Uma Ferramenta de Seleção de *Features* Baseada na Metaheurística GRASP com Fila de Prioridades para Sistemas de Detecção de Intrusão

Vagner E. Quincozes[1], Silvio E. Quincozes[2], Célio Albuquerque[1], Diego Passos[3], Daniel Mossé[4]

[1] Universidade Federal Fluminense - UFF

[2]Universidade Federal do Pampa - UNIPAMPA

[3]Instituto Superior de Engenharia de Lisboa - ISEL

[4]Universidade de Pittsburgh - PITT

vequincozes@midiacom.edu.br, silvioquincozes@unipampa.edu.br,

celio@ic.uff.br, diego.passos@isel.pt, mosse@pitt.edu

Resumo. *A crescente complexidade dos sistemas ciberfísicos exige mecanismos de segurança mais robustos. Nesse contexto, Sistemas de Detecção de Intrusão (IDSs) enfrentam o desafio de lidar com dados altamente dimensionais, o que compromete o desempenho e eleva o custo computacional. Este trabalho apresenta uma ferramenta para seleção de features em IDSs, baseada na metaheurística GRASP (Greedy Randomized Adaptive Search Procedure) com uso de fila de prioridades. A ferramenta é modular, automatizada e parametrizável, permitindo controlar aspectos como algoritmo de avaliação, número de iterações e tamanho da RCL (Restricted Candidate List). Os resultados indicam que a ferramenta reduz a dimensionalidade dos dados preservando, e em alguns casos ampliando, o desempenho preditivo dos modelos. Conclui-se que a proposta é eficaz e reprodutível para aplicações em cibersegurança.*

1. Introdução

A crescente complexidade dos sistemas ciberfísicos modernos, especialmente em infraestruturas críticas como redes elétricas inteligentes (*i.e.*, *Smart Grids*), demanda mecanismos de segurança avançados e eficientes [Abdi et al. 2024]. Nesse cenário, os sistemas de detecção de intrusão, do inglês *Intrusion Detection Systems* (IDS), desempenham papel crucial na identificação de comportamentos anômalos e tentativas de ataque em tempo real [Ji et al. 2024]. Dentre os desafios enfrentados por IDSs, destaca-se a dificuldade em lidar com conjuntos de dados altamente dimensionais, compostos por dezenas ou centenas de *features* (atributos) monitorados em tempo real [Thakkar and Lohiya 2022]. Para lidar com essa complexidade, a seleção de *features* torna-se essencial para garantir a eficiência computacional e o desempenho dos modelos de detecção, especialmente aqueles baseados em aprendizado de máquina (do inglês, *Machine Learning*), amplamente utilizados em IDSs [Quincozes et al. 2021a].

Na seleção de *features*, destacam-se duas categorias principais de métodos: os baseados em filtro, que avaliam cada atributo de forma independente do modelo, oferecendo maior velocidade com menor precisão [Zouhri et al. 2024]; e os *wrappers*, que incorporam o desempenho de um algoritmo de aprendizado de máquina na avaliação, resultando

em maior precisão, porém com maior custo computacional [Maldonado et al. 2022]. Como alternativa a essas abordagens, anteriormente [Quincozes et al. 2021b] propuseram um método híbrido que combina os pontos fortes de ambas, utilizando a metaheurística *Greedy Randomized Adaptive Search Procedure* (GRASP), cuja estratégia envolve uma fase construtiva, responsável por gerar soluções iniciais (*i.e.*, subconjuntos de *features*), seguida de uma fase de busca local, que refina essas soluções. Apesar de sua eficácia, tal solução exige alto custo computacional, devido à ausência de controle adaptativo entre os subconjuntos de *features* gerados na fase construtiva e o refinamento realizado pela busca local. Para superar esse desafio, foi proposta, em um estudo anterior [Quincozes et al. 2024], uma extensão que implementa a metaheurística GRASP com uso de fila de prioridades para seleção de *features*, buscando reduzir o tempo de seleção de *features* sem perda de desempenho na detecção [Quincozes et al. 2024].

Neste trabalho, apresentamos a ferramenta GRASPQ-FS *Tool*, que corresponde ao detalhamento, empacotamento e disponibilização pública da abordagem previamente proposta em [Quincozes et al. 2024]. A ferramenta implementa a metaheurística GRASP com uso de fila de prioridades para seleção de *features*, e agora está organizada como um pacote executável em Python, com *interface* via linha de comando, geração automatizada de *logs* e gráficos, e suporte à reprodutibilidade. O objetivo principal desta ferramenta é facilitar sua adoção por outros pesquisadores e profissionais da área de segurança cibernética. Em síntese, os principais objetivos da ferramenta GRASPQ-FS *Tool* são: 1) Otimizar o desempenho de IDSs por meio da seleção de subconjuntos reduzidos de *features*, mantendo a eficácia na classificação; 2) Promover a reprodutibilidade científica por meio da disponibilização do código-fonte, *datasets* e artefatos complementares; e 3) Apoiar pesquisas e experimentos em seleção de *features* aplicadas à IDSs, especialmente em contextos que demandam alto desempenho e reprodutibilidade.

2. Arquitetura e Funcionamento

A Figura 1 apresenta a arquitetura do GRASPQ-FS *Tool*, composta por quatro etapas principais, organizadas de forma sequencial e interdependente: i) entrada do usuário; ii) pré-processamento dos dados; iii) execução do módulo de seleção de *features* baseado na metaheurística GRASP com o uso de uma fila de prioridades; e iv) geração das saídas.

A execução se inicia com a entrada do usuário, que consiste na chamada do *script* via terminal, utilizando parâmetros definidos por linha de comando. Esses parâmetros configuram o comportamento da ferramenta nas etapas subsequentes e incluem: o algoritmo de classificação a ser utilizado na avaliação dos subconjuntos de *features*, o tamanho da Lista Restrita de Candidatos (RCL, do inglês, *Restricted Candidate List*), o número de *features* por solução inicial, e o número de iterações nas fases construtiva e de busca local.

Na sequência, é realizada a etapa de leitura e padronização dos dados, em que os arquivos de entrada (em formato .csv) são carregados e submetidos a transformações como normalização, codificação *one-hot* para *features* categóricas e codificação ordinal para as classes-alvo. Após esse pré-processamento, aplica-se uma etapa de ranqueamento das *features* com base na métrica de *Mutual Information*, que mede a dependência entre cada *feature* e a variável de classe. As *features* com maior relevância compõem a RCL, cujo tamanho é determinado pelo parâmetro fornecido pelo usuário.

Com os dados preparados e a RCL definida, a ferramenta inicia a fase construtiva.

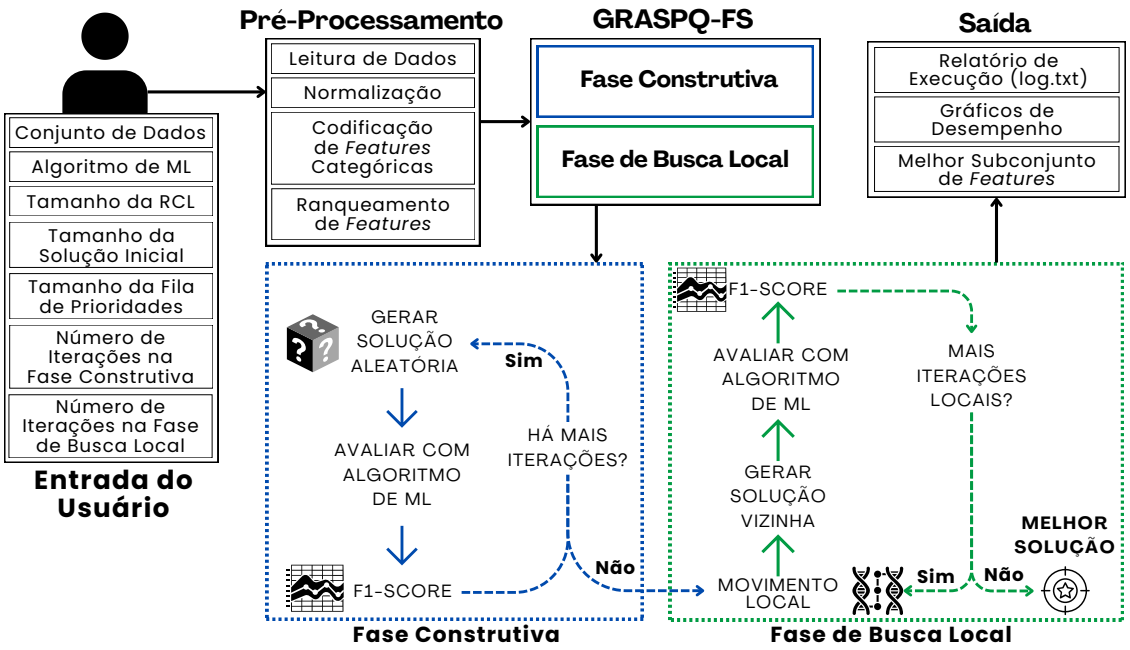

Figura 1. Visão geral da arquitetura da ferramenta GRASPQ-FS *Tool*.

Nessa etapa, a cada iteração, um subconjunto de *features* é selecionado aleatoriamente a partir da RCL para formar uma solução inicial, respeitando o número de *features* definido pelo usuário. Essa solução é então avaliada por um classificador supervisionado de *Machine Learning* (ML), também definido pelo usuário como parâmetro de execução, gerando um F1-Score associado. Soluções com bom desempenho são inseridas em uma fila de prioridades, que funciona como uma memória das melhores combinações testadas até o momento. O processo se repete ao longo das iterações, permitindo uma exploração diversificada do espaço de busca, mas guiada pela qualidade das soluções.

Concluída a fase construtiva, a ferramenta avança para a busca local, cujo objetivo é intensificar a exploração das melhores soluções encontradas. Essa etapa atua sobre os subconjuntos armazenados na fila de prioridades, ou seja, aqueles com maior potencial preditivo. Diferente da fase construtiva, que gera soluções inteiramente novas a partir de amostragens da RCL, a busca local realiza modificações pontuais em subconjuntos já existentes. Cada uma dessas soluções é ajustada por meio de pequenos movimentos locais, como a substituição de uma *feature* por outra ainda não selecionada. A cada iteração, o novo subconjunto é avaliado com um classificador de aprendizado de máquina, e seu F1-Score é comparado com o da solução anterior. Se houver melhora, a nova combinação é mantida; caso contrário, descarta-se a alteração. O processo se repete até o limite de iterações definido pelo usuário.

Por fim, são produzidas as saídas da ferramenta, que incluem: (i) o *log* da execução, (ii) gráficos que ilustram a evolução do desempenho e destacam as melhores soluções encontradas, e (iii) o relatório final com o subconjunto ótimo de *features* selecionadas, seus índices e nomes, e os tempos totais de execução de cada fase. Mais detalhes sobre essas saídas são apresentados na Seção 4.

3. Parâmetros de Execução, *Interface* e Requisitos

A ferramenta GRASPQ-FS *Tool* foi desenvolvida em Python e projetada para ser operada por linha de comando, permitindo ao usuário configurar facilmente os parâmetros de execução e ajustar o comportamento da metaheurística GRASP a diferentes objetivos experimentais. Essa abordagem garante portabilidade, simplicidade de uso e facilidade de integração em *pipelines* automatizados.

```
usage: main.py [-h] [-a {knn,dt,nb,svm,rf,xgboost,linear_svc,sgd}] [-rcl RCL_SIZE]
               [-is INITIAL_SOLUTION] [-pq PRIORITY_QUEUE] [-lc LOCAL_ITERATIONS]
               [-cc CONSTRUCTIVE_ITERATIONS]
```

(a) *Interface* de ajuda exibida com o parâmetro `-h`.

```
Execution parameters:
  Algorithm: dt
  RCL Size: 38
  Initial Solution Size: 5
  Priority Queue Size: 10
  Local Search Iterations: 100
  Constructive Iterations: 100
--------------------------------------------------
```

(b) Parâmetros definidos no início da execução.

Figura 2. *Interface* de linha de comando da ferramenta GRASPQ-FS *Tool*.

As Figuras 2a e 2b ilustram um exemplo de execução da ferramenta, apresentando a *interface* exibida ao usuário no terminal. A Figura 2a mostra a *interface* de ajuda, acessada com o argumento `python -h`, que descreve a estrutura dos parâmetros disponíveis. A Figura 2b exibe os parâmetros efetivamente utilizados durante a execução.

Tabela 1. Descrição dos parâmetros da ferramenta.

Parâmetro	Descrição	Valor Padrão
`-a`, `--algorithm`, `--alg`	Algoritmo de classificação usado na avaliação (ver Tabela 2a)	`nb`
`-rcl`, `--rcl_size`, `--rcl`	Tamanho da Lista Restrita de Candidatos (RCL)	`10`
`-is`, `--init_sol`, `--initial_solution`	Número de features em cada solução inicial	`5`
`-pq`, `--pq_size`, `--priority_queue`	Tamanho máximo da fila de prioridades	`10`
`-cc`, `--const`, `--constructive_iterations`	Número de iterações da fase construtiva	`100`
`-lc`, `--ls`, `--local_iterations`	Número de iterações da busca local	`50`

Os parâmetros controlam aspectos essenciais das fases construtiva e de busca local. É possível definir, por exemplo, o algoritmo de avaliação, o tamanho da RCL, a quantidade de *features* por solução inicial, a capacidade da fila de prioridades e o número de iterações de cada fase. A Tabela 1 resume os argumentos de execução disponíveis. Os valores-padrão foram definidos com base em experimentos preliminares descritos em [Quincozes et al. 2024], buscando equilíbrio entre desempenho e custo computacional. O classificador `GaussianNB` foi adotado como padrão por não realizar seleção interna de *features*, permitindo avaliar isoladamente o impacto da seleção de *features*. Todos os parâmetros são facilmente ajustáveis conforme o experimento.

Tabela 2. Algoritmos e bibliotecas suportadas pela GRASPQ-FS *Tool*.

Sigla	Algoritmo
`nb`	Gaussian Naive Bayes
`dt`	Decision Tree
`knn`	K-Nearest Neighbors
`rf`	Random Forest
`svm`	Support Vector Machine
`linear_svc`	Linear SVC
`sgd`	Stochastic Gradient Descent
`xgboost`	XGBoost

(a) Algoritmos suportados

Biblioteca	Versão mínima
`pandas`	≥ 1.3
`numpy`	≥ 1.21
`scikit-learn`	≥ 1.0
`matplotlib`	≥ 3.4
`seaborn`	≥ 0.11

(b) Bibliotecas necessárias

A execução da ferramenta requer Python versão 3.8 ou superior. As bibliotecas e os algoritmos de ML supervisionado suportados estão organizados na Tabela 2, que está dividida em duas partes: (a) os algoritmos disponíveis para avaliação das soluções geradas, e (b) as bibliotecas necessárias para execução do GRASPQ-FS *Tool*. Novos algoritmos podem ser adicionados à ferramenta por meio da inclusão de funções compatíveis com a *interface* do *pipeline* de avaliação. Para isso, basta implementar o classificador desejado utilizando uma biblioteca suportada (por exemplo, `scikit-learn`) e registrar seu identificador no módulo de configuração.

4. Saídas Geradas pela Ferramenta

Ao final da execução, a ferramenta gera automaticamente saídas textuais (Subseção 4.1) e gráficas (Subseção 4.2) que registram o desempenho obtido e a evolução das soluções, com foco em transparência, reprodutibilidade e apoio à análise dos resultados.

4.1. Saídas Textuais

A principal saída textual é o arquivo de *log*, que acompanha toda a execução e registra eventos relevantes como soluções testadas, F1-Scores obtidos e melhorias identificadas.

Durante a fase construtiva, a ferramenta gera soluções iniciais por meio de combinações aleatórias de *features*, uma para cada iteração. No exemplo apresentado, foram realizadas 100 iterações, resultando em 100 soluções distintas. Ao final dessa fase, as 10 com maior F1-Score foram armazenadas em uma fila de prioridades (com tamanho definido pelo usuário), enquanto as demais foram descartadas. Essa estrutura é utilizada como base para a intensificação subsequente. A Figura 3 exibe o conteúdo final da fila. A solução destacada — `[31, 21, 19, 22, 38]` — foi utilizada como exemplo para ilustrar o funcionamento da busca local, conforme mostrado na Figura 4.

A Figura 4 mostra como a solução inicial passa por modificações iterativas durante a busca local, por meio de pequenos movimentos no subconjunto de *features*. Nesta execução, para fins de demonstração, a busca local foi iniciada com a solução `[31, 21, 19, 22, 38]`, conforme indicado na primeira linha do *log*: "*starting local search with initial solution...*" com F1-Score de 80,29%. Cada nova solução gerada é avaliada por um classificador supervisionado, e seu F1-Score é comparado com o anterior. Se o desempenho melhora, a nova solução substitui a anterior como a melhor encontrada até o

```
Priority Queue:
F1-Score: 0.8495197111112741, Solution: [21, 31, 39, 30, 22]
F1-Score: 0.8371407697997221, Solution: [36, 22, 31, 30, 38]
F1-Score: 0.80740818699694, Solution: [21, 38, 31, 36, 22]
F1-Score: 0.8128303193674032, Solution: [19, 38, 31, 22, 30]
F1-Score: 0.8366737952772019, Solution: [39, 30, 36, 31, 38]
F1-Score: 0.8028878007095185, Solution: [31, 21, 19, 22, 38]
F1-Score: 0.8171862434196675, Solution: [30, 36, 38, 37, 39]
F1-Score: 0.8108138221373642, Solution: [22, 21, 38, 31, 0]
F1-Score: 0.8200157080577367, Solution: [30, 39, 36, 31, 37]
F1-Score: 0.8347355390731009, Solution: [21, 31, 38, 22, 39]
```

Figura 3. Exemplo da fila de prioridades gerada após a fase construtiva.

```
Starting Local Search with initial solution: [31, 21, 19, 22, 38], F1-Score: 0.8028878007095185
  → Local Iteration 1/10 | Current best F1: 0.8029
    ✓ Evaluated F1-Score: 0.7342 for solution: [0, 21, 30, 36, 37]
  → Local Iteration 2/10 | Current best F1: 0.8029
    ✓ Evaluated F1-Score: 0.7990 for solution: [0, 21, 30, 36, 39]
  → Local Iteration 3/10 | Current best F1: 0.8029
    ✓ Evaluated F1-Score: 0.5490 for solution: [19, 30, 31, 36, 37]
  → Local Iteration 4/10 | Current best F1: 0.8029
    ✓ Evaluated F1-Score: 0.5289 for solution: [0, 19, 21, 31, 36]
  → Local Iteration 5/10 | Current best F1: 0.8029
    ✓ Evaluated F1-Score: 0.8895 for solution: [22, 30, 31, 36, 39]
        Improvement found! New best solution: [22, 30, 31, 36, 39] with F1-Score: 0.8895
  → Local Iteration 6/10 | Current best F1: 0.8895
    ✓ Evaluated F1-Score: 0.3831 for solution: [0, 19, 21, 37, 39]
  → Local Iteration 7/10 | Current best F1: 0.8895
    ✓ Evaluated F1-Score: 0.7318 for solution: [0, 21, 30, 31, 37]
  → Local Iteration 8/10 | Current best F1: 0.8895
    ✓ Evaluated F1-Score: 0.4054 for solution: [0, 19, 21, 36, 39]
  → Local Iteration 9/10 | Current best F1: 0.8895
    ✓ Evaluated F1-Score: 0.4895 for solution: [0, 19, 31, 37, 39]
  → Local Iteration 10/10 | Current best F1: 0.8895
    ✓ Evaluated F1-Score: 0.4662 for solution: [0, 19, 21, 37, 38]
Local Search completed. Best F1-Score: 0.8894944653961446, Best Solution: [39, 31, 36, 30, 22]
```

Figura 4. *Log* da ferramenta: Exemplo de melhoria F1-Score de nova solução.

momento. A linha destacada na figura ilustra um desses momentos de melhoria. Esse processo se repete até atingir o número de iterações locais definido pelo usuário, permitindo acompanhar com precisão a evolução da intensificação.

Por fim, a ferramenta exibe um resumo final da execução. São informados a melhor solução encontrada — apresentada pelos índices e nomes extraídos do cabeçalho do *dataset* —, o melhor F1-Score obtido com a seleção e os tempos totais de execução das fases construtiva e de busca local. Também é apresentado o que chamamos de *baseline*, isto é, o F1-Score obtido com o uso de todas as *features* do *dataset*, permitindo uma comparação direta com os resultados da seleção. A Figura 5 ilustra um exemplo desse resumo gerado automaticamente ao final da execução.

```
Baseline F1-Score (All Features): 0.8590
Best F1-Score: 0.8894
Best Feature Set (indices): [39, 31, 36, 30, 22]
Best Feature Set (names): 'delay' (39), 'sqDiff' (31),
'timestampDiff' (36), 'stDiff' (30), 'StNum' (22)
Total execution time for Constructive Phase: 0.32469619 seconds
Total execution time for Local Search Phase: 0.78532939 seconds
```

Figura 5. Resumo final exibido no terminal ao término da execução.

4.2. Saídas Gráficas

Além disso, dois gráficos são gerados para ilustrar visualmente o desempenho da ferramenta, com base no F1-Score: (i) o Gráfico de Evolução das Soluções; e (ii) o Gráfico de Destaque das Melhores Soluções.

A Figura 6 apresenta um exemplo do gráfico (i), ilustrando a evolução do F1-Score ao longo das iterações da fase construtiva e da busca local. Cada barra representa uma solução avaliada, permitindo visualizar de forma clara sua qualidade individual e seu papel no processo. As barras em azul correspondem às soluções iniciais geradas na fase construtiva; em vermelho, aquelas que foram selecionadas para compor a fila de prioridades, por apresentarem desempenho superior; e, em verde, as que foram posteriormente aprimoradas pela fase de busca local, com melhorias incrementais no F1-Score. Esse gráfico facilita a interpretação dos ganhos obtidos por cada etapa e evidencia a contribuição da busca local na refinamento das soluções.

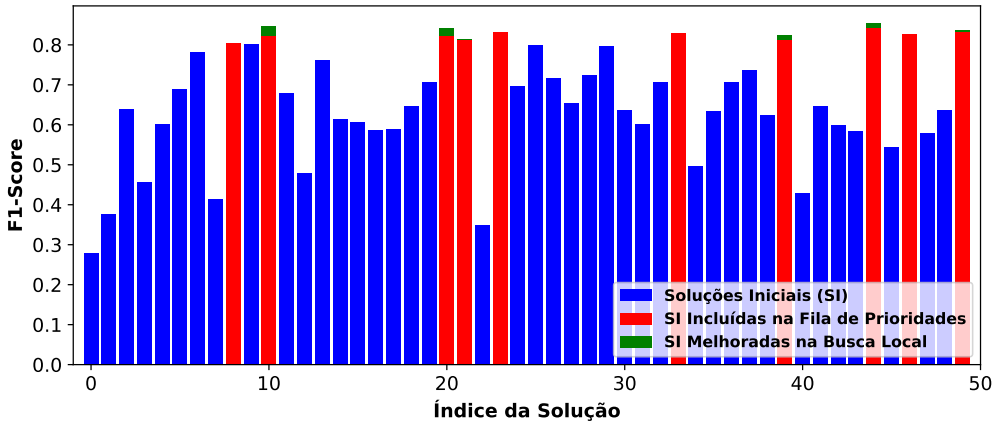

Figura 6. Gráfico de Evolução das Soluções.

A Figura 7 apresenta um exemplo do gráfico (ii), outra forma de visualização gerada automaticamente ao final da execução da ferramenta. Nesse gráfico de dispersão, cada ponto representa uma solução avaliada, posicionada conforme seu índice e F1-Score obtido. As soluções selecionadas para compor a fila de prioridades — as 10 melhores neste exemplo — são destacadas em vermelho, enquanto as demais aparecem em azul.

5. Demonstração e Disponibilidade

A ferramenta GRASPQ-FS *Tool* será demonstrada diretamente no terminal, sem necessidade de *interface* gráfica ou editor de código. A execução pode ser feita utilizando o interpretador Python (versão 3.8 ou superior) ou, alternativamente, via Docker. O script principal `main.py` aceita diferentes combinações de parâmetros definidos via `argparse`. A demonstração simulará o processo de seleção de *features* em conjuntos de dados para IDS, com geração automática de *logs* e gráficos. Para a visualização direta dos gráficos, recomenda-se a execução em ambiente com suporte a *interface* gráfica. O repositório está disponível em `https://github.com/vagnerereno/GRASPQ-FS`.

Execução da ferramenta: após clonar o repositório, recomenda-se a criação de um ambiente virtual com `python -m venv venv` e ativação com `source`

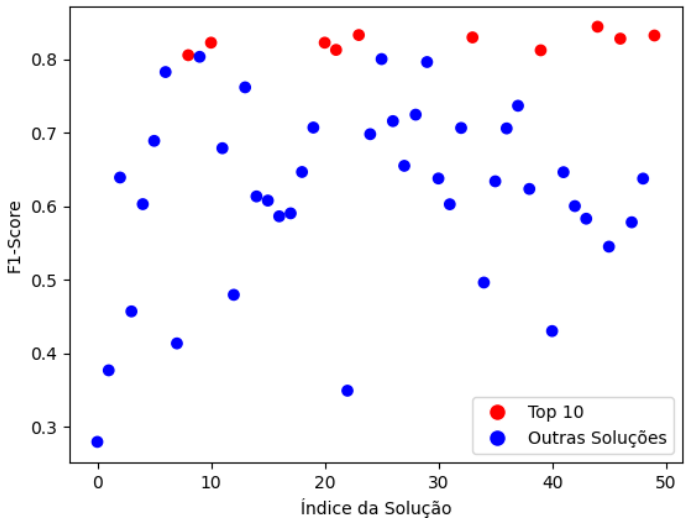

Figura 7. Gráfico de Destaque das Melhores Soluções.

`venv/bin/activate` (Linux/Mac) ou `venv\Scripts\activate` (Windows). Em seguida, as dependências devem ser instaladas com `pip install -r requirements.txt`. A execução é realizada por meio do comando `python main.py` seguido dos parâmetros desejados, como `-a nb -rcl 10 -is 5 -pq 10 -lc 50 -cc 50`, podendo variar conforme o perfil de teste. Alternativamente, a ferramenta pode ser executada via **Docker**, eliminando a necessidade de configuração do ambiente local: basta construir a imagem com `docker build -t main .` e executá-la com `docker run --rm main` seguido dos parâmetros desejados. Essa configuração também permite reproduzir o processo experimental que originou as saídas textuais e gráficas apresentadas neste trabalho.

Documentação: todas as instruções detalhadas estão disponíveis no arquivo `README.md`, incluído no repositório. O documento fornece explicações bilíngues (português e inglês), exemplos de uso, alternativas de parâmetros e requisitos mínimos. Também estão incluídos arquivos `.csv` de entrada. Além disso, há um **vídeo demonstrativo** disponível no README, com a instalação e execução da ferramenta. Alternativamente, o vídeo de demonstração pode ser acessado em `https://drive.google.com/file/d/1y3AHiyWszxBx_ExasQJ8SijLP_A3XX2t`.

6. Considerações Finais

Este trabalho apresentou a ferramenta GRASPQ-FS *Tool*, uma implementação para seleção de *features* em IDSs baseada na metaheurística GRASP com fila de prioridades. A proposta visa reduzir a dimensionalidade de conjuntos de dados sem comprometer o desempenho de classificação, com foco em aplicações críticas. A ferramenta oferece uma execução totalmente parametrizável por linha de comando, geração automática de *logs* e visualizações dos resultados, além de suporte à reprodutibilidade científica por meio da disponibilização do código-fonte e artefatos associados.

Como trabalhos futuros, pretende-se ampliar o suporte a diferentes formatos de dados e avaliar o uso de técnicas de *Explainable Artificial Intelligence* (XAI), como os valores de *SHapley Additive exPlanations* (SHAP), para compor a RCL no processo de ranqueamento de *features*.

Referências

Abdi, N., Albaseer, A., and Abdallah, M. (2024). The Role of Deep Learning in Advancing Proactive Cybersecurity Measures for Smart Grid Networks: A Survey. *IEEE Internet of Things Journal*, 11(9):16398–16421.

Ji, R., Padha, D., Singh, Y., and Sharma, S. (2024). Review of intrusion detection system in cyber-physical system based networks: Characteristics, industrial protocols, attacks, data sets and challenges. *Transactions on Emerging Telecommunications Technologies*, 35(9):e5029.

Maldonado, J., Riff, M. C., and Neveu, B. (2022). A review of recent approaches on wrapper feature selection for intrusion detection. *Expert Systems with Applications*, 198:116822.

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., 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, V. E., Quincozes, S. E., Albuquerque, C., Passos, D., and Mossé, D. (2024). Efficient Feature Selection for Intrusion Detection Systems with Priority Queue-Based GRASP. In *2024 IEEE 13th International Conference on Cloud Networking (CloudNet)*, pages 1–8.

Thakkar, A. and Lohiya, R. (2022). A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. *Artificial Intelligence Review*, 55(1):453–563.

Zouhri, H., Idri, A., and Ratnani, A. (2024). Evaluating the impact of filter-based feature selection in intrusion detection systems. *International Journal of Information Security*, 23(2):759–785.