# Microsserviços para Geração de RCLs no GRASP-FS: Uma Abordagem Escalável para Seleção de Features na Detecção de Intrusões em Sistemas Ciber-Físicos

**Nícolas Naves R. Faria**[1], **Silvio E. Quincozes**[1,2], **Juliano F. Kazienko**[3], **Vagner E. Quincozes**[4], **Gabriel Oliveira**[1] **e Estêvão F. C. Silva**[1]

[1]Universidade Federal de Uberlândia (UFU), Monte Carmelo - MG, Brasil
[2]Universidade Federal do Pampa (UNIPAMPA), Alegrete - RS, Brasil
[3]Universidade Federal de Santa Maria (UFSM), Santa Maria - RS, Brasil
[4]Universidade Federal Fluminense (UFF), Niterói - RJ, Brasil

{nicolasnaves, estevaof10, gabrieloliveirasouza620}@ufu.br
silvioquincozes@unipampa.edu.br, kazienko@redes.ufsm.br,
vequincozes@id.uff.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 [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]. Com isso, a adoção de soluções de segurança, como aquelas que analisam as características (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. 2021b] é utilizada em *Intrusion Detection Systems* (IDSs) para resolver o problema de otimização combinatória da seleção de *features*. A seleção de *features* representativas 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 *features* 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 *features* que atendem determinado critério. As *features* que não atenderem o critério estabelecido são descartadas, evitando-se sobrecarga na etapa de busca local. Tal redução se dá através de métodos que estimam o potencial de cada *feature* 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 *features* dentre as candidatas para serem otimizadas na etapa de busca local. Dessa forma, o desempenho das soluções exploradas na etapa de busca local dependem diretamente da qualidade das *features* disponíveis na RCL construída na etapa de construção [Quincozes et al. 2021b][Quincozes et al. 2022b].

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. 2021b], [Rocha et al. 2015], [Carvalho et al. 2022] [Kanakarajan and 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, ganha-se 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

e escalabilidade na geração e avaliação de RCLs.

Com isso, a principal contribuição deste trabalho consiste em abordar os desafios de escalabilidade na etapa construtiva do GRASP-FS. Como prova de conceito, utiliza-se o *framework* Kafka e quatro algoritmos de seleção de *features* 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. Resultados obtidos a partir de um cenário de Sistema Ciber-Físico — particularmente de um ambiente de redes de subestações elétricas modelado no *framework Efficacious Reproducer Engine for Network Operations* (ERENO) [Quincozes et al. 2022a] — indicam que a estratégia de construção da RCL impacta na F1-Score média das soluções geradas entre 50,47% e 84,92% para a detecção de intrusões nessas redes, a depender do algoritmo empregado.

## 2. Fundamentação Teórica

Os métodos tradicionais de seleção de *feature* podem ser classificados em três categorias: filtro (*filter*), embrulho (*wrapper*) e embutido (*embedded*). As técnicas de filtro costumam selecionar *features* que mais maximizam uma medida, tal como o ganho de informação. Já os métodos *wrapper* utilizam classificadores para avaliar os subconjuntos de *features* gerados. Por fim, a técnica *embedded* atrela o processo de seleção de *feature* ao treinamento do modelo [Quincozes et al. 2021b] [Witten et al. 2011]. Já o GRASP-FS, conforme introduzido na Seção 1, consiste em uma meta-heurística que permite a introdução de uma abordagem híbrida, aproveitando a estratégia dos algoritmos de filtro para executar uma redução inicial no número de *features* e, em seguida, usa algoritmos de *wrapper* para otimizar a seleção final por meio de buscas locais. Portanto, diz-se que o GRASP-FS possui duas etapas: uma construtiva e uma de busca local[Quincozes et al. 2021b].

O algorítimo *Information Gain* (IG) baseia-se no cálculo da redução de entropia para medir a relevância de uma *feature* em relação à classe das amostras analisadas. A Equação 1 apresenta a fórmula do *Information Gain* representado na fórmula por $IG$, $F$ é a *feature* considerada, $C$ é a classe, $H(C)$ é a entropia da classe e $H(C|F)$ é a entropia condicional da classe dado o valor da *feature* $F$.

$$IG(F, C) = H(C) - H(C|F) \tag{1}$$

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 da *feature* em distinguir as classes [Quinlan 1986].

O algoritmo *Gain Ratio* (GR) baseia-se no IG, discutido anteriormente, e leva em conta a distribuição das classes para corrigir o viés em direção a atributos com muitos valores. O *Gain Ratio* é calculado pela fórmula denotada na Equação 2, onde $GR$ é o *Gain Ratio*, $F$ é a *feature* considerada, $C$ é a classe, $IG(F, C)$ é o *Information Gain* da *feature* $F$ em relação à classe $C$, e $H(F)$ é a entropia da *feature* $F$.

$$GR(F, C) = \frac{IG(F, C)}{H(F)} \tag{2}$$

Ao calcular o GR para diferentes *features*, 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 a *feature* pode assumir [Quinlan 1993].

O algorítimo *Symmetrical Uncertainty* (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 *features* que possuam alta informação mútua e baixa redundância. O SU é dado pela fórmula denotada na Equação 3, onde $SU$ é o *Symmetrical Uncertainty*, $F$ é a *feature* considerada, $C$ é a classe, $IG(F, C)$ é o Information Gain da *feature* $F$ em relação à classe $C$, $H(F)$ é a entropia da *feature* $F$, e $H(C)$ é a entropia da classe [Kira and Rendell 1992].

$$SU(F, C) = \frac{2 \times IG(F, C)}{H(F) + H(C)} \tag{3}$$

Ao calcular o SU para diferentes *features*, é possível comparar a relevância dessas em relação à classe. Valores mais altos de SU indicam *features* 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].

O algoritmo *ReliefF* avalia a importância das *features* ao estimar o quão bem elas distinguem entre as classes por meio da comparação dos valores do atributo entre instâncias próximas e distantes, usando uma função de distância. A fórmula exata para o cálculo do ReliefF depende do cálculo das distâncias entre instâncias próximas e distantes e da atualização dos pesos dos atributos. A Equação 4 apresenta a fórmula do ReliefF, que expressa 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 $i$ da instância mais próxima pertencente à mesma classe que a instância atual, enquanto $\texttt{nearMiss}_i$ representa o valor do atributo $i$ da instância mais próxima pertencente a uma classe diferente da instância atual.

$$W_i = W_i - (x_i - \texttt{nearHit}_i)^2 + (x_i - \texttt{nearMiss}_i)^2 \tag{4}$$

Na fórmula, subtrai-se o quadrado da diferença entre o valor da *feature* atual e o valor do "*nearest hit*", e adiciona-se o quadrado da diferença entre o valor da *feature* atual e o valor do "*nearest miss*". Essa atualização de peso visa reduzir o peso da *feature* se ela difere mais dos valores de *features* próximos da mesma classe e aumentar o peso se ela difere mais dos valores de *features* próximas de classes diferentes. Essa fórmula é utilizada em cada iteração do algoritmo ReliefF para atualizar os pesos de cada *feature*, 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 and Rendell 1992].

## 3. Trabalhos Relacionados

Nesta seção são apresentados trabalhos que aplicam soluções baseadas na metaheurística GRASP para a seleção *features*. A revisão da literatura realizada revelou que nenhum

dos trabalhos existentes implementa uma arquitetura de microsserviços para a geração escalável de RCLs. Os trabalhos relacionados são discutidos a seguir. A Tabela 1 traz uma sumarização dos mesmos.

| Ref. | GRASP | Construção de RCL | Escalabilidade | Distribuição |
|---|---|---|---|---|
| [Esseghir 2010] | √ | ReliefF, SU, FCBF | × | × |
| [Quincozes et al. 2020] | √ | IG | × | × |
| [Quincozes et al. 2022b] | √ | GR | × | × |
| [Quincozes et al. 2021b] | √ | IG, GR, OneR e IWSS | × | × |
| [Rocha et al. 2015] | √ | SFS, SFE e SFFS | × | × |
| [Carvalho et al. 2022] | √ | Aleatório | × | × |
| [Diez-Pastor et al. 2011] | √ | IG | × | × |
| [Diez-Pastor et al. 2014] | √ | IG | × | × |
| [Kanakarajan and Muniasamy 2016] | √ | IG, SU e CFS | × | × |
| **Este trabalho** | √ | GR, IG, SU, ReliefF | √ | √ |

**Tabela 1. Propostas de seleção de *features* baseadas na metaheurística GRASP.**

Em [Esseghir 2010] é proposta uma solução baseada na metaheurística GRASP para resolver o problema de seleção de *features* através de abordagem híbrida entre as técnicas de seleção *filter* e *wrapper*. Os resultados revelam que o uso do GRASP é promissor para o problema da seleção de features. No entanto, as técnicas consideradas neste trabalho estão limitadas ao ReliefF, SU e FCBF, sendo que todas são executadas como alternativas e não de forma agregada tal como é proposto neste trabalho.

O trabalho [Rocha et al. 2015] apresenta uma comparação das metaheurísticas GRASP e *Tabu Search* para a seleção de *features*. Os resultados revelam que o uso do GRASP é promissor, porém, enfrenta desafios de escalabilidade quando o tamanho do conjunto de *features* aumenta. Isso se deve principalmente à ausência de mecanismos que promovam a escalabilidade na geração de soluções de conjuntos de *features*. Outra limitação se dá pelo uso exclusivo de métodos *wrapper* (*i.e., Sequential Forward Selection*, *Sequential Backward Elimination* e *Sequential Forward Floating Selection*), que são tipicamente mais custosos computacionalmente.

No trabalho [Carvalho et al. 2022], é proposta uma arquitetura para detecção e prevenção de intrusão em tempo real em *switches* eBPF baseada na meta-heurística GRASP-FS. Enquanto que há escalabilidade na detecção de intrusões por conta da implementação de classificadores embarcada nos dispositivos de rede (*switches* eBPF), a otimização dos modelos de aprendizado de máquina que incluem o processo de FS é realizado de maneira centralizada em um computador e não contempla escalabilidade.

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, onde emprega-se o algoritmo IG na construção de RCLs. Esse trabalho é estendido em [Diez-Pastor et al. 2014] através da proposta *Annealed Randomness Forest* (GAR-Forest), onde é feita uma parametrização da aleatoriedade envolvida no processo de construção de RCLs. No entanto, ambas as propostas de Diez não foram aplicadas no

contexto de detecção de intrusões e não contemplam princípios de escalabilidade.

Em [Kanakarajan and 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 *features* 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 *features*. O algoritmo IG foi usado originalmente na construção de RCL para o GRASP-FS. Posteriormente, em uma versão estendida [Quincozes et al. 2022b], tal algoritmo foi substituído pelo GR. Por fim, [Quincozes et al. 2021b] 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 *features* 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.

Com base no exposto, embora que existam diferentes propostas com a finalidade de seleção de *features* na literatura através de metaheurísticas como o GRASP, a arquitetura sequencial e falta de escalabilidade são desafios em aberto. Enquanto que os desafios da etapa de busca local do GRASP-FS são abordados em [Silva et al. 2023], neste trabalho aborda-se os desafios de escalabilidade na etapa construtiva.

## 4. Arquitetura Proposta

A construção de uma RCL representativa requer a execução de diferentes algoritmos e estratégias e, consequentemente, 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 a exploração de uma maior diversidade de 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]. Os componentes são descritos a seguir:

- 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.

Dado o que foi discutido anteriormente, a arquitetura proposta proporciona escalabilidade ao processo de seleção de *features*, 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 1 ilustra tal arquitetura, juntamente com sua entrada e saída.

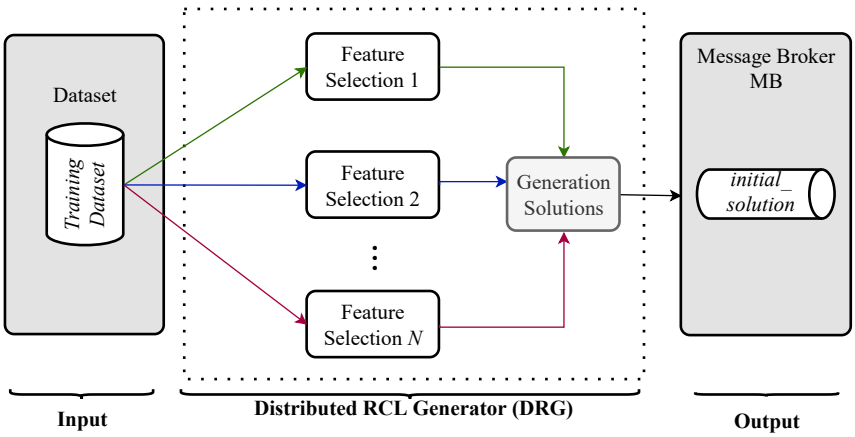



**Figura 1. A proposta de arquitetura orientada a microsserviços.**

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.,* `Feature Selection 1`, `Feature Selection 2`, ..., `Feature Selection N`) gera a sua respectiva RCL, de acordo com a estratégia e algoritmo de seleção de *features* implementado. Tal algoritmo considera todas as $M$ linhas para estimar a representatividade de cada *feature*. 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. 2021b].

Em seguida, as RCLs geradas por cada microsserviço de seleção de *features* são usadas como entrada para a função *Generation Solutions*, 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 *initial_solution*. 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].

## 5. Implementação e Experimentos

Nesta seção é apresentada como prova de conceito uma implementação que instância a arquitetura proposta. A Figura 2 ilustra os componentes implementados.

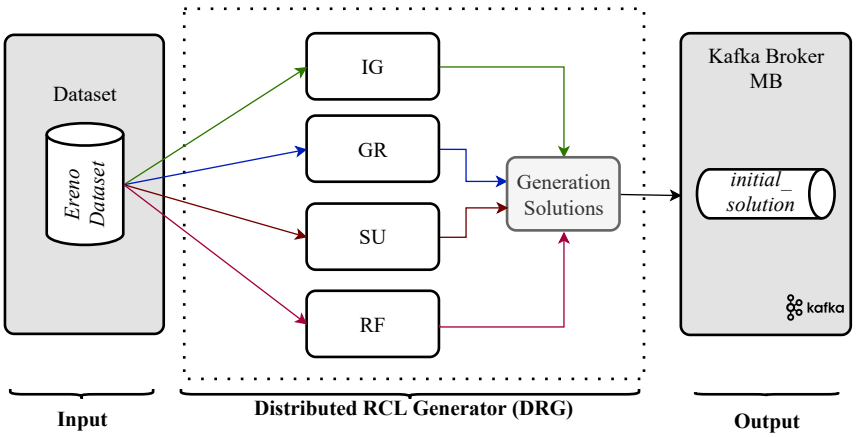



**Figura 2. Implementação dos componentes da arquitetura proposta.**

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

### 5.1. 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 abstrair e facilitar a configuração de servidores, bibliotecas, gerenciamento de dependências, métricas e *health checks*, entre outros. Ademais, foi empregado uso de contêineres Docker. A conteinerização permite a execução do aplicativo em um ambiente virtual empacotando todos os itens necessários, como arquivos, bibliotecas e outros componentes necessários. Umas das principais razões para o uso do Docker neste projeto consiste nas limitações do *framework* Spring Boot, que exige paralelismo na execução de múltiplas tarefas. O Spring Boot também pode aumentar desnecessariamente o tamanho do seu binário de implantação com dependências não utilizadas.

Os contêineres do Docker contêm apenas os dados de que seu aplicativo realmente precisa. Dessa forma, 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 de-

gradar 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 sejam executados diretamente nos contêineres do Docker, além deles o Apache Kafka também ficou sendo executado em contêiner para agilizar o desenvolvimento e testes, ajudando a criar microsserviços rapidamente [You and 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.Dessa maneira, o Apache Kafka age como um intermediário. O Kafka foi adotado na arquitetura proposta por permitir o trato com altos volumes de dados. Além disso, é projetado para aplicações de tempo-real encaminhando as mensagens para múltiplos consumidores eficientemente. O Kafka fornece integração desacoplada entre informações de produtores e consumidores, com envio de mensagens assíncronas. Além disso, os produtores não precisam ter conhecimento de quais são os dispositivos consumidores nem a sua localização [Garg 2013].

Assim, 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.

### 5.2. Cenário

De modo a construir o cenário de avaliação utilizado nos experimentos realizados, adotou-se a ferramenta ERENO [Quincozes et al. 2022a] 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 cenário modelado no ERENO é baseado na norma IEC-61850, que abrange a comunicação e a automação de subestações elétricas. Ele é dividido em dois níveis principais: baia (bay) e processo (process). O ERENO permite a geração de recursos representativos, tanto em termos de features de rede quanto de nível de aplicação, incluindo features cibernéticas e físicas, como dados de corrente e tensão. Em particular, foram modelados (i) IEDs de proteção, no nível de baia, os quais transmitem mensagens GOOSE para a notificação de eventos de faltas elétricas e (ii) Mergint Units (MUs), no nível de processo, que simulam a observação de uma linha de transmissão entre duas cidades brasileiras [Quincozes et al. 2022a].

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 Dispositivos Eletrônicos Inteligentes, do inglês, *Intelligent Electronic Devices* (IEDs). Cada amostra possui 70 *features* [Quincozes et al. 2021a][Quincozes et al. 2022a].

É importante destacar que tais *features* são usadas para treinar algoritmos de aprendizado de máquina tendo como propósito a detecção de diferentes tipos de ataques

modelados no *framewrok ERENO*, e.g., reprodução, injeção, mascaramento, envenenamento, inundação e adulteração de status de mensagens. [Quincozes et al. 2022a].

## 6. Resultados

De forma a avaliar a qualidade das soluções geradas pela etapa construtiva do GRASP-FS e publicadas no tópico *initial_solution*, neste trabalho utilizou-se do algoritmo classificador J48. Como métrica de comparação para as soluções geradas, adotou-se o cálculo da *F1-Score* (Equação 5). Tal métrica consiste na média harmônica das métricas *Precision* ($\frac{VP}{VP+FP}$) e *Recall* ($\frac{VP}{VP+FN}$), onde os seguintes valores são considerados: Falsos Positivos (FP), Falsos Negativos (FN) e Verdadeiros Positivos (VP).

$$F1Score = 2 \times \frac{Precison \times Recall}{Precision + Recall} \tag{5}$$

### 6.1. 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.

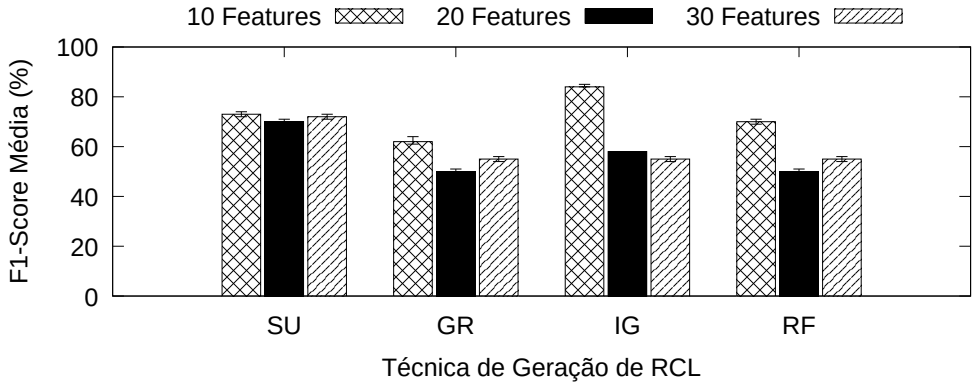



**Figura 3. F1-Score média alcançada por cada implementação de RCL.**

Na Figura 3 os resultados de cada algoritmo são detalhados de acordo com o número de *features* que compunha cada RCL. Nota-se que o uso do algoritmo IG com um uma RCL constituída de dez *features* foi capaz de gerar boas soluções com uma alta probabilidade: em média, para as 10.000 soluções geradas a partir de tal RCL, alcançou-se uma F1-Score de 84,92%. Por outro lado, é possível observar que o mesmo algoritmo apresenta desempenho inferior ao SU quando um número maior de *features* é usado para a construção da RCL. Em particular, o algoritmo SU não foi afetado significativamente em função do número de *features* selecionadas 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 *features* . Em contraste, as soluções geradas a partir do RCL construído pelo algoritmo IG

apresentam um desempenho decrescente quando um número maior de *features* 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 *features*, 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* compondo o RCL: os algoritmos GR e RF apresentam, respectivamente, F1-Scores médias de 62,98% e 70,61%. Já com vinte *features* ambos pioram significativamente, onde GR e RF apresentam, respectivamente, F1-Scores médias de 50,45% e 50,47%. Interessantemente, RCLs construídas com trinta *features* apresentaram um desempenho superior às RCLs construídas com vinte *features* e inferior às RCLs geradas com dez *features* pelos algoritmos GR e RF, que alcançaram, respectivamente, médias de 55,68% e 55,92%.

### 6.2. 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 4 são apresentadas as F1-Scores médias alcançadas pelos algoritmos implementados em cada microsserviço, agrupando-se pelo número de *features* configurado para compor o RCL de cada instância.

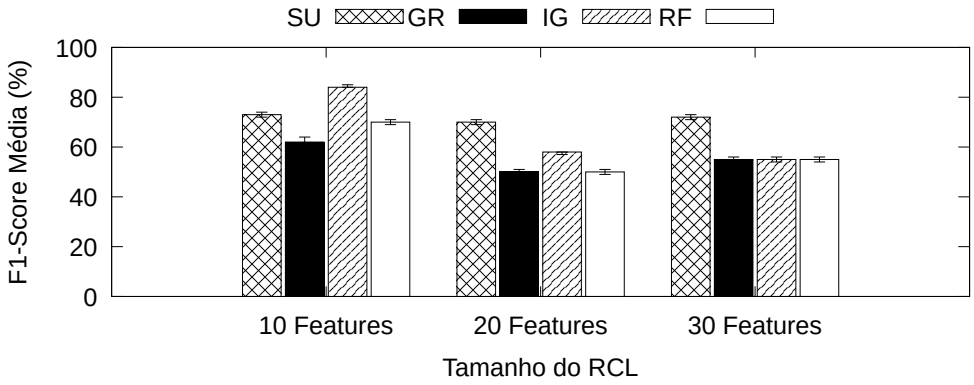



**Figura 4. F1-Score média alcançada por cada tamanho de RCL.**

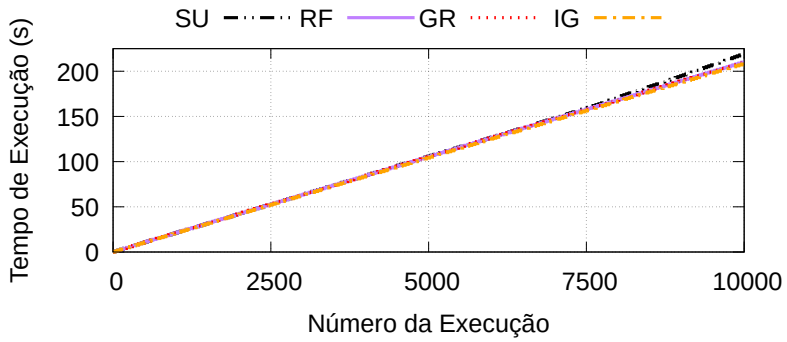
Por meio desta análise, é possível observar que RCLs de dez *features* apresentam de fato maior propensão de soluções geradas aleatoriamente apresentarem um bom nível de qualidade. Em particular, a média dos 40.000 experimentos que geraram soluções a partir de RCLs de 10 *features* pelos quatro algoritmos 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 *features* foi de apenas 57,47%. Similarmente, a média das 40.000 soluções geradas a partir de RCLs com trinta *features* foi 59,90%. Cabe destacar que o algoritmo SU obteve destaque na geração de RCLs com 30 *features*, o qual apresentou uma média de 72,57% enquanto os demais algoritmos apresentaram médias entre 55,44% e 55,92%.

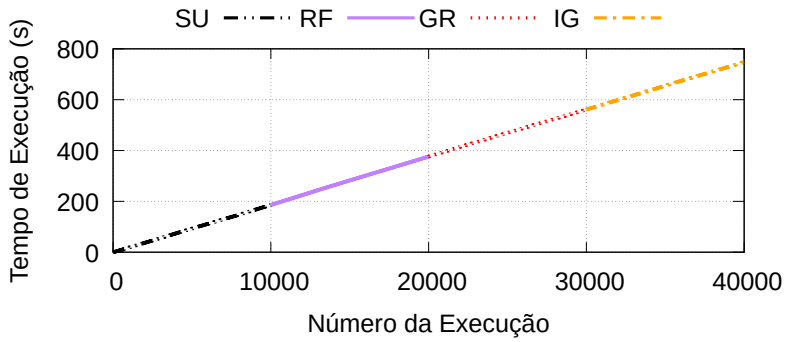
### 6.3. Avaliação de Escalabilidade

Conforme ilustrado na Figura 5a, o processamento paralelo dos algoritmos revelou-se cerca de 3,4 vezes mais rápido do que a execução sequencial ilustrada na Figura 5b. 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 5. Comparação entre execuções: Paralelo X Sequencial.**

Os resultados recém apresentados indicam diferença considerável no tempo de execução em favor da execução paralela. Ela permite que várias tarefas sejam executadas simultaneamente melhorando a escalabilidade e acelerando o tempo de execução dos microsserviços implementados. Além disso, o processamento paralelo pode ser escalonado de forma eficiente, aumentando o número de processadores para executar o mesmo conjunto de tarefas em um tempo ainda mais curto.

### 6.4. Discussão

Através dos estudos apresentados nesta seção, percebe-se que para o contexto estudado o algoritmo que apresenta a melhor capacidade de geração de soluções com alta qualidade é o IG. No entanto, tal afirmação somente é válida para o caso da adequada parametrização do tamanho do RCL. No cenário estudado, o uso de uma RCL composta pelas dez melhores *features* do IG apresentou uma F1-Score média de 84,92% para as 10.000 soluções aleatórias experimentadas. Já nos casos onde não se conhece o tamanho ideal para o RCL, a técnica mais indicada consiste no SU, que para todas as 30.000 soluções geradas apresenta uma média superior a 70,7%, independentemente do número de *features* (*i.e.,* 30, 20 ou 10) usados para compor o RCL.

De fato, a variação nos resultados entre diferentes configurações e algoritmos em cada instância de microsserviço demonstra a necessidade e importância de se ter diversas implementações. Somente dessa forma é possível avaliar em tempo real o desempenho

médio das mesmas e otimizar o processo de seleção de *features* através da metaheurística GRASP-FS [Quincozes et al. 2022b]. Por fim, o uso de microsserviços permitiu que a execução dos algoritmos de seleção de *features* se desse de modo paralelo. Tal abordagem permitiu a aceleração do processo em cerca de 3,4 vezes.

## 7. 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 *features* em aplicações de tempo real: à medida que o padrão de tráfego muda, *features* irrelevantes podem ganhar mais valor e as *features* representativas podem perder relevância para os modelos de *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 anteriormente, apresentamos a 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 *features*, 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 pelas dez melhores *features* 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.

## Referências


Ajeet Singh Raina and Johan Giraldo (2022). Kickstart Your Spring Boot Application Development. Disponível em: https://www.docker.com/blog/kickstart-your-spring-boot-application-development/. Acesso em: 18 de Novembro 2022.

Anderson, C. (2015). Docker [software engineering]. *IEEE Software*, 32(3):102–c3.

Carvalho, D., Quincozes, V. E., Quincozes, S. E., Kazienko, J. F., and dos Santos, C. R. P. (2022). 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*, pages 139–152. SBC.

Diez-Pastor, J.-F., Garcia-Osorio, C., and Rodriguez, J. J. (2014). Tree ensemble construction using a GRASP-based heuristic and annealed randomness. *Information Fusion*, 20:189–202.

Diez-Pastor, J. F., Garcıa-Osorio, C., Rodrıguez, J. J., and Bustillo, A. (2011). GRASP Forest: A New Ensemble Method for Trees. In *International Workshop on Multiple Classifier Systems*, pages 66–75.

Esseghir, M. A. (2010). Effective wrapper-filter hybridization through grasp schemata. In *Feature selection in data mining*, pages 45–54. PMLR.

FortiGuard Labs (2021). Brasil sofreu mais de 88,5 bilhões de tentativas de ataques cibernéticos em 2021. Disponível em: https://www.fortinet.com/br/corporate/about-us/newsroom/press-releases/2022/fortiguard-labs-relatorio-ciberataques-brasil-2021. Acesso em: 15 de Março de 2023.

Garg, N. (2013). *Apache Kafka*. Packt Publishing Birmingham, UK.

Kanakarajan, N. K. and Muniasamy, K. (2016). Improving the accuracy of intrusion detection using garforest with feature selection. In *Proceedings of the 4th International Conference on Frontiers in Intelligent Computing: Theory and Applications (FICTA) 2015*, pages 539–547. Springer.

Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. In *Machine learning proceedings 1992*, pages 249–256. Elsevier.

Kononenko, I. et al. (1994). Estimating attributes: Analysis and extensions of relief. In *ECML*, volume 94, pages 171–182. Citeseer.

Newman, S. (2021). *Building microservices*. "O'Reilly Media, Inc.".

Quincozes, S. E., Albuquerque, C., Passos, D., and Mossé, D. (2021a). A survey on intrusion detection and prevention systems in digital substations. *Computer Networks*, 184:107679.

Quincozes, S. E., Albuquerque, C., Passos, D., and Mossé, D. (2022a). ERENO: An Extensible Tool For Generating Realistic IEC-61850 Intrusion Detection Datasets. In *Concurso de Teses e Dissertações do XXII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 1–8.

Quincozes, S. E., Mossé, D., Passos, D., Albuquerque, C., Ochi, L. S., and dos Santos, V. F. (2021b). On the Performance of GRASP-Based Feature Selection for CPS Intrusion Detection. *IEEE Transactions on Network and Service Management*, 19(1):614–626.

Quincozes, S. E., Passos, D., Albuquerque, C., Mossé, D., and Ochi, L. S. (2022b). An extended assessment of metaheuristics-based feature selection for intrusion detection in CPS perception layer. *Annals of Telecommunications*, pages 1–15.

Quincozes, S. E., Passos, D., Albuquerque, C., Ochi, L. S., and Mossé, D. (2020). GRASP-based Feature Selection for Intrusion Detection in CPS Perception Layer. In *2020 4th Conference on Cloud and Internet of Things (CIoT)*, pages 41–48. IEEE.

Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1:81–106.

Quinlan, J. R. (1993). Program for machine learning. *C4. 5*.

Rocha, H. P., Ladeira, R. W. S., and Braga, A. P. (2015). Análise de Métodos Construtivos, Busca Local e Metaheurísticas para o Problema de Seleção de Características. In *Anais do XII Congresso Brasileiro de Inteligência Computacional*, pages 1–6.

Silva, E. F. C., Naves, N., Quincozes, S. E., Quincozes, V. E., Kazienko, J. F., and Cheikhrouhou, O. (2023). GDLS-FS: Scaling Feature Selection for Intrusion Detection with GRASP-FS and Distributed Local Search. In *37th Int. Conf. on Advanced Information Networking and Applications*, pages 199–210.

Witten, I. H., Frank, E., and Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, Amsterdam, 3 edition.

You, L. and Sun, H. (2022). Research and design of docker technology based authority management system. *Computational Intelligence and Neuroscience*, 2022.