after splitting.
  */
     public final double newEnt(Distribution bags) {
          double num1 = 0;
           double num2 = 0;
           double returnValue = 0;
           int i, j;

           for (i = 0; i < bags.numBags(); i++) {
              num1=0;
               for (j = 0; j < bags.numClasses(); j++)
                 if (alfa==1.0)
                       returnValue = returnValue
                       + logFunc(bags.perClassPerBag(i, j));
                 else{
                    if (bags.perBag(i)!=0)
                      num1 = num1+
(Math.pow(bags.perClassPerBag(i, j),alfa)/Math.pow(bags.perBag(i), alfa));
                     }
                  if(alfa==1.0){
                         returnValue = returnValue - logFunc(bags.perBag(i));
                     }
                  else{
                         num2 = num2+((bags.perBag(i)/bags.total())*(1/(alfa-1))*(1-num1));
                        }
                 }
                 if (alfa!=1.0){
                    return num2;
                 }
                 else
                     return -returnValue;
            }

/**
* Computes entropy after splitting without considering the
* class values.
*/
     public final double splitEnt(Distribution bags) {

          double returnValue = 0;
          double num = 0;
          int i;

          for (i = 0; i < bags.numBags(); i++)
                if (alfa==1.0)
                         returnValue = returnValue + logFunc(bags.perBag(i));
                 else
                  num = num + Math.pow(bags.perBag(i), alfa);
                 if (alfa == 1.0)
                         return logFunc(bags.total()) - returnValue;
                 else{
                       returnValue = returnValue + logFunc(num);
                       return (1/(1-alfa))*
(returnValue-logFunc(Math.pow(bags.total(), alfa)));
                     }
            }
        }

/*
```

```
*    This program is free software; you can redistribute it and/or modify
*    it under the terms of the GNU General Public License as published by
*    the Free Software Foundation; either version 2 of the License, or
*    (at your option) any later version.
*
*    This program is distributed in the hope that it will be useful,
*    but WITHOUT ANY WARRANTY; without even the implied warranty of
*    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*    GNU General Public License for more details.
*
*    You should have received a copy of the GNU General Public License
*    along with this program; if not, write to the Free Software
*    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/
/*
*    GainRatioSplitCrit.java
*    Copyright (C) 1999 University of Waikato, Hamilton, New Zealand
*
*/

    package weka.classifiers.trees.j48;

    import weka.core.Utils;

/**
* Class for computing the gain ratio for a given distribution.
*
* @author Eibe Frank (eibe@cs.waikato.ac.nz)
* @version $Revision: 1.7 $
*/
 public final class GainRatioSplitCrit extends EntropyBasedSplitCrit {

     /** for serialization */
     private static final long serialVersionUID = -433336694718670930L;

     /**
    * This method is a straightforward implementation of the gain
    * ratio criterion for the given distribution.
    */
    public final double splitCritValue(Distribution bags) {
           double numerator;
           double denumerator;
           numerator = oldEnt(bags) - newEnt(bags);

          // Splits with no gain are useless.
          if (Utils.eq(numerator, 0))
                return Double.MAX_VALUE;
          denumerator = splitEnt(bags);

          // Test if split is trivial.
          if (Utils.eq(denumerator, 0))
              return Double.MAX_VALUE;

         //  We take the reciprocal value because we want to minimize the
        // splitting criterion's value.
                return denumerator / numerator;
           }

 /**
```

```
* This method computes the gain ratio in the same way C4.5 does.
*
* @param bags the distribution
* @param totalnoInst the weight of ALL instances
* @param numerator the info gain
*/
     public final double splitCritValue(Distribution bags,
          double totalnoInst, double numerator) {

          double denumerator;
          double noUnknown;
          double unknownRate;
          int i;

         // Compute split info.
          denumerator = splitEnt(bags, totalnoInst);

        // Test if split is trivial.
        if (Utils.eq(denumerator, 0))
               return 0;
        return numerator / denumerator;

        }

/**
* Help method for computing the split entropy.
*/
    private final double splitEnt(Distribution bags, double totalnoInst) {

         double returnValue = 0;
         double noUnknown;
         double alfa=2;
         double num=0;
         int i;

         noUnknown = totalnoInst - bags.total();
         if (Utils.gr(bags.total(), 0)) {
              for (i = 0; i < bags.numBags(); i++)
                if (alfa==1)
                 returnValue = returnValue - logFunc(bags.perBag(i));
                else{
                  num=num+(Math.pow((bags.perBag(i)/bags.total()),alfa));
                    }
                if(alfa!=1)
                    returnValue = (1-num)- returnValue;
                }
                return (1/(alfa-1)*returnValue);
            }
        }
```

ANEXOS