UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Nícolas Naves Rezende Faria

Microserviços para Geração de RCLs no GRASP-FS: Uma Abordagem Escalável para Seleção de Atributos em IDSs Industriais

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Nícolas Naves Rezende Faria

Microserviços para Geração de RCLs no GRASP-FS: Uma Abordagem Escalável para Seleção de Atributos em IDSs Industriais

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Silvio E. Quincozes

Coorientador: Prof. Dr. Juliano Fontoura Kazienko

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2023

Nícolas Naves Rezende Faria

Microserviços para Geração de RCLs no GRASP-FS: Uma Abordagem Escalável para Seleção de Atributos em IDSs Industriais

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 01 de novembro de 2023:

Prof. Dr. Silvio E. Quincozes
Orientador

Rodrigo Sanches Miani

Vagner Ereno Quincozes

Uberlândia, Brasil
2023

Agradecimentos

Caro Silvio Quincozes,

Gostaria de expressar minha sincera gratidão por toda a orientação e apoio que você me proporcionou ao longo da minha jornada de pesquisa. Sua experiência e conhecimento foram fundamentais para o sucesso do meu trabalho de conclusão de curso. Agradeço por dedicar seu tempo e energia para me orientar, compartilhando suas ideias e *feedback* valiosos. Sua orientação cuidadosa e sua disposição para discutir minhas dúvidas e preocupações contribuíram imensamente para o desenvolvimento do meu projeto. Sou verdadeiramente grato por sua paciência e compromisso em me ajudar a alcançar os melhores resultados possíveis.

Gostaria também de agradecer ao meu coorientador, Juliano Kazienko, pela contribuição valiosa ao meu TCC. Suas orientações adicionais e apoio têm sido cruciais para a conclusão bem-sucedida do meu trabalho. Suas perspectivas únicas e visões acadêmicas enriqueceram meu projeto, e sou grato pela oportunidade de aprender com você.

À minha amada família, minha mãe Francisnéia Naves, meu pai Alexandre Miranda, e meus irmãos Alexandre Naves e Vinícius Naves, quero transmitir todo o meu apreço. O apoio constante e o incentivo que vocês me proporcionaram ao longo dessa jornada foram verdadeiramente inestimáveis. Seu amor incondicional, compreensão e paciência foram os pilares que me impulsionaram até a conclusão do meu TCC. Sou imensamente grato por terem estado ao meu lado, oferecendo um apoio incondicional em todos os momentos.

Aos meus amigos, agradeço por serem uma fonte constante de encorajamento e apoio emocional. Suas palavras de ânimo e os momentos de descontração foram essenciais para manter minha motivação durante essa fase desafiadora. Sou grato por ter amigos tão incríveis ao meu lado.

Também gostaria de expressar minha sincera gratidão a Júlia Rangel. Sua presença durante essa fase de graduação foi extremamente significativa. Seu amor, apoio, compreensão e encorajamento foram essenciais para minha jornada acadêmica. Atravessamos desafios juntos e guardarei com carinho as lembranças dos momentos felizes que compartilhamos. Sou imensamente grato por ter tido você ao meu lado nessa etapa importante da minha formação.

A todos vocês, minha gratidão é imensa. Sem a ajuda, o amor e o apoio de cada um, eu não teria conseguido alcançar este marco em minha vida acadêmica.

Com profunda gratidão, Nícolas Naves Rezende Faria.

Resumo

Este trabalho apresenta uma arquitetura escalável orientada a microsserviços, chamada *Distributed RCL Generator* (DRG), para desacoplar e paralelizar a fase de construção na metaheurística *Greedy Randomized Adaptive Search Procedure for Feature Selection* (GRASP-FS) na Seleção de *Feature* (FS) para detecção de intrusões. No GRASP-FS, são gerados conjuntos de atributos que são otimizadas na sua fase de busca local. Através de uma prova de conceito baseada no *framework* Kafka e quatro algoritmos de FS, demonstrou-se que os algoritmos usados na estratégia de construção do RCL impactam na F1-Score média da detecção de intrusões em um cenário de Sistemas Ciber-Físicos, do inglês, Cyber-Physical Systems (CPSs), de 50,47% até 84,92%. Ademais, constatou-se que o processamento paralelo pode acelerar a fase de construção em cerca de 3,4 vezes.

Palavras-chave: GRASP-FS, *Feature*, Atributos, *Selection*, Seleção, Microsserviço, Cibersegurança, CPSs.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CPSs	*Cyber-Physical Systems*
DRG	*Distributed RCL Generator*
GRASP-FS	*Greedy Randomized Adaptive Search Procedure for Feature Selection.*
FS	*Feature Selection*
RCL	*Restricted Candidate List*
Fig.	Figura
IDSs	*Intrusion Detection Systems*
JARs	Executáveis de Java
IG	*Information Gain*
GR	*Gain Ratio*
SU	*Symetrical Uncertainty*
RF	*RelieF*
SFS	*Sequential Forward Selection*
SFE	*Sequential Backward Elimination*
SFFS	*Sequential Forward Floating Selection*
IWSS	*Incremental Wrapper Subset Selection*
FCBF	*Fast Correlation-Based Filter*
CFS	*Correlation-Based Feature Selection*
KNN	*MLK-Nearest Neighbors*
eBPF	*Extended Berkeley Packet Filter*
ERENO	*Efficacious Reproducer Engine for Network Operations*
IEDs	*Intelligent Electronic Devices*
FP	Falsos Positivos

FN	Falsos Negativos
VP	Verdadeiros Positivos
API	*Application Programming Interface*
GOOSE	*Generic Object Oriented Substation Events*
SV	*Sampled Values*
IEC	*International Electrotechnical Commission*
IEDs	*Intelligent Eletronic Devices*
MUs	*Measurement Units*

Sumário

1 Introdução

Os Sistemas Ciber-Físicos, do inglês, *Cyber-Physical Systems* (CPSs), que integram o mundo cibernético e componentes do mundo físico, são a base para o avanço de múltiplos setores como o de saúde, cidades inteligentes, além daqueles com infraestrutura crítica como é o caso da produção e distribuição de energia. Nesse cenário — que está em rápido crescimento — a importância da cibersegurança vem aumentando de forma constante (QUINCOZES et al., 2021). De acordo com os dados levantados por (FortiGuard Labs, 2021), em 2021 o Brasil ocupou o segundo lugar em número de ataques na América Latina e Caribe. Foram registrados 289 bilhões de ataques, representando um crescimento de mais de 600% com relação ao ano anterior, onde foram registrados 41 bilhões de ataques. Ademais, guerras também são travadas no campo virtual. Um ataque cibernético russo atingiu instalações elétricas ucranianas em 2022 (CARVALHO et al., 2022). Portanto, a adoção de soluções de segurança, como aquelas que analisam os atributos (do inglês, *features*) que permitem a detecção de ameaças é fundamental. A metaheurística *Greedy Randomized Adaptive Search Procedure for Feature Selection* (GRASP-FS) (QUINCOZES et al., 2021) é utilizada em *Intrusion Detection Systems* (IDSs) para resolver o problema de otimização combinatória da seleção de atributoss. A seleção de atributos representativos é essencial para que os IDSs tenham um bom desempenho na detecção de atividades maliciosas.

O GRASP-FS é composto por duas etapas iterativas: (i) a etapa de construção e (ii) a etapa de busca local. Para cada iteração do GRASP-FS, uma solução diferente é gerada na etapa de construção e otimizada posteriormente na etapa de busca local. Na etapa de construção, há uma redução inicial do número de atributos que irão compor as soluções candidatas. Nesse processo, é construída uma Lista Restrita de Candidatos, do inglês, *Restricted Candidate List* (RCL). A RCL consiste em um conjunto de atributos que atendem determinado critério. Os atributos que não atenderem o critério estabelecido são descartados, evitando-se sobrecarga na etapa de busca local. Tal redução se dá através de métodos que estimam o potencial de cada atributo para a composição de uma solução viável, isto é, um conjunto reduzido com atributos representativos para descrever comportamentos maliciosos. Uma vez construída a RCL, são selecionados subconjuntos de atributos dentre os candidatos para serem otimizados na etapa de busca local. Portanto, o desempenho das soluções exploradas na etapa de busca local dependem diretamente da qualidade dos atributos disponíveis na RCL construída na etapa de construção (QUINCOZES et al., 2021)(QUINCOZES et al., 2022).

1.1 Justificativa

De modo a gerar uma RCL de boa qualidade, é importante que se considerem diferentes alternativas de algoritmos para tal propósito. No entanto, a qualidade de uma RCL depende dos padrões de tráfego no momento de sua geração. Uma vez que hajam mudanças em tais padrões, novas RCLs precisam ser geradas — eventualmente, um algoritmo diferente daquele usado na geração da RCL anterior pode se tornar mais promissor. Em ambientes tais como em redes industriais, onde existem requisitos de tempo real (*e.g,* subestações elétricas), o alto volume de tráfego pode gerar desafios na geração e avaliação de múltiplas estratégias de geração de RCLs. No entanto, a geração de RCLs através das implementações existentes na literatura não contempla escalabilidade, já que as mesmas são monolíticas e sequenciais (ESSEGHIR, 2010)(QUINCOZES et al., 2021) (HONOVAN; LADEIRA; BRAGA, 2015)(CARVALHO et al., 2022)(KANAKARAJAN; MUNIASAMY, 2016). Segundo (NEWMAN, 2021), os microsserviços são pequenos e totalmente focados em realizar algo específico, seja de alto ou baixo grau de complexidade. Com isso, pode-se ganhar escalabilidade na geração de RCLs para o GRASP-FS.

Este trabalho propõe uma arquitetura chamada *Distributed RCL Generator* (DRG) para decomposição dos algoritmos empregados na implementação monolítica da etapa construtiva do GRASP-FS em microsserviços distribuídos a fim de alcançar paralelismo, escalabilidade e desempenho na geração e avaliação de RCLs. Como prova de conceito, utiliza-se o *framework* Kafka e quatro algoritmos de seleção de atributos a fim de avaliar a escalabilidade da arquitetura proposta, variando-se parâmetros como o tamanho do RCL e verificando-se o tempo para descoberta de soluções. Os resultados obtidos a partir de um ambiente de redes de subestações elétricas modelado no *framework* ERENO (QUINCOZES et al., 2022) indicam que a estratégia de construção do RCL impacta na F1-Score média das soluções geradas: entre 50,47% e 84,92% para a detecção de intrusões em redes de subestações, a depender do algoritmo empregado.

1.2 Objetivos

Este trabalho tem por finalidade a decomposição dos algoritmos empregados na implementação monolítica da etapa construtiva do GRASP-FS em microsserviços distribuídos a fim de alcançar paralelismo, escalabilidade e desempenho. Com isso, espera-se prover uma RCL com os atributos mais representativos, facilitando assim o processo de busca local a partir da solução gulosa gerada. Neste contexto, os objetivos específicos deste trabalho serão abordados, incluindo:

- Estudo de arquiteturas baseadas em microsserviços;
- Estudo de técnicas para a geração de RCLs;

- Implementação de microsserviços para geração de RCLs;
- Apresentar um estudo comparativo de desempenho entre a geração de RCL monolítica e a abordagem distribuída proposta;
- Avaliar a escalabilidade da arquitetura proposta, variando-se parâmetros como o tamanho do RCL e verificando-se o tempo para descoberta de soluções;
- Avaliação de soluções descobertas através da implementação de um IDS que faz uso de algoritmos de aprendizado de máquina.

1.3 Metodologia

Para resolver os problemas de escalabilidade no emprego de novas formas de construção de RCLs e para alcançar uma maior eficiência na execução da etapa de construção, propõe-se neste trabalho uma arquitetura baseada em microsserviços, utilizando alguns *frameworks* existentes como suporte. Os materiais e métodos que serão utilizados no processo de decomposição da aplicação monolítica do algoritmo GRASP-FS para uma arquitetura de microsserviços são os seguintes:

- *Spring Boot*: O *Spring Boot* será utilizado para o desenvolvimento dos microsserviços. Cada microsserviço será responsável por um algoritmo de seleção de atributos específico. O uso do *Spring Boot* permite uma abordagem eficiente e simplificada para a construção dos microsserviços, fornecendo recursos e bibliotecas que facilitam o desenvolvimento, a configuração e a implantação.
- Apache Kafka: O Apache Kafka será adotado como sistema de mensageiria de publicação e subscrição para realizar o transporte de dados entre os microsserviços e as fases de construção e busca do GRASP-FS. O uso do Apache Kafka permite o envio assíncrono de mensagens entre os componentes do sistema, proporcionando uma comunicação eficiente e escalável. Ele atua como um barramento de mensagens, permitindo que os microsserviços se comuniquem de forma assíncrona e tolerante a falhas.
- Docker: O Docker será utilizado para a conteinerização do Apache Kafka e dos microsserviços desenvolvidos com o *Spring Boot*. A conteinerização permite empacotar cada componente do sistema, juntamente com suas dependências, em contêineres isolados. Isso simplifica o processo de implantação e garante a portabilidade e a consistência do ambiente de execução.

A utilização dessas tecnologias e *frameworks* visa proporcionar uma arquitetura distribuída e escalável para a construção de RCLs no algoritmo GRASP-FS. A abordagem de

microsserviços permite a modularização e o isolamento dos diferentes algoritmos de seleção de atributos, facilitando a execução paralela e distribuída das etapas de construção. Além disso, o uso do Apache Kafka como sistema de mensageiria proporciona uma comunicação assíncrona e eficiente entre os microsserviços. A conteinerização com o *Docker* garante a portabilidade e a consistência do ambiente de execução, simplificando a implantação e o gerenciamento dos componentes. Essa arquitetura baseada em microsserviços e os recursos tecnológicos utilizados contribuem para melhorar a escalabilidade, eficiência e flexibilidade do processo de construção de RCLs no algoritmo GRASP-FS.

2 Referencial Teórico

Nesse capítulo será abordar os principais conceitos utilizados por esse trabalho que são IDS, a metaheurística GRASP-FS e sua fase de construção, os métodos de seleção de atributos, os protocolos utilizados para obtenção de dados para teste e os trabalhos relacionados que utilizam o GRASP.

2.1 Conceitos Fundamentais

Esta seção tem como objetivo apresentar a base teórica necessária para o desenvolvimento da pesquisa. Inicialmente, focaremos na descrição dos protocolos GOOSE e SV, estabelecidos pela norma IEC-61850 e utilizados na comunicação em subestações digitais. Posteriormente, será discutido IDS e as técnicas de seleção de características, que são aplicadas na detecção de intrusões a esses protocolos. Cada um desses tópicos contribui para a melhor compreensão do campo de estudo e do contexto da pesquisa em andamento.

2.1.1 Redes de Comunicação em Subestações Digitais

Esta subseção se concentra nos principais componentes de uma rede de comunicação em subestações digitais, especificamente nos protocolos padrão IEC 61850. Tais protocolos, que incluem o Evento Genérico Orientado a Objeto de Subestação, ou em inglês, *Generic Object Oriented Substation Events* (GOOSE), e os Valores Amostrados, ou em inglês, *Sampled Values* (SV), desempenham papéis fundamentais na eficiência e segurança da transmissão e distribuição de energia.

A IEC-61850 é uma norma internacional desenvolvida pela Comissão Eletrotécnica Internacional, ou do inglês, *International Electrotechnical Commission*(IEC), que define um conjunto de padrões para a comunicação e integração de sistemas de automação em subestações elétricas. Essa norma estabelece diretrizes para a troca de informações entre os Dispositivos Eletrônicos Inteligentes, ou do inglês, *Intelligent Electronic Devices* (IEDs), utilizados nas subestações, visando garantir a interoperabilidade, eficiência e segurança das operações elétricas.

O protocolo GOOSE é um componente fundamental do padrão IEC 61850. Ele possibilita a troca eficiente e confiável de informações entre os IEDs sobre eventos, status de equipamentos e comandos de controle. O GOOSE foi projetado para ser transmitido em *multicast* na camada de enlace de dados, o que significa que uma única mensagem pode ser enviada para vários dispositivos simultaneamente. Essa abordagem permite uma comunicação ágil e sincronizada entre os diversos componentes de uma subestação. Ao

utilizar o protocolo GOOSE, é possível estabelecer uma comunicação em tempo real e confiável entre os dispositivos da subestação, possibilitando uma melhor coordenação e controle dos equipamentos elétricos (KUSH et al., 2014).

Já o protocolo SV permite a transferência de amostras digitalizadas de corrente e tensão para os IEDs por meio de *Ethernet*. Essas amostras são coletadas por meio de sinais analógicos de equipamentos elétricos e convertidas em sinais digitais pelas Unidades de Medição, ou no inglês, *Measurement Units* (MUs). Uma vez convertidas, as mensagens SV são transmitidas para dispositivos assinantes, como os IEDs de controle e/ou proteção (AFTAB et al., 2020).

2.1.2 Sistemas de Detecção de Intrusões

Os IDSs são ferramentas fundamentais na área de segurança da informação, projetadas para monitorar e analisar eventos em um sistema de computador ou rede em busca de sinais de intrusões. Essas intrusões são definidas como tentativas de comprometer a confidencialidade, integridade ou disponibilidade dos recursos, bem como contornar os mecanismos de segurança estabelecidos. Esses sistemas são capazes de detectar diferentes tipos de intrusões, incluindo invasores externos que exploram vulnerabilidades por meio da Internet, usuários autorizados que tentam obter privilégios além dos concedidos ou usuários autorizados que fazem uso indevido de seus privilégios (BACE; MELL et al., 2001).

A detecção de intrusões é realizada por meio do monitoramento contínuo dos eventos e registros gerados pelo sistema. Existem duas abordagens principais utilizadas pelos IDSs: detecção de anomalias e detecção de assinaturas (AXELSSON, 2000).

A detecção de anomalias envolve a identificação de comportamentos incomuns ou padrões que não se encaixam no perfil normal de operação do sistema. Essa abordagem permite identificar atividades suspeitas que podem indicar a presença de um invasor (AXELSSON, 2000).

Já a detecção de assinaturas envolve a comparação dos eventos observados com padrões pré-definidos de ataques conhecidos. Esses padrões, também chamados de assinaturas, são baseados em informações de ataques previamente registrados. Quando uma correspondência é encontrada, o IDS emite um alerta indicando a possível presença de uma intrusão (AXELSSON, 2000).

2.1.3 Seleção de Features

As categorias dos métodos para a seleção de atributos variam de acordo com a estratégia. Na Seção 2.1.3.1 serão apresentadas as abordagens tradicionais para a seleção

de atributos. Em seguida, na Seção 2.1.3.2, será abordado o uso de metaheurísticas para otimizar o processo de seleção de atributoss.

2.1.3.1 Métodos Tradicionais

Os métodos tradicionais de seleção de atributos podem ser classificados em três categorias: filtro (*filter*), embrulho (*wrapper*) e embutido (*embedded*) (GUYON; ELISSEEFF, 2003). As técnicas de filtro selecionam subconjuntos de atributos na etapa de pré-processamento, sem auxílio de um classificador. A avaliação para cada subconjunto envolve diversas métricas como ganho de informação, consistência e relevância (ESKANDARI; JAVIDI, 2016). Os métodos *wrapper* utilizam classificadores para avaliar os subconjuntos de atributos gerados. Por fim, a técnica *embedded* que atrela o processo de seleção de atributos ao treinamento do modelo.

Considerando-se a aplicabilidade em cenários de redes industriais onde existem dispositivos que operam sob requisitos de tempo real, neste trabalho são adotados algoritmos da categoria *filter* devido ao menor custo computacional envolvido (ABREU et al., 2020). A seguir, tais algoritmos são descritos.

A técnica de ganho de informação, do inglês *Information Gain* (IG), calcula a redução na entropia ao transformar um conjunto de dados. É usada para construir árvores de decisão com base em um conjunto de dados de treinamento. Avalia o ganho de informação de cada variável e escolhe aquela que maximiza o ganho de informação, minimizando a entropia e dividindo os dados em grupos para classificação eficiente (KULLBACK; LEIBLER, 1951). O critério de ganho seleciona o atributo de teste que maximiza o ganho de informação. O desafio principal é que os atributos com múltiplos valores são preferidos. O IG é baseado no cálculo da redução de entropia para medir a relevância de um atributo em relação à classe das amostras analisadas. A fórmula do *Information Gain*, representada pela Equação 2.1, onde IG, é dada por $H(C) - H(C|F)$, onde F é o atributo considerado, C é a classe, $H(C)$ é a entropia da classe e $H(C|F)$ é a entropia condicional da classe dado o valor de F.

$$IG(F, C) = H(C) - H(C|F) \tag{2.1}$$

Ao calcular o IG para diferentes atributos, é possível determinar a relevância desses atributos em relação à classe. Quanto maior o valor do IG, maior a redução de entropia e maior a relevância do atributo em distinguir as classes (QUINLAN, 1986).

Para superar o desafio do algoritmo IG, foi proposta uma nova técnica chamada Razão de Ganho, do inglês *Gain Ratio* (GR). O GR consiste em utilizar o ganho de informação ponderado como critério de avaliação (SALZBERG, 1994). A técnica envolve duas etapas. Na primeira, são calculados os ganhos de informação para todos os atributos.

Em seguida, são selecionados apenas os atributos que obtiveram um ganho de informação igual ou superior à média, sendo escolhido aquele com a melhor razão de ganho (QUILAN, 1988). O algoritmo *Gain Ratio* (GR) é baseado no IG discutido anteriormente e considera a distribuição das classes para corrigir o viés em direção a atributos com muitos valores. O *Gain Ratio* é calculado pela fórmula apresentada na Equação 2.2, onde GR é o *Gain Ratio*, F é o atributo considerado, C é a classe, $IG(F, C)$ é o *Information Gain* do atributo F em relação à classe C, e $H(F)$ é a entropia do atributo F.

$$GR(F, C) = \frac{IG(F, C)}{H(F)} \tag{2.2}$$

Ao calcular o GR para diferentes atributos, é possível comparar a relevância desses atributos em relação à classe, levando em consideração a distribuição das classes e evitando o viés em direção a atributos com muitos valores. Valores mais altos de *Gain Ratio* indicam atributos que possuem alta informação em relação à classe e são menos dependentes do número de valores que o atributo pode assumir (QUINLAN, 1993).

Outra variação do IG consiste no algoritmo de Incerteza Simétrica, do inglês *Symmetrical Uncertainty* (SU). Tal técnica pondera o ganho de informação, mas dividindo o ganho de informação pela entropia do atributo somada a entropia da classe. Esse valor é multiplicado por 2 para que os valores estejam entre 0 e 1 (KANNAN; RAMARAJ, 2010). O algorítimo SU leva em consideração a entropia conjunta entre o atributo e a classe, assim como a entropia individual do atributo e da classe, para identificar a dependência mútua entre eles e selecionar atributos que possuam alta informação mútua e baixa redundância. O SU é dado pela fórmula denotada na Equação 2.3, onde SU é o *Symmetrical Uncertainty*, F é o atributo considerado, C é a classe, $IG(F, C)$ é o *Information Gain* do atributo F em relação à classe C, $H(F)$ é a entropia do atributo F, e $H(C)$ é a entropia da classe (KIRA; RENDELL, 1992).

$$SU(F, C) = \frac{2 \times IG(F, C)}{H(F) + H(C)} \tag{2.3}$$

Ao calcular o SU para diferentes atributos, é possível comparar a relevância desses em relação à classe. Valores mais altos de SU indicam atributo que possuem alta informação mútua com a classe e baixa redundância, sendo assim, são considerados mais relevantes para a tarefa de classificação (KONONENKO et al., 1994).

Por fim, a técnica *ReliefF* tem como objetivo selecionar variáveis que possam distinguir entre casos de diferentes classes. O *ReliefF* se baseia na preservação da estrutura especial da matriz de similaridade dos dados, pressupondo que o conjunto de dados possua o mesmo número de ocorrências em cada classe (ROBNIK-ŠIKONJA; KONONENKO, 2003). É importante observar que a medida *ReliefF* utiliza testes estatísticos, o que pode levar a comportamentos incorretos quando aplicada a termos raros em uma coleção ou

quando o número de exemplos positivos no treinamento é muito baixo para um determinado conceito.

O algoritmo *ReliefF* avalia a importância dos atributos ao estimar o quão bem eles conseguem distinguir entre as classes, comparando os valores dos atributos entre instâncias próximas e distantes por meio de uma função de distância. A fórmula exata para o cálculo do *ReliefF* depende do cálculo das distâncias entre instâncias próximas e distantes, assim como da atualização dos pesos dos atributos. A Equação 2.4 apresenta a fórmula do *ReliefF*, que descreve os cálculos relacionados aos pesos dos atributos. Nessa equação, W_i representa o peso atual do atributo i da instância em consideração. Além disso, $\texttt{nearHit}_i$ corresponde ao valor do atributo i na instância mais próxima pertencente à mesma classe que a instância atual, enquanto $\texttt{nearMiss}_i$ representa o valor do atributo i na instância mais próxima pertencente a uma classe diferente da instância atual.

$$W_i = W_i - (x_i - \texttt{nearHit}_i)^2 + (x_i - \texttt{nearMiss}_i)^2 \tag{2.4}$$

Nessa fórmula, subtrai-se o quadrado da diferença entre o valor atual do atributo e o valor do "*nearest hit*", e adiciona-se o quadrado da diferença entre o valor atual do atributo e o valor do "*nearest miss*". Essa atualização dos pesos tem como objetivo reduzir o peso do atributo se ele difere mais dos valores dos atributos próximos da mesma classe e aumentar o peso se ele difere mais dos valores dos atributos próximos de classes diferentes. Essa fórmula é utilizada em cada iteração do algoritmo *ReliefF* para atualizar os pesos de cada atributo, levando em consideração as instâncias próximas e distantes. Ao final do algoritmo, os pesos atualizados podem ser normalizados para obter um vetor de relevância, onde atributos com pesos mais altos são considerados mais relevantes na distinção entre as classes (KIRA; RENDELL, 1992).

2.1.3.2 Metaheurísticas

Ao passo que as técnicas tradicionais de seleção de atributos apresentadas anteriormente se dividam em três principais categorias, pode-se ainda explorar alternativas baseadas em metaheurísticas para combinar tais categorias e, dessa forma, permitir a otimização do processo de seleção de atributos. As metaheurísticas permitem, por exemplo, a combinação entre técnicas baseada em filtro e baseada em *wrapping* (ESSEGHIR, 2010).

Existem diversas metaheurísticas disponíveis na literatura, mas algumas demonstram-se mais promissoras em alguns estudos com o propósito de seleção de atributos. Yusta (YUSTA, 2009) demonstra que o GRASP pode superar os algoritmos *Sequential Forward Feature Selection* (SFFS), Busca Tabu, Genético e Memético na geração do melhor subconjunto de características para classificar amostras de diferentes bancos de dados. Portanto, nesta monografia, será adotada a implementação do GRASP-FS (QUINCOZES et al., 2020)(QUIN-

COZES et al., 2021)(QUINCOZES et al., 2022) como base para a proposta da otimização da sua etapa construtiva através de microsserviços. Mais detalhes acerca de estudos da literatura envolvendo tal metaherurística serão apresentados na Seção 2.2. Já os detalhes acerca da implementação do GRASP-FS, bem como sua otimização através de microserviços serão apresentados na Seção 3.3.

2.2 Trabalhos Relacionados

Nesta seção são apresentados trabalhos que aplicam soluções baseadas na metaheurística GRASP para a seleção de atributos. Com base em nossa revisão da literatura, nenhum dos trabalhos existentes implementa uma arquitetura de microsserviços para a geração escalável de RCLs. Os trabalhos relacionados são apresentados na Tabela 1 e discutidos a seguir.

Ref.	GRASP	Construção de RCL	Escalabilidade	Distribuição
Esseghir (2010)	√	ReliefF, SU, FCBF	×	×
Quincozes et al. (2020)	√	IG	×	×
Quincozes et al. (2022)	√	GR	×	×
Quincozes et al. (2021)	√	IG, GR, OneR e IWSS	×	×
Honovan, Ladeira e Braga (2015)	√	SFS, SFE e SFFS	×	×
Carvalho et al. (2022)	√	Aleatório	×	×
Diez-Pastor et al. (2011)	√	IG	×	×
Pastor, Osorio e Rodriguez (2014)	√	IG	×	×
Kanakarajan e Muniasamy (2016)	√	IG, SU e CFS	×	×
Esse trabalho	√	GR, IG, SU, ReliefF	√	√

Tabela 1 – Propostas de seleção de features baseadas na metaheurística GRASP. Fonte: Do Autor.

Em Esseghir (2010) é proposta uma solução baseada na metaheurística GRASP para resolver o problema de seleção de atributos. Ao empregar tal metaheurística, é possível obter-se uma abordagem híbrida entre as técnicas de seleção *filter* e *wrapper*. A eficácia das diferentes combinações de componentes GRASP foi avaliada empiricamente e com resultados obtidos, confirmam a robustez da proposta. Esse trabalho motiva a investigação em aspectos algorítmicos e comportamentais em relação as combinações e novas formas de construção de RCL. No entanto, os autores não consideram as técnicas de seleção de atributos IG e GR nos experimentos. Outra limitação consiste na falta de soluções para o alcance de escalabilidade na solução.

O trabalho Honovan, Ladeira e Braga (2015) apresenta uma comparação das metaheurísticas GRASP e *Tabu Search* para a seleção de atributos. Tal trabalho tem como finalidade obter o menor conjunto de atributos capaz de fornecer a mais alta acurácia

para os algoritmos classificadores. Os experimentos realizados mostraram a eficácia da abordagem em termos de acurácia. Contudo, conforme o número de atributos aumenta, o custo computacional pode crescer a ponto de não se tornar viável atingir uma solução adequada em tempo hábil. Isso se deve principalmente à falta de escalabilidade da solução, visto que não são consideradas estratégias de paralelismo. Outra limitação consiste no uso de apenas algoritmos *wrapper* (*i.e.,* SFS, SFE e SFFS). Tais algoritmos tipicamente demandam mais processamento — e tempo para atingir uma solução adequada — quando comparados aos algoritmos da categoria *filter* (QUINCOZES et al., 2020).

No trabalho Carvalho et al. (2022), é proposta uma arquitetura para detecção e prevenção de intrusão em tempo real em *switches* eBPF. Tal proposta baseia-se na otimização assíncrona do modelo de detecção de intrusão com a metaheurística GRASP-FS e a adaptação do algoritmo *MLK-Nearest Neighbors* (KNN) com filtros eBPF. De modo a alcançar a escalabilidade na detecção de intrusões em tempo real, os classificadores são instalados diretamente nos dispositivos de rede (*switches* eBPF). No entanto, a configuração e a otimização dos modelos de aprendizado de máquina (*e.g., feature selection*) são realizadas de maneira sequencial em um computador com poder de computação e memória suficientes. Portanto, embora que a execução do GRASP-FS não afete o tempo de detecção do ataque, o tempo de convergência para o alcance de boas soluções ainda representa um gargalo para a solução.

No trabalho Diez-Pastor et al. (2011) é proposto o algoritmo GRASP Forest (G-Forest) para a construção de conjuntos (*ensembles)* de árvores de decisão. Tal método emprega os conceitos do GRASP tanto para a seleção de atributoss quanto para a escolha de pontos de divisão para cada nó das árvores de decisão. O algoritmo IG é usado para a construção de RCLs para a seleção de atributos. Então, atributos são escolhidos aleatoriamente para a construção das árvores de decisão. O IG também é empregado na escolha do ponto de divisão de cada nó. Tal processo ocorre iterativamente até alcançar um número desejado de árvores de decisão. Em um trabalho posterior, Pastor, Osorio e Rodriguez (2014) estende o G-Forest para a proposta de "Floresta de Aleatoriedade Recozida", do inglês, *Annealed Randomness Forest* (GAR-Forest). A ideia principal dessa versão estendida consiste na parametrização da aleatoriedade durante a fase de construção de RCLs. Ambas as propostas de Diez não foram aplicadas no contexto de detecção de intrusões e não contemplam processamento distribuído nem escalabilidade.

Em Kanakarajan e Muniasamy (2016), o algoritmo GAR-Forest é empregado na detecção de intrusões. Ademais, os algoritmos IG, SU e *Correlation-based Feature Subset* (CFS) foram utilizados para selecionar atributos relevantes e melhorar a precisão da GAR-Forest. No entanto, as limitações de escalabilidade não são resolvidas.

No trabalho Quincozes et al. (2020) é proposta a abordagem GRASP-FS, que consiste em uma adaptação da metaheurística GRASP aplicada a seleção de atributos. O

algoritmo IG foi usado originalmente na construção de RCL para o GRASP-FS. Posteriormente, em uma versão estendida Quincozes et al. (2022), tal algoritmo foi substituido pelo GR. Por fim, Quincozes et al. (2021) contempla uma comparação entre os algoritmos IG, GR, OneR e IWSS. Os resultados revelam que GRASP supera os métodos tradicionais baseados em filtros. Tais estudos apontam que o descarte prematuro de atributos para compor o RCL deve ser evitado e o mesmo deve ser construído considerando-se um comprimento variável. Essas decisões permitem que o GRASP funcione adequadamente e, assim, otimize os resultados. No entanto, em todas as variações propostas, adotou-se métodos de construção sequencial de RCLs, sem implementar distribuição e paralelização para o alcance de escalabilidade. Ademais, as técnicas *ReliefF* e SU não são estudadas.

Por fim, percebe-se que embora que existam diferentes propostas com a finalidade de seleção de atributos na literatura através da metaheurística GRASP, a arquitetura sequencial e falta de escalabilidade são desafios comuns para todos esses trabalhos. O GRASP-FS emprega algoritmos de seleção de atributos para processar volumes potencialmente grandes de dados em sua fase construção a fim de construir listas de atributos candidatos (*i.e.,* RCLs) a compor soluções (*i.e.,* conjuntos de atributos) com alta representatividade para a detecção de intrusões. A partir do RCL, a etapa de busca local do GRASP-FS gera soluções mais otimizadas. Assim, a de construção de uma RCL representativa é fundamental para que boas soluções sejam encontradas na etapa de busca local. Tanto a fase construtiva quanto a de busca local do GRASP-FS apresentam desafios de escalabilidade. Em particular, este trabalho aborda o gargalo da etapa construtiva do GRASP-FS enquanto que os desafios da etapa de busca local são abordados em Silva et al. (2023).

3 Desenvolvimento

Neste capítulo, serão abordados diversos aspectos relacionados à seleção de características utilizando o algoritmo GRASP-FS. A seleção de atributos é uma etapa fundamental no processo de modelagem de dados, visando identificar um subconjunto otimizado de características que impacte positivamente o desempenho dos modelos de aprendizado de máquina (ESSEGHIR, 2010)(QUINCOZES et al., 2020).

Inicialmente, na Seção 3.1, serão discutidos os princípios e o funcionamento do algoritmo. Em seguida, na Seção 3.2, será apresentada a principal contribuição deste trabalho, a qual consiste na proposta de uma abordagem orientada a microserviços para a etapa construtiva do GRASP-FS, que discute uma arquitetura de sistema que utiliza microserviços para a implementação do GRASP-FS em ambientes distribuídos. Por fim, na Seção 3.3, será descrita a implementação prática do algoritmo em um ambiente de teste, seguida da realização de experimentos para avaliar sua eficácia e desempenho.

Essas seções fornecerão uma visão abrangente e detalhada sobre a aplicação do GRASP-FS na seleção de atributos, bem como a implementação e avaliação de sua viabilidade em um cenário prático.

3.1 Seleção de Features com GRASP-FS

Nesta seção, apresenta-se a implementação do algoritmo GRASP-FS, que consiste em um processo iterativo de múltiplas inicializações. Cada iteração do algoritmo é composta por duas fases: construção e busca local, conforme ilustrado na Figura 1, a qual é auto-descritiva. Nessa ilustração, a fase de construção que compreende o escopo deste trabalho é destacada na cor verde.

Durante a fase de construção, é gerada uma solução viável, e posteriormente, na fase de busca local, o espaço de vizinhança é explorado até que seja encontrada uma solução ótima local. O início dessas fases é identificado pelas linhas 9 e 14 do Algoritmo 1. A aplicação do GRASP ao problema de seleção de características tem como objetivo maximizar uma função de aptidão, tal como acurácia ou F1-Score, ao mesmo tempo em que se controla o custo computacional. Assim, obtém-se como resultado um subconjunto otimizado de características (QUINCOZES et al., 2021).

A fim de reduzir o conjunto de recursos para análise, uma abordagem conhecida como RCL é empregada. A RCL compreende de uma lista restrita de candidatos que têm maior potencial de gerar uma solução factível, isto é, com maiores probabilidades de fornecer um conjunto de atributos que forneça uma boa performance na detecção de

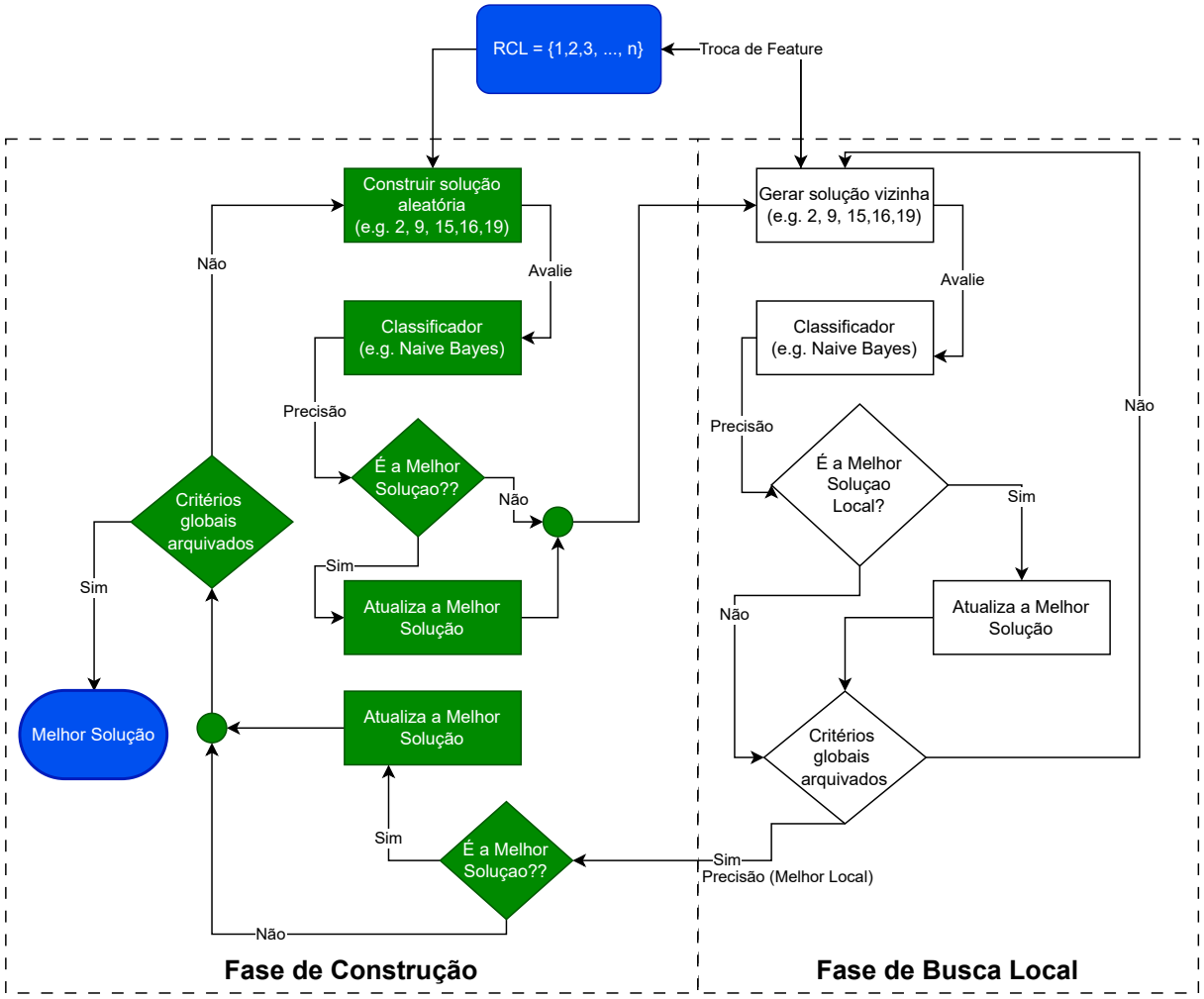


Figura 1 – Processo GRASP baseado em FS.
Fonte: Adaptado de (QUINCOZES et al., 2021).

intrusões. Portanto, a criação de uma RCL envolve a exclusão de atributos que, consequentemente, não serão utilizadas tanto na fase de construção quanto na fase de busca local. Nesta implementação específica, a RCL é gerada apenas uma vez para evitar sobrecarga desnecessária, uma vez que os algoritmos de filtragem de atributos empregados nesta etapa são determinísticos (QUINCOZES et al., 2021).

Os subconjuntos gerados na fase de construção são utilizados como soluções iniciais para explorar a vizinhança durante o procedimento de busca local. Esse processo é repetido até que um critério global seja alcançado. Em cada iteração global, a busca local é realizada até atingir o número máximo de iterações. Cada solução viável encontrada é avaliada por meio da função de aptidão, e a melhor solução geral é mantida. Quando o critério local é atingido, a melhor solução local é fornecida como resultado e, em seguida, comparada com a melhor solução global. Após atingir o tempo máximo, o melhor resultado global é apresentado como a saída final (QUINCOZES et al., 2021).

Algorithm 1: GRASP-FS Adaptado de (QUINCOZES et al., 2020)

```
Entrada: all // Todos os atributos disponíveis
         maxTime // tempo máximo de execução do GRASP
         L-RCL // quantidade de atributos para compor a RCL
         |FS| // comprimento inicial dos atributos
         maxIt // número máximo de iterações
Saída  : bestFS // subconjunto de atributos otimizado
begin
    bestFS ← ∅
    bestFS.accuracy ← 0
    while currentTime < maxTime do
        greRaFS ← construct(|FS|, L-RCL, all)
        greRaFS.accuracy ← eval(greRaFS)
        if greRaFS.accuracy > bestFS.accuracy then
            bestFS ← greRaFS
        end
        bestLocal ← localSearch(greRaFS, RCL, maxIt)
        if bestLocal.accuracy > bestFS.accuracy then
            bestFS ← bestLocal
        end
    end
end
return bestFS
```

Algorithm 2: Fase de Construção Adaptado de (QUINCOZES et al., 2021)

```
entrada: all // todas os atributos disponíveis
         L-RCL // quantidade de atributos para compor a RCL
         |FS| // comprimento inicial do conjunto de atributos
saída  : greRaFS // subconjunto Greedy-Random FS
begin
    if rclFeatures = ∅ then
        foreach feature ∈ all do
            GR ← getGr(feature)// cálculo do GR
        end
        all ← attributeRankByGR(all)// classificar os atributos por GR rclFeatures
          ← selectTopRanked(all, L-RCL)// selecionar os melhores atributos
    end
    greRaFS ← ∅
    while (|greRaFS| < |FS|) do
        greRaFS ← ∪ selectRandomly(feature ∈ rclFeatures )// seleção aleatória de
          atributos
    end
end
return greRaFS
```

Nesta implementação, o método GR foi empregado para a realização da classificação dos atributos durante a fase de construção e, em seguida, selecionar as características com melhor classificação. Portanto, na primeira iteração do GRASP, quando a lista de características candidatas (`rclFeatures`) está vazia, o valor GR é calculado para cada atributo (veja linha 4, Algoritmo 2). Uma vez que a RCL é gerada, ela é usada para gerar a solução inicial (*i.e., greRaFS*, na Figura 1) de forma gulosa e aleatória para cada iteração do GRASP. Em particular, atributos são escolhidos aleatoriamente da RCL para serem adicionados a *greRaFS* (veja linha 10, Algoritmo 2) até que o comprimento da solução atinja o limite `|FS|`. Esta etapa é representada como Construir solução aleatória, na Figura 1, e é seguida por uma avaliação de classificador. A solução avaliada é comparada com a melhor solução atual, que é atualizada quando é superada (QUINCOZES et al., 2021).

O algoritmo descrito nesta seção foi projetado para ser executado sequencialmente, portanto, as etapas de construção e busca local não acontecem em paralelo. Similarmente, para experimentar-se diferentes alternativas de algoritmos para a etapa construtiva, deve-se repetir a execução completa do algoritmo GRASP-FS. Tal abordagem dificulta a adequação do conjunto de atributos para o contexto observado dinamicamente por IDSs em redes que podem ter seu padrão de tráfico alterado constantemente. Desta forma, para um IDS que opera em tempo real é fundamental que o GRASP-FS seja implementado de modo a fornecer soluções continuamente, de forma iterativa e paralela. Isso se deve ao fato de que um conjunto de atributos ideal em um cenário pode deixar de apresentar boa performance em caso de mudanças comportamentais na rede. Na próxima seção, é apresentada uma abordagem baseada em microsserviços para resolver tais problemas.

3.2 Uma Abordagem Orientada a Microserviços para o GRASP-FS

A construção de uma RCL representativa requer a execução de diferentes algoritmos e estratégias e, portanto, representa em um gargalo na etapa de construção do GRASP-FS. Para resolver os problemas de escalabilidade no emprego de múltiplas formas de construção de RCL, que tipicamente se dá de forma sequencial, e para alcançar uma maior eficiência na execução da etapa de construção, neste trabalho é proposta a utilização de uma arquitetura orientada a microsserviços chamada *Distributed RCL Generator* (DRG). O DRG permite a distribuição e o paralelismo na etapa de construção na metaheurística GRASP-FS, permitindo assim a exploração de uma maior diversidade de algoritmos e estratégias para a geração de RCLs sem comprometer o desempenho computacional.

Para esse trabalho utilizamos da arquitetura baseada em eventos, onde cada microsserviço possui os componentes: *controller*, *service*, *producer*, *consumer* que possui suas

bases teóricas em princípios de design orientado a eventos, onde os componentes do sistema são desacoplados e se comunicam através da troca de mensagens assíncronas. Essa abordagem oferece várias vantagens, como maior escalabilidade, robustez, flexibilidade e facilidade de manutenção (NEWMAN, 2021).

- O *controller* atua como um ponto de entrada para o DRG. Ele pode ser implementado como uma Interface de Programação de Aplicação, do inglês, *Application Programming Interface* (API) ou outro mecanismo de interface do usuário. O *controller* recebe as requisições, e as direciona para o componente *service*.
- O *service* é responsável por executar a lógica e processar os dados necessários para atender às solicitações. Ele pode ser composto por um ou mais componentes que trabalham em conjunto para realizar tarefas específicas.
- O *producer* é responsável por publicar mensagens no barramento de mensagens (*broker*). Essas mensagens geralmente contêm dados relevantes para o sistemas. O *producer* é acionado pelo *service* quando ocorrem eventos importantes que precisam ser propagados para outros componentes do sistema.
- O *consumer* é responsável por consumir as mensagens publicadas no *broker*. Ele pode ser projetado para realizar diversas tarefas, como processar os dados, executar ações específicas ou acionar outros serviços. O *consumer* é configurado para escutar as mensagens relevantes para sua funcionalidade e responder de acordo.

Portanto, seguindo os princípios discutidos acima, a arquitetura proposta proporciona escalabilidade ao processo de seleção de atributos, acelerando a entrega de soluções iniciais para a fase de busca local. Para alcançar maior escalabilidade, foi realizado a distribuição e processamento de FS entre vários microsserviços, cada um com uma responsabilidade diferente. Esses microsserviços utilizam o paradigma publicação e assinatura, do inglês, *publish/subscribe*, publicando as soluções iniciais que serão utilizadas na fase de busca local. A Figura 2 ilustra tal arquitetura, juntamente com sua entrada e saída.

A arquitetura proposta proporciona escalabilidade ao processo de seleção de atributos, acelerando a entrega de soluções iniciais para a fase de busca local. Para alcançar maior escalabilidade, foi realizado a distribuição e processamento de FS entre vários microsserviços, cada um com uma responsabilidade diferente. Esses microsserviços utilizam o paradigma publicação e assinatura, do inglês, *publish/subscribe*, publicando as soluções iniciais que serão utilizadas na fase de busca local.

Primeiramente, como entrada para o procedimento do GRASP-FS, é necessário a definição de um conjunto de amostras com M linhas e N colunas. A partir das N colunas, cada microsserviço (*i.e.,* Seleção de Atributo 1, Seleção de Atributo 2, ..., Seleção de Atributo N) gera a sua respectiva RCL, de acordo com a estratégia e algoritmo de seleção de

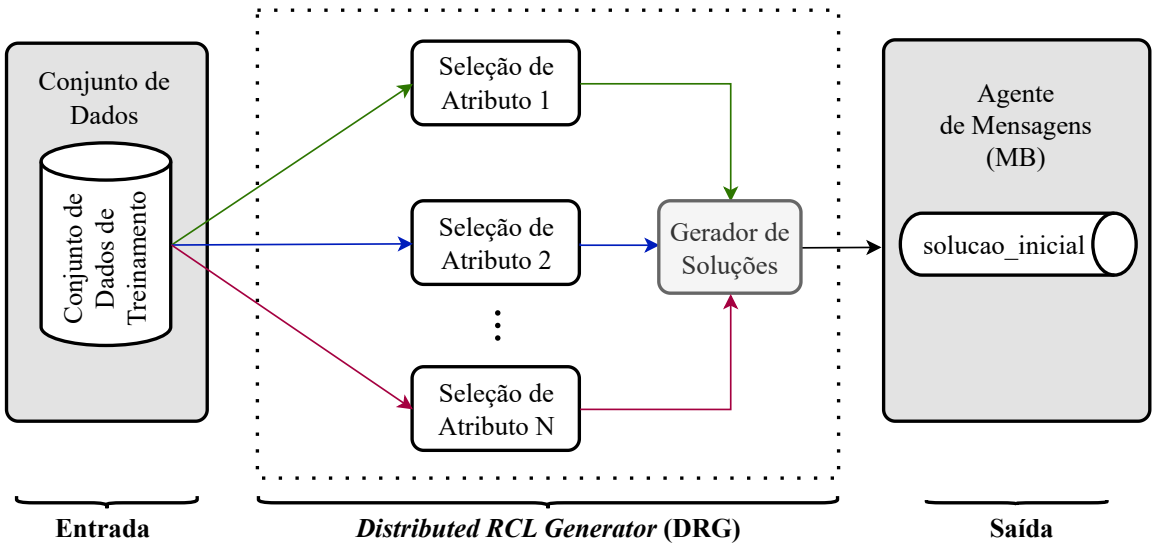


Figura 2 – A proposta de arquitetura orientada a microsserviços.
Fonte: Do Autor.

atributos implementado. Tal algoritmo considera todas as M linhas para estimar a representatividade de cada atributo. Nesse processo, é importante observar que a geração de uma RCL pequena significa menos diversidade e um risco maior de super-especialização (*overfitting*), bem como um potencial para perder características importantes. Em contraste, um RCL excessivamente grande reduz a chance de uma boa solução ser alcançada em um tempo factível. Assim, a definição do tamanho do RCL é importante para o *trade-off* entre diversidade e *overfitting* (QUINCOZES et al., 2021).

Em seguida, as RCLs geradas por cada microsserviço de seleção de atributos são usadas como entrada para a função Gerador de Solução, a qual executa a geração aleatória de soluções iniciais a partir desses RCLs recebidos.

Por fim, as soluções geradas são publicadas em um tópico de soluções iniciais chamado solucao_inicial. Esse tópico representa o ponto de partida para a segunda fase do método GRASP-FS, na qual se realiza a busca local (SILVA et al., 2023).

3.3 Implementação e Experimentos

Nesta seção é apresentada como prova de conceito uma implementação que instância a arquitetura proposta. A Figura 3 ilustra os componentes implementados.

De modo a comparar as técnicas de geração de RCL implementadas por meio de microsserviços na arquitetura proposta, tais serviços foram executados de modo a gerar 30.000 soluções para cada algoritmo de construção de RCL implementado. Foram implementados quatro microsserviços, cada um com um dos seguintes algoritmos: *Information Gain* (IG), *Gain Ratio* (GR), *ReliefF* (RF) e *Symmetrical Uncertainty* (SU). Para cada algoritmo, foram executadas três instâncias, variando-se o tamanho do número de atri-

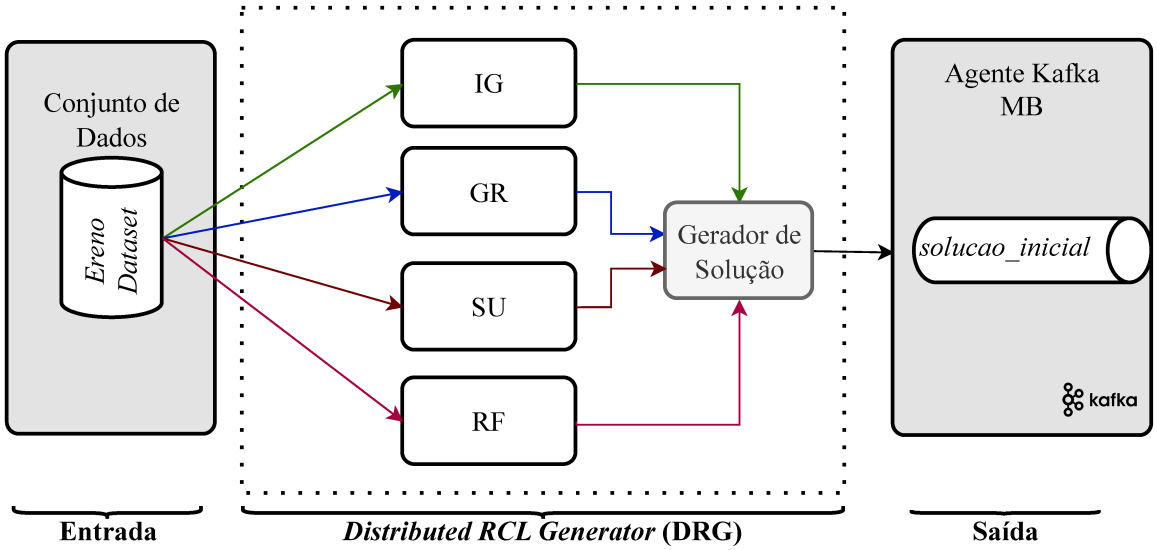


Figura 3 – Implementação dos componentes da arquitetura proposta.
Fonte: Do Autor.

butos selecionados para compor o RCL: uma instância com 10 atributos, outra com 20 atributos e a terceira com 30 atributos. Assim, cada instância de cada microsserviço gera 10.000 soluções (totalizando-se as 30.000 por algoritmo).

3.3.1 Implementação de Microsserviços

De modo a implementar-se os microsserviços, utilizou-se das ferramentas Spring Boot, Docker e Apache Kafka. Tais ferramentas são descritas a seguir.

O Spring Boot consiste em um *Framework* Java de código aberto destinada a facilitar o processo de desenvolvimento de microsserviços para aplicativos Java. Como resultado, a configuração inicial requer menos tempo, tornando o processo de desenvolvimento mais ágil. Ao utilizar o Spring Boot pode-se abstrair e simplificar a configuração de servidores, configurações de bibliotecas, gerenciamento de dependências, métricas e *health checks*, entre outros.

Ademais, o *Framework* Spring Boot pode atuar como um "micro-framework", permitindo a escolha das partes do *framework* de que precisam ser utilizadas para não sobrecarregar o tempo de execução com dependências. Ele também permite que aplicativos Spring Boot sejam agrupados em unidades menores para publicações, e sistemas de *build* podem ser usados para criar instalações como executáveis Java (*JARs*). Isso é possível porque esse *framework* é construído como uma coleção de módulos conhecidos como *starters*. Esses iniciadores são agregados de versões interoperáveis de bibliotecas que podem ser usadas para fornecer determinada funcionalidade ao aplicativo. Eles também formam a estrutura que permite que o Spring Boot configure o aplicativo de acordo com a convenção do modelo de configuração usada para corresponder à arquitetura de microsserviços

e expor a funcionalidade principal (REDDY, 2017).

Os contêineres do Docker contêm apenas os dados de que seu aplicativo realmente precisa. Portanto, os contêineres e as imagens são muito compactos em comparação com as máquinas virtuais (ANDERSON, 2015). Com isso, o arquivo executável conhecido como *Java ARchive* (JAR) pode ficar demasiadamente grande e degradar o desempenho. Outros desafios incluem a alta curva de aprendizado e a complexidade de criar um mecanismo de registro personalizado. A compensação dessas deficiências requer o uso do Docker, pois simplifica e acelera o fluxo de trabalho e permite que os artefatos do Spring Boot e o Apache Kafka sejam executados diretamente nos contêineres do Docker. Isso ajuda você a criar microsserviços rapidamente (LI; SUN, 2022)(Ajeet Singh Raina and Johan Giraldo, 2022).

O Apache Kafka consiste em um sistema de código aberto que implementa um *broker* de mensagens distribuído, por meio do paradigma *publish/subscribe*. Nesse paradigma, um microsserviço cliente publicador envia uma mensagem para o *broker* em determinado tópico e todos os clientes assinantes do respectivo tópico recebem uma cópia da mensagem. Portanto, o Apache Kafka age como um intermediário. O Kafka é comumente considerado como uma solução para lidar com altos volumes de informações em aplicações de tempo real, encaminho as mensagens para múltiplos consumidores rapidamente. O Kafka fornece integração desacoplada entre informações de produtores e consumidores. Com isso, produtores enviam mensagens assíncronas, sem nenhum tipo de bloqueio. Além disso, os produtores não precisam ter conhecimento de quais são os dispositivos consumidores nem a sua localização (GARG, 2013).

Portanto, uma vez que o Apache Kafka fornece uma solução de publicação e assinatura de mensagens em tempo real que é adequada para superar os desafios de escalabilidade de aplicações distribuídas, seu uso será adotado neste trabalho para prover a escalabilidade na geração de RCLs para a meta-heurísticas GRASP-FS.

3.3.2 Cenário

De modo a construir o cenário de avaliação utilizado nos experimentos realizados, adotou-se a ferramenta *Efficacious Reproducer Engine for Network Operations* (ERENO) (QUINCOZES et al., 2022) para a geração de um *dataset* de intrusões. Tal ferramenta contempla uma estrutura de código aberto para geração de *datasets* consistentes com ambientes ciber-físicos no contexto de subestações elétricas digitais em redes que estão em conformidade com a norma IEC–61850. O ERENO ainda permite a geração de recursos representativos, tanto em termos de atributos de rede quanto de nível de aplicação, incluindo atributos cibernéticos e físicos (*e.g.,* dados de corrente e tensão). A Figura 4 ilustra o cenário de referência (QUINCOZES et al., 2021) usado na modelagem pelo ERENO.

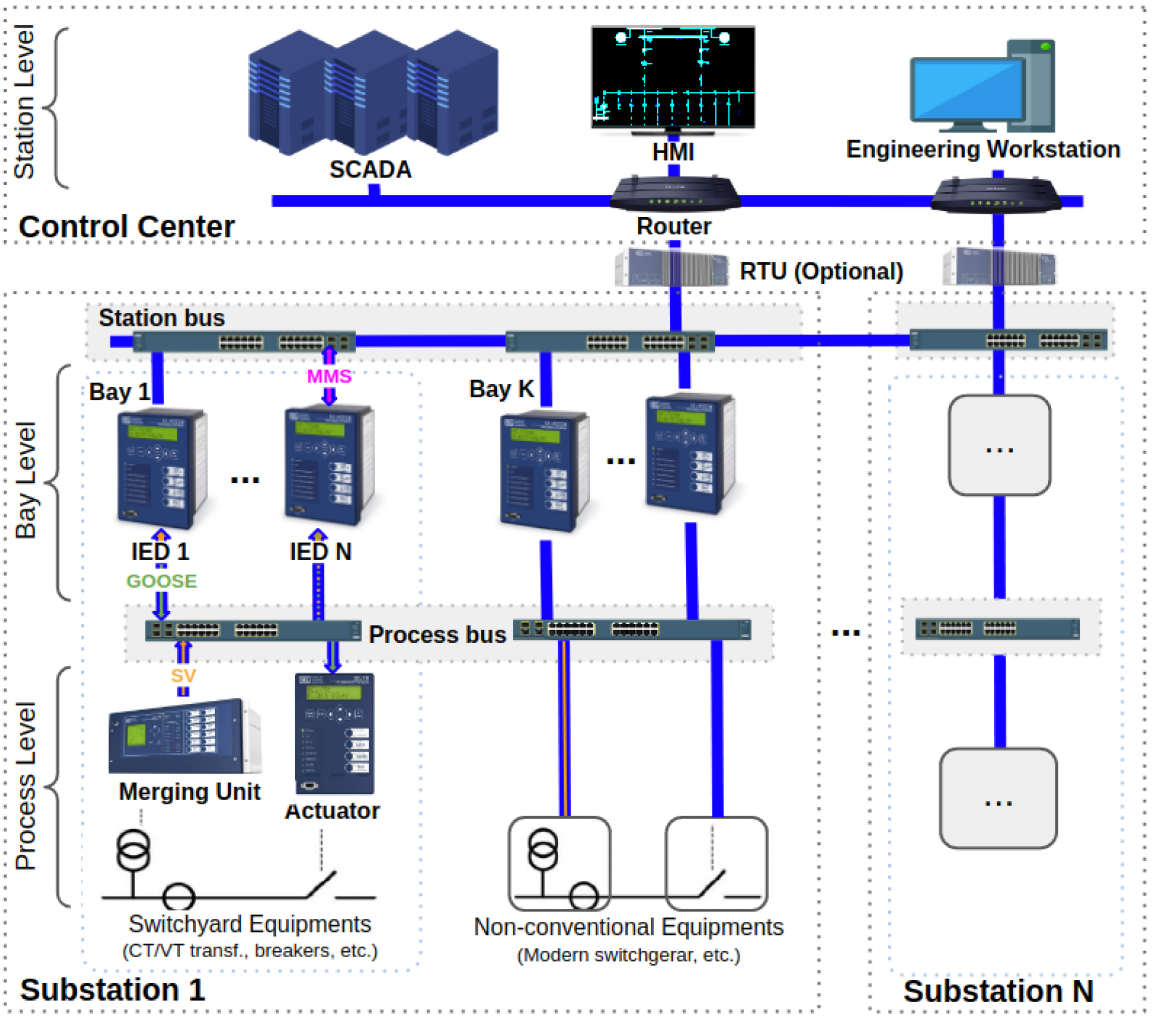


Figura 4 – Cenário de experimentação simulado.
Fonte: Retirado de (QUINCOZES et al., 2021).

Como prova de conceito, o *framework* ERENO foi configurado para a geração de um *dataset* com 1.000 amostragens que representam correlações entre mensagens transmitidas usando os protocolos GOOSE e SV, incluindo amostras de comportamento normal e também ataques contra os IEDs. Cada amostra possui 70 atributos (QUINCOZES et al., 2021)(QUINCOZES et al., 2022).

4 Resultados

Por meio da arquitetura desenvolvida e da execução dos microsserviços, foram geradas várias RCLs, nas quais determinados parâmetros foram modificados como o tamanho da RCL com o intuito de realizar uma análise posterior. Nesse contexto, será iniciada à abordagem das métricas empregadas.

4.1 Métricas

As soluções geradas aleatoriamente a partir das RCLs provindas de cada microsserviço são avaliadas através da métrica *F1-Score* (Equação 4.3). Tal métrica consiste na média harmônica das métricas Precisão e Revocação, as quais são, por sua vez, calculadas a partir do total de Falsos Positivos (FP), Falsos Negativos (FN) e Verdadeiros Positivos (VP), conforme denotado nas Equações 4.2 e 4.1 .

$$Precisão = \frac{VP}{VP + FP} \tag{4.1}$$

$$Revocação = \frac{VP}{VP + FN} \tag{4.2}$$

$$F1Score = 2 \times \frac{Precisão \times Revocação}{Precisão + Revocação} \tag{4.3}$$

4.2 Estudo de Técnicas RCLs

Primeiramente, foram medidos os desempenhos médios em termos da métrica F1-Score para cada algoritmo que implementa as técnicas de construção de RCL.

O desempenho médio alcançado a partir de soluções aleatórias geradas através das técnicas de construção ficou entre 56,37% (para o algoritmo *Gain Ratio*) e 72,32% (para o algoritmo *Symmetrical Uncertainty*). Os algoritmos *Information Gain* e *ReliefF* obtiveram um desempenho médio de 66,18% e 59%, respectivamente. Tais resultados consistem na média geral das 30.000 soluções aleatórias geradas a partir de cada RCL construído pelos respectivos algoritmos experimentados.

Na Figura 5 os resultados de cada algoritmo são detalhados de acordo com o número de *features* que são atributos que compõe cada RCL. Nota-se que o uso do algoritmo IG com uma RCL constituída de dez atributos foi capaz de gerar boas soluções com uma alta probabilidade: em média, para as 10.000 soluções geradas a partir de tal RCL,

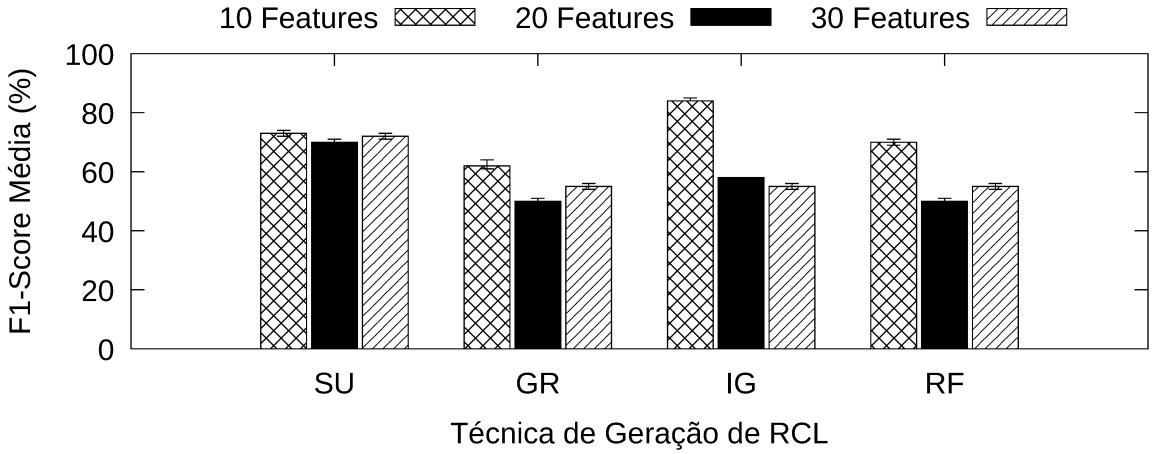


Figura 5 – F1-Score média alcançada por cada implementação de RCL.
Fonte: Do Autor.

alcançou-se uma F1-Score de 84,92%, pois com uma menor RCL, o algoritmo IG tem menos opções para escolher e pode focar nos atributos mais discriminantes e informativas. Isso resulta em um desempenho melhor, pois os atributos selecionados têm um impacto mais significativo na classificação dos dados. No entanto, à medida que a quantidade de atributos selecionados aumenta, o algoritmo IG tem um conjunto maior de opções para considerar. Isso leva a uma diluição do impacto individual de cada atributo selecionado, já que a informação é distribuída entre um maior número de atributo. Por isso, é possível observar que o mesmo algoritmo apresenta desempenho inferior ao SU quando um número maior de atributos é usado para a construção da RCL.

Em particular, o algoritmo SU não foi afetado significativamente portanto é mais estável em função do número de atributos selecionados para compor o RCL: foram alcançadas as médias de 73,63%, 70,78% e 72,57%, respectivamente para os conjuntos com 10, 20 e 30 atributos. Em contraste, as soluções geradas a partir do RCL construído pelo algoritmo IG apresentam um desempenho decrescente quando um número maior de atributos é selecionado para a construção do RCL: foram alcançadas F1-Scores médias de 84,92%, 58,19% e 55,44%, a partir dos RCLs com 10, 20 e 30 atributos, respectivamente.

Por fim, observa-se que os RCLs gerados pelos algoritmos GR e RF alcançam uma maior probabilidade de geração de boas soluções com um número de dez *features* que são atributos compondo o RCL: os algoritmos GR e RF apresentam, respectivamente, F1-Scores médias de 62,98% e 70,61%. Já com vinte atributos ambos pioram significativamente, onde GR e RF apresentam, respectivamente, F1-Scores médias de 50,45% e 50,47%. Porém, RCLs construídas com trinta atributos apresentaram um desempenho superior às RCLs construídas com vinte atributos e inferior às RCLs geradas com dez atributos pelos algoritmos GR e RF, que alcançaram, respectivamente, médias de 55,68% e 55,92%. A adição de mais 20 atributos pode resultar na inclusão de características menos relevantes ou redundantes, o que pode introduzir ruído ou informações menos dis-

criminantes durante o processo de classificação. Isso dilui o impacto dos atributos mais importantes e causa uma diminuição no desempenho. Por outro lado, ao aumentar para 30 atributos, é possível que algumas das novas características sejam mais informativas e contribuam positivamente para a classificação. Esses atributos extras podem fornecer informações complementares que resultam em um melhor desempenho em comparação com as RCLs de 20 atributos.

4.2.1 Impacto do Tamanho do RCL

Conforme discutido anteriormente, o algoritmo escolhido para a implementação dos microsserviços não é o único fator que pode afetar a F1-Score das soluções geradas. O tamanho do RCL também afeta. Na Figura 6 são apresentadas as F1-Scores médias alcançadas pelos algoritmos implementados em cada microsserviço, agrupando-se pelo número de *features* que são os atributos configurado para compor o RCL de cada instância.

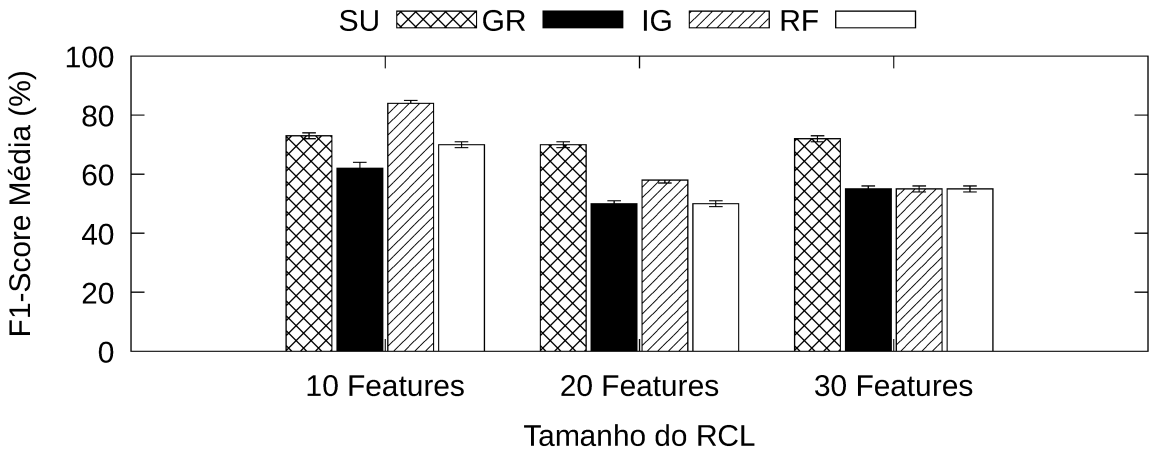


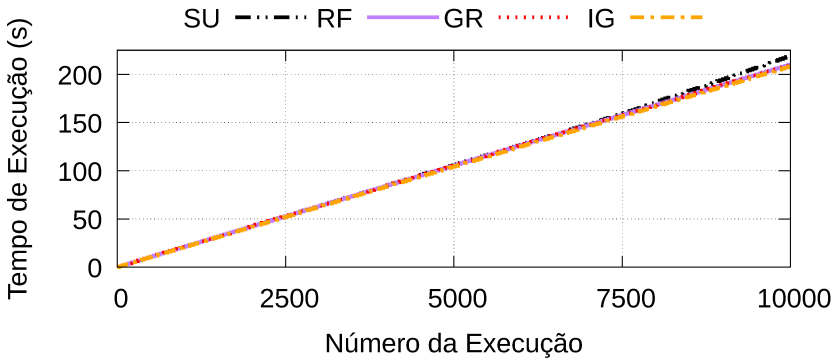
Figura 6 – F1-Score média alcançada por cada tamanho de RCL.
Fonte: Do Autor.

Por meio desta análise, é possível observar que RCLs de dez atributos apresentam de fato maior propensão de soluções geradas aleatoriamente apresentarem um melhor nível de qualidade. Em particular, a média dos 40.000 experimentos que geraram soluções a partir de RCLs de 10 atributos pelos quatro algoritmos experimentados foi a superior em termos de F1-Score, chegando a 73,04%.

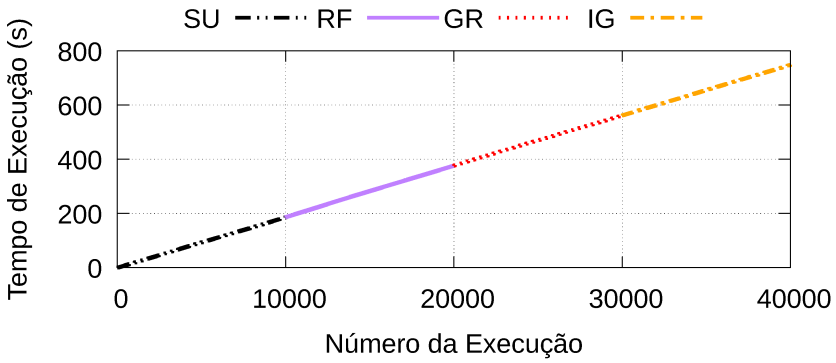
Por outro lado, a média das 40.000 soluções geradas a partir de RCLs com vinte atributos foi de apenas 57,47%. Similarmente, a média das 40.000 soluções geradas a partir de RCLs com trinta atributos foi 59,90%. Cabe destacar que o algoritmo SU obteve destaque na geração de RCLs com 30 atributos, o qual apresentou uma média de 72,57% enquanto os demais algoritmos apresentaram médias entre 55,44% e 55,92%.

4.3 Avaliação de Escalabilidade

Conforme ilustrado na Figura 7a, o processamento paralelo dos algoritmos revelou-se cerca de 3,4 vezes mais rápido do que a execução sequencial ilustrada na Figura 7b. Ao executar 10.000 instâncias para cada um dos algoritmos de seleção considerados, os tempos foram de 219,713s para SU, 210,821s para RF, 210,691s para GR e 208,773s para IG. Ou seja, a execução paralela é finalizada em 219,713s, quando a execução do SU é concluída. Em contraste, a execução sequencial dos algoritmos levou 748,368s.



(a) Resultados para a execução paralela.



(b) Resultados para a execução sequencial.

Figura 7 – Comparação entre execuções: Paralelo X Sequencial.
Fonte: Do Autor.

Os resultados apresentam grande relevância, pois revelam uma diferença significativa no tempo de execução em favor da abordagem de execução paralela adotada na arquitetura DRG. A execução paralela permite que várias tarefas sejam executadas simultaneamente, proporcionando benefícios notáveis em termos de escalabilidade e aceleração do tempo de execução dos microsserviços implementados.

Ao adotar o processamento paralelo, a arquitetura DRG se mostra altamente eficiente, pois permite que o trabalho seja distribuído entre vários processadores, possibilitando a execução simultânea de um conjunto de tarefas. Esse escalonamento eficiente dos recursos de processamento resulta em um menor tempo de execução global, à medida que mais processadores são alocados para lidar com as demandas computacionais. Dessa forma, é

possível obter um aumento significativo na velocidade de processamento, realizando as mesmas tarefas em um período de tempo ainda mais curto.

Essa capacidade de escalonamento eficiente do processamento paralelo na arquitetura DRG é particularmente benéfica em cenários onde a velocidade de processamento é essencial, como é o caso de aplicações em tempo real. Com a distribuição das tarefas entre vários processadores, é possível obter uma resposta mais rápida e um processamento contínuo, mesmo quando há um aumento na complexidade e na quantidade de dados a serem processados.

Além dos benefícios em relação à escalabilidade e ao tempo de execução, o processamento paralelo também oferece maior disponibilidade de recursos. Ao distribuir a carga de trabalho entre vários processadores, evita-se a sobrecarga de um único processador e reduz-se o risco de falhas ou interrupções no processamento. Isso resulta em um sistema mais robusto e confiável, capaz de lidar com volumes maiores de dados e demandas crescentes (TANENBAUM, 2007).

4.4 Discussão

Os estudos apresentados nesta seção fornecem *insights* valiosos sobre o desempenho dos algoritmos de seleção de atributos no contexto analisado. Com base nos resultados obtidos, o algoritmo IG se destaca como aquele com a melhor capacidade de gerar soluções de alta qualidade. No entanto, é importante ressaltar que essa afirmação é válida somente quando o tamanho da RCL é adequadamente parametrizado.

Nos experimentos realizados, a utilização de uma RCL composta pelos dez melhores atributos identificados pelo algoritmo IG resultou em uma média de F1-Score de 84,92% para as 10.000 soluções aleatórias testadas. Por outro lado, nos casos em que não se conhece o tamanho ideal da RCL, a abordagem mais indicada é o uso do algoritmo SU. Independentemente do número de atributos (*i.e.,* 30, 20 ou 10) usadas para compor a RCL, o algoritmo SU apresentou uma média superior a 70,7% para todas as 30.000 soluções geradas.

A variação nos resultados observada entre diferentes configurações e algoritmos em cada instância de microsserviço enfatiza a importância de ter múltiplas implementações disponíveis. Somente dessa forma é possível avaliar em tempo real o desempenho médio de cada algoritmo e otimizar o processo de seleção de atributos por meio da metaheurística GRASP-FS (QUINCOZES et al., 2022).

Além disso, é relevante destacar que o uso de microsserviços permitiu a execução paralela dos algoritmos de seleção de atributos. Essa abordagem resultou em uma aceleração do processo em cerca de 3,4 vezes, melhorando significativamente a eficiência

do sistema. A execução paralela dos microsserviços aproveita os recursos computacionais disponíveis, permitindo que várias tarefas sejam executadas simultaneamente e, assim, reduzindo o tempo total de processamento.

5 Conclusão

A adoção de métodos FS é fundamental para permitir um bom desempenho de detecção de IDSs. No entanto, um grande desafio é realizar uma seleção precisa de atributos em aplicações de tempo real: à medida que o padrão de tráfego muda, atributos irrelevantes podem ganhar mais valor e os atributos representativas podem perder relevância para os modelos de aprendizado de máquina ou no inglês, *machine learning*. Embora o GRASP-FS ofereça um *trade-off* entre alta qualidade e custo computacional, ele não é escalável.

Para enfrentar os desafios mencionados acima, foi proposto neste trabalho uma nova arquitetura orientada a microsserviços chamada DRG. Tal arquitetura possibilita a distribuição e paralelização do processamento da fase de construção do GRASP-FS. A arquitetura proposta aborda os problemas de escalabilidade ao desacoplar os microsserviços usando um agente de mensagens por meio do paradigma de publicação/assinatura. Como prova de conceito, foi implementada uma instância da arquitetura proposta por meio do *framework* Kafka, com quatro algoritmos de seleção de atributos, a saber: GR, IG, RF e SU. Esses componentes tem por finalidade a construção de RCLs e geração de soluções iniciais a partir das mesmas, as quais serão usadas como início da fase de busca local.

Os resultados obtidos revelam que o uso do IG apresenta a melhor capacidade de geração de soluções com alta qualidade quando se tem um tamanho de RCL adequado composta pelos dez melhores atributos do IG apresentando uma F1-Score média de 84,92%. Já para tamanhos aleatórios maiores que dez o uso do SU é mais indicado, visto o mesmo apresenta uma média superior a 70,7% para todas as 30.000 soluções geradas.

Em trabalhos futuros, planejamos implementar a arquitetura GRASP-FS completa, abrangendo a fase de Busca Local. Além disso, pretendemos experimentar em cenários mais desafiadores e avaliar outros algoritmos de seleção de atributos da categoria *wrapper* para implementar novos microsserviços. Por fim, pretende-se avaliar a resiliência da arquitetura proposta através da implementação de padrões tal como o *Retry* e *Circuit Breaker* (COSTA et al., 2022).

Referências

ABREU, D. de; CARVALHO, I.; ABELéM, A. J.; MENASCHé, D.; LEãO, R. M.; SILVA, E. Seleção de características por clusterização para melhorar a detecção de ataques de rede. In: **XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2020. p. 295–308. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/12290>. Citado na página 17.

AFTAB, M. A.; HUSSAIN, S. S.; ALI, I.; USTUN, T. S. Iec 61850 based substation automation system: A survey. **International Journal of Electrical Power & Energy Systems**, Elsevier, v. 120, p. 106008, 2020. Citado na página 16.

Ajeet Singh Raina and Johan Giraldo. **Kickstart Your Spring Boot Application Development.** 2022. Disponível em: <https://www.docker.com/blog/kickstart-your-spring-boot-application-development/>. Acesso em: 18 de Novembro 2022. Citado na página 30.

ANDERSON, C. Docker [software engineering]. **IEEE Software**, IEEE, v. 32, n. 3, p. 102–c3, 2015. Citado na página 30.

AXELSSON, S. Intrusion detection systems: A survey and taxonomy. Citeseer, 2000. Citado na página 16.

BACE, R. G.; MELL, P. et al. Intrusion detection systems. US Department of Commerce, Technology Administration, National Institute of . . . , 2001. Citado na página 16.

CARVALHO, D.; QUINCOZES, V. E.; QUINCOZES, S. E.; KAZIENKO, J. F.; SANTOS, C. R. P. dos. BG-IDPS: Detecção e prevenção de intrusões em tempo real em switches eBPF com o filtro de pacotes berkeley e a metaheurística GRASP-FS. In: SBC. **XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**. [S.l.], 2022. p. 139–152. Citado 4 vezes nas páginas 11, 12, 20 e 21.

COSTA, T.; VASCONCELOS, D.; ADERALDO, C.; MENDONçA, N. Avaliação de desempenho de dois padrões de resiliência para microsserviços: Retry e circuit breaker. In: **XL Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. Porto Alegre, RS, Brasil: SBC, 2022. p. 517–530. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/21194>. Citado na página 38.

DIEZ-PASTOR, J. F.; GARCIA-OSORIO, C.; RODRIGUEZ, J. J.; BUSTILLO, A. GRASP forest: a new ensemble method for trees. In: SPRINGER. **International Workshop on Multiple Classifier Systems**. [S.l.], 2011. p. 66–75. Citado 2 vezes nas páginas 20 e 21.

ESKANDARI, S.; JAVIDI, M. M. Online streaming feature selection using rough sets. **International Journal of Approximate Reasoning**, Elsevier, v. 69, p. 35–57, 2016. Citado na página 17.

ESSEGHIR, M. A. Effective wrapper-filter hybridization through grasp schemata. In: PMLR. **Feature selection in data mining**. [S.l.], 2010. p. 45–54. Citado 4 vezes nas páginas 12, 19, 20 e 23.

FortiGuard Labs. **Relatório ciberataque no Brasil em 2021 feito pelo FortiGuard Labs.** 2021. Disponível em: <https://www.fortinet.com/br/corporate/about-us/newsroom/press-releases/2022/fortiguard-labs-relatorio-ciberataques-brasil-2021>. Acesso em: 20 de Outubro 2022. Citado na página 11.

GARG, N. **Apache kafka**. [S.l.]: Packt Publishing Birmingham, UK, 2013. Citado na página 30.

GUYON, I.; ELISSEEFF, A. An introduction to variable and feature selection. **Journal of machine learning research**, v. 3, n. Mar, p. 1157–1182, 2003. Citado na página 17.

HONOVAN, P. R.; LADEIRA, R. W. S.; BRAGA, A. P. Análise de métodos construtivos, busca local e metaheurísticas para o problema de seleção de características. In: Bastos Filho, C. J. A.; POZO, A. R.; LOPES, H. S. (Ed.). **Anais do 12 Congresso Brasileiro de Inteligência Computacional**. Curitiba, PR: ABRICOM, 2015. p. 1–6. Citado 2 vezes nas páginas 12 e 20.

KANAKARAJAN, N. K.; MUNIASAMY, K. Improving the accuracy of intrusion detection using gar-forest with feature selection. In: SPRINGER. **Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA) 2015**. [S.l.], 2016. p. 539–547. Citado 3 vezes nas páginas 12, 20 e 21.

KANNAN, S. S.; RAMARAJ, N. A novel hybrid feature selection via symmetrical uncertainty ranking based local memetic search algorithm. **Knowledge-Based Systems**, Elsevier, v. 23, n. 6, p. 580–585, 2010. Citado na página 18.

KIRA, K.; RENDELL, L. A. A practical approach to feature selection. In: **Machine learning proceedings 1992**. [S.l.]: Elsevier, 1992. p. 249–256. Citado 2 vezes nas páginas 18 e 19.

KONONENKO, I. et al. Estimating attributes: Analysis and extensions of relief. In: CITESEER. **ECML**. [S.l.], 1994. v. 94, p. 171–182. Citado na página 18.

KULLBACK, S.; LEIBLER, R. A. On information and sufficiency. **The annals of mathematical statistics**, JSTOR, v. 22, n. 1, p. 79–86, 1951. Citado na página 17.

KUSH, N. S.; AHMED, E.; BRANAGAN, M.; FOO, E. Poisoned goose: Exploiting the goose protocol. In: AUSTRALIAN COMPUTER SOCIETY. **Proceedings of the Twelfth Australasian Information Security Conference (AISC 2014)[Conferences in Research and Practice in Information Technology, Volume 149]**. [S.l.], 2014. p. 17–22. Citado na página 16.

LI, Y.; SUN, H. Research and design of docker technology based authority management system. **Computational Intelligence and Neuroscience: CIN**, Hindawi Limited, v. 2022, 2022. Citado na página 30.

NEWMAN, S. **Building microservices**. [S.l.]: "O'Reilly Media, Inc.", 2021. Citado 2 vezes nas páginas 12 e 27.

PASTOR, J. F. D.; OSORIO, C. G.; RODRIGUEZ, J. J. Tree ensemble construction using a grasp-based heuristic and annealed randomness. **Information Fusion**, Elsevier, v. 20, p. 189–202, 2014. Citado 2 vezes nas páginas 20 e 21.

QUILAN, J. R. Decision trees and multi-valued attributes. **Machine intelligence**, p. 305–318, 1988. Citado na página 18.

QUINCOZES, S. E.; ALBUQUERQUE, C.; PASSOS, D.; MOSSÉ, D. A survey on intrusion detection and prevention systems in digital substations. **Computer Networks**, Elsevier, v. 184, p. 107679, 2021. Citado 2 vezes nas páginas 30 e 31.

_____. ERENO: An Extensible Tool For Generating Realistic IEC-61850 Intrusion Detection Datasets. In: SBC. **Concurso de Teses e Dissertações do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg)**. [S.l.], 2022. p. 1–8. Citado 3 vezes nas páginas 12, 30 e 31.

QUINCOZES, S. E.; MOSSÉ, D.; PASSOS, D.; ALBUQUERQUE, C.; OCHI, L. S.; SANTOS, V. F. dos. On the performance of grasp-based feature selection for CPS intrusion detection. **IEEE Transactions on Network and Service Management**, IEEE, v. 19, n. 1, p. 614–626, 2021. Citado 9 vezes nas páginas 11, 12, 20, 22, 23, 24, 25, 26 e 28.

QUINCOZES, S. E.; PASSOS, D.; ALBUQUERQUE, C.; OCHI, L. S.; MOSSÉ, D. Grasp-based feature selection for intrusion detection in cps perception layer. In: IEEE. **2020 4th Conference on cloud and internet of things (CIoT)**. [S.l.], 2020. p. 41–48. Citado 5 vezes nas páginas 19, 20, 21, 23 e 25.

QUINCOZES, S. E.; PASSOS, D.; ALBUQUERQUE, C.; MOSSÉ, D.; OCHI, L. S. An extended assessment of metaheuristics-based feature selection for intrusion detection in cps perception layer. **Annals of Telecommunications**, Springer, p. 1–15, 2022. Citado 4 vezes nas páginas 11, 20, 22 e 36.

QUINLAN, J. R. Induction of decision trees. **Machine learning**, Springer, v. 1, p. 81–106, 1986. Citado na página 17.

_____. Program for machine learning. **C4. 5**, Morgan Kaufmann Pub, 1993. Citado na página 18.

REDDY, K. S. P. **Beginning Spring Boot 2: Applications and microservices with the Spring framework**. [S.l.]: Apress, 2017. Citado na página 30.

ROBNIK-ŠIKONJA, M.; KONONENKO, I. Theoretical and empirical analysis of relieff and rrelieff. **Machine learning**, Springer, v. 53, n. 1, p. 23–69, 2003. Citado na página 18.

SALZBERG, S. L. **C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993**. [S.l.]: Kluwer Academic Publishers, 1994. Citado na página 17.

SILVA, E. F. C.; NAVES, N.; QUINCOZES, S. E.; QUINCOZES, V. E.; KAZIENKO, J. F.; CHEIKHROUHOU, O. GDLS-FS: Scaling Feature Selection for Intrusion Detection with GRASP-FS and Distributed Local Search. In: UFJF. **37th International Conference on Advanced Information Networking and**

Applications (AINA-2023). [S.l.], 2023. p. 1–12. (to appear – paper accepted in AINA 2023). Citado 2 vezes nas páginas 22 e 28.

TANENBAUM, A. S. **Distributed systems principles and paradigms**. [S.l.: s.n.], 2007. Citado na página 36.

YUSTA, S. C. Different metaheuristic strategies to solve the fs problem. **Pattern Recognition Letters**, Elsevier, v. 30, n. 5, p. 525–534, 2009. Citado na página 19.

Apêndices

APÊNDICE A – Artigo Submetido ao SBSeg

O presente trabalho foi submetido no SBSEG (Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais), explorando a **Fase Construtiva** do GRASP-FS. Como se trata de um trabalho não publicado na presente data, o mesmo foi omitido dos anexos, mantendo-se apenas a primeira página.

Uma Arquitetura Orientada a Microsserviços para a Filtragem de Features e Construção de RCLs no GRASP-FS Aplicada a um Cenário de Sistemas Ciber-Físicos

Nícolas Naves R. Faria[1], Silvio E. Quincozes[1], Juliano F. Kazienko[2], Vagner E. Quincozes[3], Gabriel Oliveira[1] e Estêvão F. C. Silva[1]

[1]FACOM – Universidade Federal de Uberlândia (UFU), Monte Carmelo, Brasil

[2]CTISM – Universidade Federal de Santa Maria (UFSM), Santa Maria, Brasil

[3]Universidade Federal do Pampa (UNIPAMPA), Alegrete, Brasil

{nicolasnaves, sequincozes, estevaof10, gabrieloliveirasouza620}@ufu.br

vagnerquincozes.aluno@unipampa.edu.br, kazienko@redes.ufsm.br

Abstract. *This paper presents a scalable microservices-oriented architecture, called Distributed RCL Generator (DRG), to decouple and parallelize the construction phase in the Greedy Randomized Adaptive Search Procedure for Feature Selection (GRASP-FS) metaheuristic in Feature Selection (FS) for intrusion detection. In GRASP-FS, the construction phase generates initial feature subsets that are optimized in the local search phase. As a proof-of-concept based on the Kafka framework and four FS algorithms, we demonstrated that the adopted algorithm strategy impacts the intrusion detection F1-Score in a Cyber-Physical Systems scenario ranging from 50.47% to 84.92%. Finally, we show that parallel processing can accelerate the construction phase about 3.4 times.*

Resumo. *Este artigo apresenta uma arquitetura escalável orientada a microsserviços, chamada Distributed RCL Generator (DRG), para desacoplar e paralelizar a fase de construção na metaheurística Greedy Randomized Adaptive Search Procedure for Feature Selection (GRASP-FS) na Seleção de Features (FS) para detecção de intrusões. No GRASP-FS, são gerados conjuntos de features que são otimizadas na sua fase de busca local. Através de uma prova de conceito baseada no framework Kafka e quatro algoritmos de FS, demonstrou-se que os algoritmos usados na estratégia de construção do RCL impactam na F1-Score média da detecção de intrusões em um cenário de Sistemas Ciber-Físicos de 50,47% até 84,92%. Ademais, constatou-se que o processamento paralelo pode acelerar a fase de construção em cerca de 3,4 vezes.*

1. Introdução

Os Sistemas Ciber-Físicos, do inglês, *Cyber-Physical Systems* (CPSs), que integram o mundo cibernético e componentes do mundo físico, são a base para o avanço de múltiplos setores como o de saúde, cidades inteligentes, além daqueles com infraestrutura crítica como é o caso da produção e distribuição de energia. Nesse cenário — que está em rápido crescimento — a importância da cibersegurança vem aumentando de forma constante [Quincozes et al. 2021b]. De acordo com os dados levantados por

APÊNDICE B – Artigo Publicado no AINA (*Advanced Information Networking and Applications*), explorando a **Fase de Busca Local** do GRASP-FS

GDLS-FS: Scaling Feature Selection for Intrusion Detection with GRASP-FS and Distributed Local Search

Estêvão F. C. Silva, Nícolas Naves, Silvio E. Quincozes, Vagner E. Quincozes, Juliano F. Kazienko, and Omar Cheikhrouhou

Abstract This paper presents a scalable microservice-oriented architecture, called Distributed LS (DLS), for enhancing the Local Search (LS) phase in the Greedy Randomized Adaptive Search Procedure for Feature Selection (GRASP-FS) metaheuristic. We distribute the DLS processing among multiple microservices, each with a different responsibility. These microservices are decoupled because they communicate with each other by using a message broker through the publish and subscribe paradigm. As a proof-of-concept, we implemented an instance of our architecture through the Kafka framework, two neighborhood structures, and three LS algorithms. These components look for the best solution which is published in a topic to the Intrusion Detection System (IDS). Such a process is iterated continuously to improve the solution published, providing IDS with the best feature selection solution at the end of the search process with scalability and time reduction. Our results show that using RVND may take only 19.53% of the time taken by VND. Therefore, the RVND approach is the most efficient for exploiting parallelism in distributed architectures.

Estêvão F. C. Silva
TQI Tecnologia, Uberlândia – Brazil. e-mail: estevao.silva@tqi.com.br
FACOM/UFU, Uberlândia – Brazil. e-mail: estevaof10@ufu.br

Nícolas Naves
Facom/UFU, Uberlândia – Brazil. e-mail: nicolasnaves@ufu.br

Silvio E. Quincozes
FACOM/UFU, Monte Carmelo – Brazil. e-mail: sequincozes@ufu.br

Vagner E. Quincozes
UNIPAMPA, Alegrete – Brazil. e-mail: vagnerquincozes.aluno@unipampa.edu.br

Juliano F. Kazienko
CTISM/UFSM, Santa Maria – Brazil. e-mail: kazienko@redes.ufsm.br

Omar Cheikhrouhou
CES Lab, University of Sfax — Tunisia. e-mail: omar.cheikhrouhou@isetsf.rnu.tn

1. Introduction

Nowadays, information security is a major concern for cybernetic applications across multiple domains, including billions of devices worldwide. A to the MCA Department, 98% of applications published on the web are vulnerable to cyberattacks in 2019 [25]. This problem is still more serious when the critical infrastructure (*e.g.,* gas and oil platforms, power grid, electrical substations) is affected: in the worst cases, human lives may be at risk [21][3][16]. In this context, the information security area studies and applies methods to mitigate computer systems and communication network vulnerabilities, such as Intrusion Detection Systems (IDSs). However, even with the proper preventive measures, attackers have increasingly demonstrated the ability to control them and succeed in their attacks. Therefore, to properly detect the intruders' behavior it is essential to employ sophisticated IDSs (*e.g.,* using machine learning) and feed them with representative information (*e.g.,* selected features)[10].

To provide representative information for IDSs based on machine learning, employing preprocessing techniques, such as Feature Selection (FS) is necessary. Such techniques aim to select meaningful information to feed the IDSs and help it in distinguishing between legitimate and malicious traffic [9]. There are different FS approaches, each with its advantages and weakness. The filter-based FS attempts to estimate the most relevant features individually using statistical methods. Such estimation is lightweight but does not always reflect performance improvements for machine learning algorithms. On the other hand, Wrapping-based FS employs a machine learning algorithm to assess multiple combinations of features together. Wrapping-based methods are more assertive, however, they are more computationally expensive, and exploring all combinations of features may be impracticable. Therefore, using hybrid metaheuristic algorithms that combine Wrapping and Filter is an alternative to reaching suboptimal solutions in a feasible time [18].

The metaheuristic Greedy Randomized Adaptive Search Procedure for Feature Selection (GRASP-FS) generates *greedy* and *random* solutions using filter-based FS and employs a Local Search (LS) procedure using wrapping-based FS to optimize the initial solution. However, the original GRASP-FS proposal [19] needs to parameterize manually, and its execution is sequential in a single monolithic application. As the traffic pattern varies with the application context, finding the best algorithms to compose the LS phase in GRASP-FS may require extensive analysis and comparison of results. Furthermore, in case of changes in the network traffic profile, the whole process needs to be performed again. Therefore, a crucial challenge of this approach is the lack of dynamicity and scalability in the feature selection process.

In this work, we propose a microservice-oriented architecture for GRASP-FS with Distributed LS (GDLS-FS) algorithms. The proposed architecture employs the publish/subscribe paradigm to decouple the LS algorithms and address the scalability issues in metaheuristics. To evaluate our approach, we compare the VND and RVND neighborhood structures and the IWSSR, IWSS, and Bit-Flip local searches.

2. Background

In this section, we present algorithms and concepts related to the GRASP-FS metaheuristic, local search algorithms, and neighborhood structures.

The GRASP-FS [18] is an application of the GRASP metaheuristic [20] for Feature Selection (FS) purposes. The main goal of the GRASP-FS is to select a representative subset of features, discarding those that do not aggregate value to classifier models. GRASP-FS can be used in the intrusion detection context to select relevant information to represent a malicious activity. The GRASP-FS metaheuristic is divided into two main steps: (i) one for generating initial solutions and (ii) another for performing local movements and optimizing the initial solutions.

The first step of GRASP-FS is the construction phase, in which an initial reduction of the total amount of features available is performed. Such reduction is typically based on a process that employs statistical methods such as Information Gain (IG) or Gain Ratio (GR) to rank the available features according to their representativeness. After ranking, the *N* top-ranked features are used to compose a Restricted Candidate List (RCL). From the RCL, *greedy* and *random* solutions are generated. In this work, we call the generated feature subset as *initial solution*. These solutions are considered greedy because they consider only the statistical methods employed to filter their representativeness. They are random solutions because the selected features to compose such solutions are picked up at random from the RCL.

In the second step of GRASP-FS, the *initial solutions* are optimized through local movements performed by an LS algorithm. These LS algorithms are coordinated by a Neighborhood Structure. The feature subset optimization relies on an objective function such as F1-Score (Eq. 1) to compare the found solutions. The F1-Score metric represents the harmonic mean of Precision (Eq. 2) and Recall (Eq. 3) [18].

$$F1Score = 2 \times \frac{Precison \times Recall}{Precision + Recall} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

The GRASP-FS metaheuristic was designed to work as a monopolistic application. The first version of GRASP-FS [19] implements a single and simple LS method, whereas its improved versions [18][17] support different neighbor structures and LS algorithms. Variable Neighborhood Descent (VND) [7] and Random Variable Neighborhood Descent (RVND) [23] are neighbor structures that were employed to handle the LS algorithms in GRASP-FS.

The VND walks on the initial solutions neighborhoods through a list of LS algorithms and moves from there to new neighborhoods after a number of iterations or maximum time criteria are reached. If improvements have been made, it moves to the first LS algorithm. Otherwise, the next LS algorithm in the list is used to explore

a different neighborhood. According to [14], the resulting hybrid GRASP/VND algorithm is simple and fast, reaching solutions equal to or better than those obtained by the more complex existing procedures. The RVND neighborhood structure operates similarly to VND, but instead of going through the LS list sequentially, random movements are made every a threshold is reached.

Regardless of the neighborhood structure adopted, the LS algorithms are responsible to define the movements from one solution to one of its neighbors. The Incremental Wrapper Subset Selection (IWSS) is an algorithm in which the RCL features are pickup incrementally to compose the neighbor solutions. The added features are kept if and only if they present an improvement to the neighbor solution (*e.g.,* a better F1-Score than the previous one) [2]. The Incremental Wrapper Subset Selection with replacement (IWSSr) algorithm is a variation of IWSS which also performs sequential movements, but rather than only adding it also attempts to perform replacements between the solution and RCL features [11].

Finally, Bit-Flip (BF) is the simplest LS procedure already implemented within GRASP-FS. It operates by performing iteratively exchanges among features from the RCL and the selected to compose the initial solution [19].

3. Related Works

The main related works in the current literature can be divided into three categories: those that adopt the local search (i) within GRASP metaheuristic, (ii) without GRASP, and (iii) a distributed/parallel architecture. In this section, we discuss these works. Table 1 summarizes the works. Note that, to the best of our knowledge, this work is the only one that combines all four characteristics.

Ref.	GRASP	Feature Selection	Local Search	Distributed Local Search
[22, 19, 5, 11]	√	√	×	×
[12]	√	√	√	×
[13]	×	√	√	×
[6]	×	√	√	Parallel
[4]	×	×	√	√
[1]	×	×	√	Parallel
This work	√	√	√	√

Table 1 Related Works.

The GRASP-based metaheuristics are found in the literature in applications such as feature selection in cardiovascular disease classification [22], feature selection in high-dimensional datasets, and feature selection to detect and prevent instructions in Cyber-Physical Systems [18] and real-time applications [5]. Subanya et al. [22]

uses an artificial colony of bees for cardiovascular disease classification and reduces computational classification complexity by eliminating redundant resources. The authors implement the Support Vector Machine (SVM) classifier to perform the wrapper-based assessment in the GRASP LS. Other work [5] presents an architecture for real-time intrusion detection and prevention in eBPF switches. The authors use asynchronously optimized models through the GRASP-FS metaheuristic. Finally, an important remark is that the aforementioned works do not address the local search distribution in order to scale feature selection services.

A further work that adopts the GRASP metaheuristic is the [12]. The authors propose an approach to feature selection using GRASP with local search based on Simulated Annealing (SA) algorithm. The main goal is to reduce the classification processing time based on reducing the number of features and new parameters embedded in SA. Thus, the authors argue that their proposal is scalable. Nevertheless, the authors in fact do not use distribution and parallelism to reach scalability. Other work [11] employ GREselect a subset of features with a high-dimensional dataset. The authors apply the FICA and IWSSr algorithms in the wrapper phase to incorporate the selection of relevant features. The results indicate an improvement in terms of accuracy in the wrapper phase.

Another local search approach consists in to perform LS without GRASP. Nekkaa et al. [13] propose a local hybrid search based on harmony search (HSA) and stochastic search (SLS) for classification and feature selection. HSA-SLS algorithms are combined with a Support Vector Machine (SVM) classifier to send optimized parameters. Thus, according to the authors, it is possible to maximize the classification accuracy of the SVM. Another proposal [6] implements LS and data sorting using SVM. The goal is select and classifies features optimally. In addition, the proposal seeks to reduce the use of resources. The authors suggest diversifying the search space through multiple LS algorithms running in parallel and periodically sharing information.

Distribution and parallelism are the focus of another works in literature. Cai et.al [4] presents a distributed planning approach. The proposal allows robots to build a team plan collaboratively plan via distributed local search. The robots propose local operations based on their trajectories. Computing and communication requirements reduce using Lazy Search and Greedy Warm. Already, Araujo et.al [1] propose to use GPU for parallel processing of LS neighborhood structures instances in order to minimize the latency compared to sequential workflows. However, different from that work, a broader scaling strategy is adopted in this work based on LS execution through microservices. It allows distribution – and scaling – in different scenarios ranging from clusters to GPUs, as proposed by [1]. Particularly, our work proposes LS scaling to solve the feature selection problem for IDSs.

4. The GDLS-FS Architecture

The GRASP-FS with Distributed LS (GDLS-FS) is a novel microservice-oriented architecture that enables the distribution and parallelism of microservices to solve the scalability issues of LS algorithms in the GRASP-FS metaheuristic. GRASP-FS employs machine learning algorithms to process potentially large volumes of data in its LS phase. Therefore, these algorithms may cause overhead for processing and slowness depending on the amount of processed data: this is the main bottleneck of GRASP-FS. In this context, our approach gives scalability to the LS process, speeding up the convergence to the best solution. To reach more scalability, we distribute the LS processing among multiple microservices, each with a different responsibility. These microservices communicate with each other by using the *publish/subscribe* paradigm. Our proposed architecture is illustrated in Figure 1.

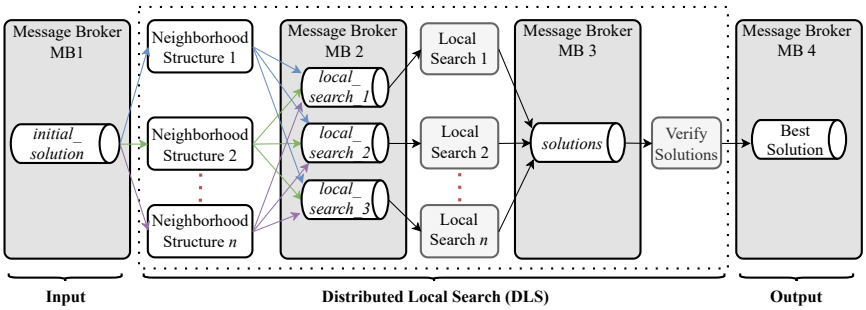


Fig. 1 The proposed micro-service oriented architecture.

Firstly, there is a Message Broker `MB1` which is responsible for initializing the proposed LS process by providing an initial solution to be optimized. In GRASP-FS the initial solutions would be provided by specific algorithms, implemented in its constructive phase (the first phase of GRASP-FS), which combines a greedy and random procedure. In this work, we focus on the LS only (the second phase of GRASP-FS). Therefore, the component responsible for providing initial solutions should be transparent to our architecture. The process starts whenever an already computed initial solution is given.

The `Neighborhood Structure` microservices are responsible for handling the LS algorithms. This includes scheduling and feeding them with the initial solution. Therefore, each `Neighborhood Structure` microservice subscribes to the *initial_solution* topic in `MB1` and publishes the received initial solutions, according to its implementation, into one or multiple *local_search* topics in `MB2`. For each initial solution consumed, the `Neighborhood Structure` microservice publishes, iteratively, multiple times until it reaches a predefined threshold (*e.g.,* number of iterations, maximum running time, etc.).

The `LS` microservices implement a set of *local movements* to derive neighbor solutions (*i.e.,* feature subsets with few features different from the initial solutions) iteratively until reaching a predefined threshold. Every solution generated is assessed by a wrapping-based approach — which employs a machine learning algorithm. Each microservice may implement a distinct LS procedure. Alternatively, they may be deployed as replicas with redundant implementation for attaining greater scalability due to the smaller overhead of each microservice. Considering replicas for this microservice is relevant since it uses machine learning algorithms to assess the quality of each generated neighbor solution, which may cause the main bottleneck of the proposed architecture. The generated solutions are published into the *solutions* topic of `MB3`, which is consumed by the interested microservices, such as the `VerifySolutions` or the `Neighborhood Structure` microservices that need to receive feedback to perform their next decisions.

Finally, the `VerifySolutions` microservice subscribes to the *solutions* topic from `MB3` to process and compare all solution founds, regardless of the LS algorithm used to find such solutions. In a real and production scenario, a real-time IDS would subscribe to the *solutions* topic to receive optimized solutions whenever they are generated by our architecture. This procedure avoids the IDS being unnecessarily re-trained when no updated solutions are available and keeps the IDSs continuously updated with the current best-known feature subset.

5. Proof-of-Concept: Implementation

To evaluate the novel approach presented in this work, we implemented an instance of the proposed GDLS-FS architecture as a proof-of-concept to compare its performance with the traditional GRASP-FS metaheuristic implementation [19] and its extended version [18]. The implemented architecture components are depicted in Figure 2 and discussed below.

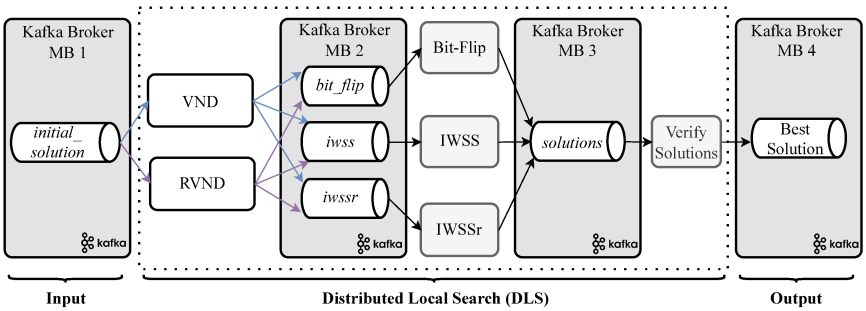


Fig. 2 Implementation of the Distribute Local Search (DLS) as a part of GDLS-FS.

Our implementation has three main categories of microservices: Neighborhood Structures (*i.e.,* `VND` and `RVND` microservices), Local Searches (*i.e.,* `Bit-Flip`, `IWSS`, and `IWSSr` microservices), and the `Verify Solutions` microservice. Whereas the two formers are aimed to handle the local movements through the solutions' neighborhood, the latter is responsible for delivering the optimized solutions to the subscribers IDSs interested in that information. As classifier algorithms for wrapping-based evaluations within the LS microservices, we used J48 algorithm with support of the Weka machine learning workbench [8]. To implement the message brokers `M1`, `M2`, `M3`, and `M4`, we adopted the Apache Kafka [24].

To assess the implemented instance of GDLS-FS in light of a critical application, we generate 2,000 samples of traffic from electrical substation networks through the ERENO Framework [15]. ERENO makes it possible to model and simulate cyber-attacks to the electrical power grid infrastructure.

As the scope of this work is the DLS as a part of the GRASP-FS metaheuristic, we abstract the GRASP-FS' construction phase by using a fixed initial solution `S = {1, 2, 3, 4, 5}` as input to DLS process. This allows us to assess the DLS process with a unified input as a starting point. The remaining features are added to `RCL = {6, 7, ..., 69}` for enabling the LS algorithms to walk through the neighbor solutions.

To start the DLS process, S is published into *initial_solution* topic in `MB1` and consumed by VND and RVND. From this point, messages are sent iteratively to LS microservices by using S to initialize them — which microservice is initialized depends on the internal logic of the neighborhood structure. For each message published by VND and/or RVND, IWSS and IWSSr explore all *RCL* features sequentially and *Bit-Flip* performs $N = 100$ neighbor solutions. We publish $M = 10$ messages for each neighborhood structure. Therefore, for each message published by a neighborhood structure, the explored neighborhood has:

- $M \times N$ neighbors generated by *Bit-Flip*;
- $M \times |RCL|$ neighbors generated by IWSS, where $|RCL|$ is the RCL length; and
- $M \times |RCL| + R$ neighbor solutions generated by IWSSr, where R is the number of replace movements.

6. Experiments and Results

In this section, we present and discuss the found results. Firstly, we show how the Neighborhood Structure Microservices performed when publishing messages to the LS subscribers (Section 6.1). Then, we analyzed one iteration from each LS algorithm to figure out how they performed (Section 6.2).

6.1 Neighborhood Structure Microservices

Figure 3 depicts the sequential iterations of local search microservices. We used a logarithmic timestamp in the chart to enable the visualization of all LS algorithms, as IWSSr took too much time in comparison to IWSS and Bit-Flip. The entire processing of VND and LS algorithms took 3,205,075 *ms* (53,4 minutes). Note that at the end of the execution, there are still novel solutions being reached by IWSS and Bit-Flip. This occurs because the choice of which LS will be requested by VND depends upon the results of the previous LS it requested.

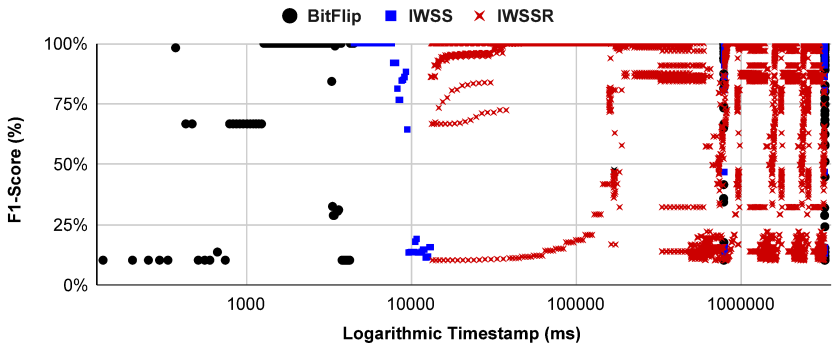


Fig. 3 VND iterations

One of the key advantages of using RVND rather than VND in a distributed architecture is the expanded exploration of parallelism. Figure 4 depicts the random and parallel iterations of local search microservices.

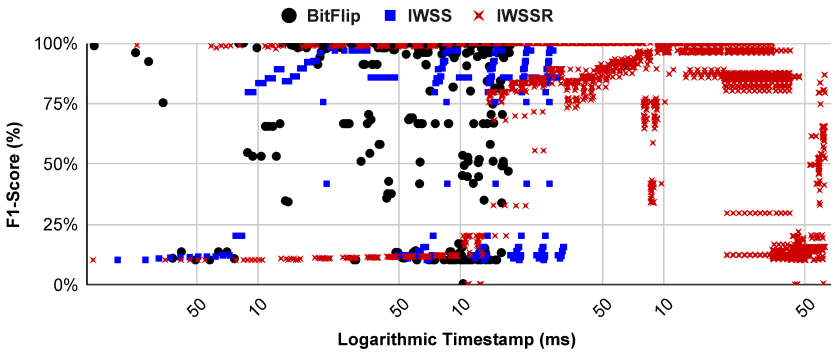


Fig. 4 RVND iterations.

The entire processing of RVND and LS algorithms took 626,096 *ms* (10,43 minutes). RVND is faster than VND because it do not need a feedback from the LS. Therefore, RVND can explore better the parallel processing among the LS microservices. In this scenario, Bit-Flip took only 17,650 *ms* and IWSS in 32,354 *ms*. The graph's logarithmic timestamp enables a better visualization of all LS algorithms.

6.2 Local Search Microservices

Figure 5 depicts the F1-Score of the solutions found within one iteration for each LS microservice. Bit-Flip reaches high and low F1-Scores more randomly because of their random movements. In contrast, IWSS has a more stable behavior which follows a sequential pattern. The IWSS performed worse at the end of the processing because those RCL features were less representative in the assessed.

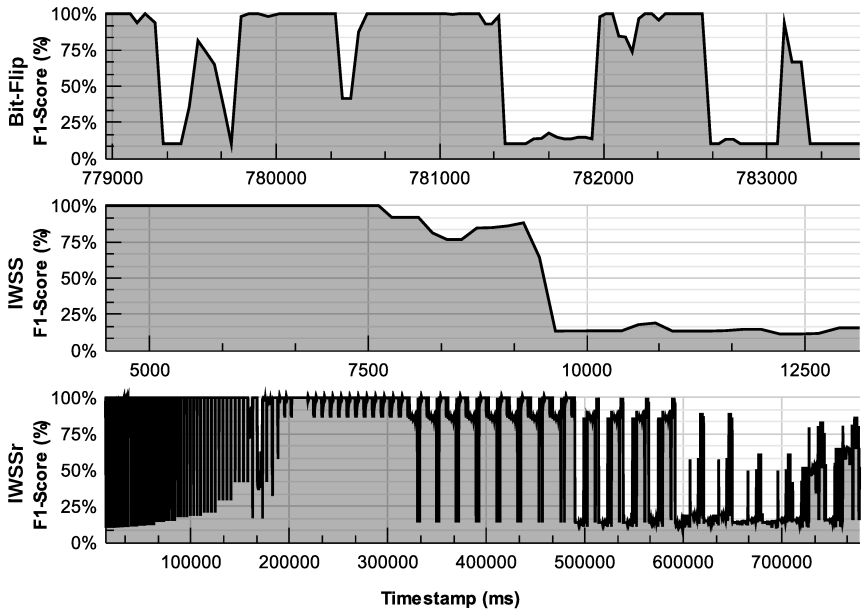


Fig. 5 Comparison of the behavior of Bit-Flip, IWSS, and IWSSr.

Finally, IWSSr presented a very unstable behavior because for each *add* movement it tested the replacement of the current solution features by another from the RCL. This may be the main reason for the high and low F1-Scores found along the processing. All methods reached a 100% F1-Score in the experimented dataset.

7 Conclusion

Adopting FS methods are fundamental to enable good detection performance for IDSs. However, a major challenge is performing an accurate feature selection in real-time applications: as the traffic pattern changes, irrelevant features may gain more value and the representative features may loose relevance for the machine learning models. Although GRASP-FS offers a trade-off between the high-quality and computational expensiveness, it is not scalable.

To address the aforementioned challenges, we presented the GDLS-FS: a novel microservices-oriented architecture for GRASP-FS with DLS algorithms. The proposed architecture addresses the scalability issues by decoupling the LS microservices through the publish/subscribe paradigm. As a proof of concept, we implement an instance of our architecture using the Kafka framework, two neighborhood structures (i.e., VND and RVND), and three LS algorithms (i.e., IWSS, IWSSR, and Bit-Flip). Our results show that using RVND may takes only 19.53% of the time taken by VND. Therefore, the RVND approach is the most efficient for exploiting parallelism in distributed architectures.

In future work, we plan to implement the full GRASP-FS architecture, covering the construction phase. Besides, we intend to experiment in more challenging scenarios and to assess other algorithms to implement our microservices. Finally, our future experiments will use a larger dataset to assess the scalability of the proposed solution under massive data processing.

Acknowledgments

This work is financially supported by TQI Tecnologia.

References

1. ARAUJO, R. P., COELHO, I. M., AND MARZULO, L. A. J. A multi-improvement local search using dataflow and GPU to solve the minimum latency problem. *Parallel Computing 98* (2020), 102661.
2. BERMEJO, P., GAMEZ, J. A., AND PUERTA, J. M. Incremental wrapper-based subset selection with replacement: An advantageous alternative to sequential forward selection. In *2009 IEEE Symposium on Computational Intelligence and Data Mining* (2009), pp. 367–374.
3. BORGIANI, V., MORATORI, P., KAZIENKO, J. F., TUBINO, E. R. R., AND QUINCOZES, S. E. Toward a distributed approach for detection and mitigation of denial-of-service attacks within industrial internet of things. *IEEE Internet of Things Journal 8*, 6 (2021), 4569–4578.
4. CAI, X., SCHLOTFELDT, B., KHOSOUSSI, K., ATANASOV, N., PAPPAS, G. J., AND HOW, J. P. Non-Monotone Energy-Aware Information Gathering for Heterogeneous Robot Teams. In *2021 IEEE Int. Conf. on Robotics and Automation (ICRA)* (2021), IEEE, pp. 8859–8865.
5. CARVALHO, D., QUINCOZES, V. E., QUINCOZES, S. E., KAZIENKO, J. F., AND SANTOS, C. R. P. BG-IDPS: Detecção e prevenção de intrusões em tempo real em switches eBPF com o filtro de pacotes berkeley e a metaheurística GRASP-FS. In *XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais - SBSeg* (2022), SBC, pp. 139–152.
6. CURA, T. Use of support vector machines with a parallel local search algorithm for data classification and feature selection. *Expert Systems with Applications 145* (2020), 113133.
7. HANSEN, P., MLADENOVIĆ, N., BRIMBERG, J., AND PÉREZ, J. A. M. Variable neighborhood search. In *Handbook of metaheuristics*. Springer, 2019, pp. 57–97.
8. HOLMES, G., DONKIN, A., AND WITTEN, I. H. Weka: A machine learning workbench. In *Australian New Zealnd Intelligent Information Systems Conference* (1994), pp. 357–361.
9. LIAO, H.-J., LIN, C.-H. R., LIN, Y.-C., AND TUNG, K.-Y. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications 36*, 1 (2013), 16–24.

10. MIHOUB, A., FREDJ, O. B., CHEIKHROUHOU, O., DERHAB, A., AND KRICHEN, M. Denial of service attack detection and mitigation for internet of things using looking-back-enabled machine learning techniques. *Computers & Electr. Eng. 98* (2022), 107716.
11. MORADKHANI, M., AMIRI, A., JAVAHERIAN, M., AND SAFARI, H. A hybrid algorithm for feature subset selection in high-dimensional datasets using fica and iwssr algorithm. *Applied Soft Computing 35* (2015), 123–135.
12. MOSHKI, M., KABIRI, P., AND MOHEBALHOJEH, A. Scalable Feature Selection in High-Dimensional Data Based on GRASP. *Applied Artificial Intelligence 29*, 3 (2015), 283–296.
13. NEKKAA, M., AND BOUGHACI, D. Hybrid harmony search combined with stochastic local search for feature selection. *Neural Processing Letters 44*, 1 (2016), 199–220.
14. PARREÑO, F., ALVAREZ-VALDÉS, R., OLIVEIRA, J. F., AND TAMARIT, J. M. A hybrid GRASP/VND algorithm for two-and three-dimensional bin packing. *Annals of Operations Research 179*, 1 (2010), 203–220.
15. QUINCOZES, S. *ERENO: An Extensible Tool for Generating Realistic IEC–61850 Intrusion Detection Datasets*. PhD thesis, Fluminense Federal University, 2022.
16. QUINCOZES, S. E., ALBUQUERQUE, C., PASSOS, D., AND MOSSÉ, D. A survey on intrusion detection and prevention systems in digital substations. *Computer Networks 184* (2021),
17. QUINCOZES, S. E., MOSSÉ, D., PASSOS, D., ALBUQUERQUE, C., OCHI, L. S., AND DOS SANTOS, V. F. On the performance of grasp-based feature selection for cps intrusion detection. *IEEE Transactions on Network and Service Management 19*, 1 (2021), 614–626.
18. QUINCOZES, S. E., PASSOS, D., ALBUQUERQUE, C., MOSSÉ, D., AND OCHI, L. S. An extended assessment of metaheuristics-based feature selection for intrusion detection in cps perception layer. *Annals of Telecommunications* (2022), 1–15.
19. QUINCOZES, S. E., PASSOS, D., ALBUQUERQUE, C., OCHI, L. S., AND MOSSÉ, D. Grasp-based feature selection for intrusion detection in cps perception layer. In *2020 4th Conference on cloud and internet of things (CIoT)* (2020), IEEE, pp. 41–48.
20. RESENDE, M. G., AND RIBEIRO, C. C. GRASP: Greedy randomized adaptive search procedures. In *Search methodologies*. Springer, 2014, pp. 287–312.
21. SOARES, A. A. Z., SOARES, L. F., MATTOS, D. P., PINHEIRO, P. H., QUINCOZES, S. E., FERREIRA, V. C., APOSTOLO, G. H., CARRARA, G. R., MORAES, I. M., ALBUQUERQUE, C., ET AL. Enabling emulation and evaluation of IEC 61850 networks with TITAN. *IEEE Access 9* (2021), 49788–49805.
22. SUBANYA, B., AND RAJALAXMI, R. Feature selection using artificial bee colony for cardiovascular disease classification. In *2014 International conference on electronics and communication systems (ICECS)* (2014), IEEE, pp. 1–6.
23. SUBRAMANIAN, A., DRUMMOND, L. M., BENTES, C., OCHI, L. S., AND FARIAS, R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research 37*, 11 (2010), 1899–1911.
24. THEIN, K. M. M. Apache kafka: Next generation distributed messaging system. *International Journal of Scientific Engineering and Technology Research 3*, 47 (2014), 9478–9483.
25. VANDER-PALLEN, M. A., ADDAI, P., ISTEEFANOS, S., AND MOHD, T. K. Survey on types of cyber attacks on operating system vulnerabilities since 2018 onwards. In *2022 IEEE World AI IoT Congress (AIIoT)* (2022), IEEE, pp. 01–07.