UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA MECÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA NAVAL

OTIMIZAÇÃO DE HIPERPARÂMETRO DE UMA REDE NEURAL ARTIFICIAL POR GRASP REATIVO PARA PREVISÃO DE FALHA EM DUTOS CORROÍDOS

SORAIA MARIA DA SILVA

Recife
2024

SORAIA MARIA DA SILVA

OTIMIZAÇÃO DE HIPERPARÂMETRO DE UMA REDE NEURAL ARTIFICIAL POR GRASP REATIVO PARA PREVISÃO DE FALHA EM DUTOS CORROÍDOS

Trabalho apresentado a Universidade Federal de Pernambuco, Campus Recife, como requisito para obtenção do título de Bacharel em Engenharia Naval.

Orientador: Prof. Dr. Adriano Dayvson Marques Ferreira

Recife

2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Silva, Soraia Maria da.
Otimização de hiperparâmetro de uma rede neural artificial por GRASP reativo para previsão de falha em dutos corroídos / Soraia Maria da Silva. Recife, 2024.
49 p.

Orientador(a): Adriano Dayvson Marques Ferreira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Naval Bacharelado, 2024.

Inclui referências, apêndices.

1. Pressão de falha. 2. Dutos corroídos. 3. Rede Neural Artificial. 4. Meta- heurística. 5. GRASP. I. Ferreira, Adriano Dayvson Marques. (Orientação). II. Título.

620 CDD (22.ed.)

SORAIA MARIA DA SILVA

OTIMIZAÇÃO DE HIPERPARÂMETRO DE UMA REDE NEURAL ARTIFICIAL POR GRASP REATIVO PARA PREVISÃO DE FALHA EM DUTOS CORROÍDOS

Trabalho de conclusão de curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia Naval, pela Universidade Federal de Pernambuco.

Aprovado em: 28 de março de 2024.

Banca Examinadora

_____

Prof. Dr. Adriano Dayvson Marques Ferreira (Orientador)

Universidade Federal de Pernambuco

_____

Profa. Dra. Luciete Alves Bezerra (Examinador 1)

Universidade Federal de Pernambuco

_____

Me. Marco Antônio Figueirôa da Silva Cabral (Examinador 2)

Universidade Federal de Pernambuco

AGRADECIMENTOS

Agradeço primeiramente a Deus pela força para perseverar e depois a minha família, em especial a minha mãe e minhas irmãs por tudo.

Agradeço também aos amigos que fiz durante o curso e a Matheus, que sempre me deu apoio.

Ao meu orientador, Prof Dr. Adriano Dayvson Marques Ferreira, pela confiança em permitir com que eu continuasse o seu trabalho. E também a todos os outros professores que passaram pela minha vida acadêmica e me permitiram ser o que sou hoje.

“Hold on

Let time be patient

You are still strong

Let pain be gracious

Love will soon come

Just hold, hold on”

Adele

RESUMO

A análise de integridade estrutural de dutos corroídos torna-se crucial no cenário atual, impulsionada pelo aumento do uso de dutos para o transporte de óleo e gás. A corrosão emerge como um mecanismo primário de falha em dutos, demandando avaliações de probabilidade de falha. Embora métodos experimentais e modelos semiempíricos sejam comuns para essa finalidade, apresentam limitações significativas em termos de custo e conservadorismo. No estudo de Ferreira (2022), foi proposto o uso de Redes Neurais Artificiais (RNA) para prever a pressão de falha em dutos corroídos com defeitos axissimétricos, proporcionando resultados mais rápidos que os métodos de elementos finitos. Entretanto, a otimização dos hiperparâmetros da RNA foi realizada por meio de tentativa e erro. Brandão (BRANDÃO, 2023) avançou na pesquisa ao otimizar os hiperparâmetros da RNA proposta por Ferreira (FERREIRA, 2022), através da meta-heurística *Particle Swarm Optimization* (PSO), comparando os resultados obtidos. Neste trabalho, uma RNA foi otimizada para prever falhas em dutos corroídos com defeitos axissimétricos, utilizando a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), e os resultados foram comparados com os estudos de Ferreira (FERREIRA, 2022) e Brandão (BRANDÃO, 2023). Todas as análises foram implementadas em *Python* (PYTHON, 2024). Os resultados obtidos superaram em velocidade e precisão os dados comparativos dos estudos. Apesar de apresentar uma estrutura mais densa do que a proposta por Ferreira (2022), a abordagem GRASP demonstrou-se mais eficaz.

Palavras-chave: pressão de falha; dutos corroídos; rede neural artificial; meta-heurística; GRASP.

ABSTRACT

The analysis of structural integrity in corroded pipelines has become crucial in the current scenario, driven by the increased use of pipelines for the transportation of oil and gas. Corrosion emerges as a primary failure mechanism in pipelines, necessitating assessments of failure probability. Although experimental methods and semi-empirical models are common for this purpose, they present significant limitations in terms of cost and conservatism. In Ferreira's study Ferreira (2022), the use of Artificial Neural Networks (ANN) was proposed to predict failure pressure in corroded pipelines with axisymmetric defects, providing faster results than finite element methods. However, the optimization of ANN parameters was carried out through trial and error. Brandão (2023) advanced the research by optimizing the parameters of the ANN proposed by Ferreira (2023) using the Particle Swarm Optimization (PSO) metaheuristic, comparing the obtained results. In this work, an ANN was optimized to predict failures in corroded pipelines with axisymmetric defects, employing the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, and the results were compared with the studies of Ferreira (2022) and Brandão (2023). All analyses were implemented in Python (2024). The results surpassed the comparative data of the studies in terms of speed and precision. Despite having a denser structure than the one proposed by Ferreira (2022) the GRASP approach proved to be more effective.

Keywords: failure pressure; corroded pipelines; artificial neural network; metaheuristic; GRASP.

LISTAS DE FIGURAS

LISTAS DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O aumento da utilização de dutos como modal de transporte para óleo e gás se deve ao meio de transporte ser constante e o preço de transporte apresenta um bom custo-benefício em comparação a caminhões e trens. Entretanto, os dutos podem vir a falhar, ocasionando consequências financeiras e ambientais. Por exemplo, vale citar o rompimento do oleoduto da Refinaria Duque de Caxias (Reduc), no ano 2000, onde foi liberando 1300 m^3 de produto e, o vazamento se espalhou por toda Baía de Guanabara e praias próximas afetando a economia, fauna e flora da região (CETESB, 2022). A pesca foi proibida por 30 dias e a Petrobras (empresa responsável pela Reduc) recebeu uma multa de R$51,05 milhões de reais (NITAHARA, 2016). A corrosão é um dos principais mecanismos de falha em dutos (PIMENTEL et al., 2020), sendo então necessário fazer análises dela. Para isso, as indústrias que utilizam dutos, avaliam se a pressão de falha devido à corrosão é inferior a pressão operacional máxima (+ - MOP) do duto. A Figura 1 apresenta o estado avançado de corrosão em um duto.

Figura 1 – Corrosão final em um duto.

Fonte: (RK CONSULTING ENGINEERS, 2020).

Há diversas formas de prever a pressão de falha em dutos devido à corrosão as mais comuns são os métodos experimentais e modelos semiempíricos (e.g., ASME B31G, RSTRENG e DNV RPF101 simples e complexo) (XIE; TIAN, 2018). O método experimental tem custo elevado e os modelos empíricos são conservadores em relação à previsão de falha

(i.e., no sentido de não subestimar a chance de falhas, de forma que o duto seja removido precipitadamente).

Para que os dutos não sejam removidos de forma precipitada, o Método dos Elementos Finitos (MEF) é utilizado. Ele obtém resultados próximos aos experimentais (com porcentagens inferiores a 9% do valor experimental (PIMENTEL et al., 2020)). Ainda assim, o MEF é um processo demorado (horas ou dias, dependendo do refinamento da malha) e sujeito a erros, por ser um processo repetitivo. Cabral et al (2022) fez a avaliação de em dutos com defeito axissimétrico (i.e., apresenta simetria em relação a um eixo) usando MEF com dados de RBP (modelo construindo a partir de espessura mínimas de dutos) e obteve resultados com desvio de 3% em relação ao experimental, mas a análise de Elementos Finitos (EF) é muito demorada.

Para resolver o problema da demora do MEF, o trabalho de Ferreira (2021) propôs a utilização de Rede Neural Artificial (RNA) para avaliar a pressão de falha em dutos obtendo resultados com erros menores e mais rápidos (resultado instantâneo, dependendo da variação da magnitude dos dados obtidos a rede vai demorar para treinar, mas o seu resultado é instantâneo).

Todavia, no trabalho de Ferreira et al (2021) não foi otimizado a seleção de alguns hiperparâmetros (e.g., o número de épocas; número de neurônios; número de camadas ocultas), e pode ser feito utilizando a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), como foi sugerido por Angel-Bello et al (2006).

Este Trabalho de Conclusão de Curso (TCC) otimizará hiperparâmetros de uma RNA para prever a pressão de falha em dutos com defeitos de corrosão axissimétricos. Cuja motivação é diminuir o tempo de seleção dos hiperparâmetros da RNA, garantindo uma precisão na previsão de falha, permitindo que ocorra mais rapidamente a tomada de decisão perante a retirada ou não de um trecho de duto, evitando trágicos acidentes. Este TCC é estruturado da seguinte maneira: Primeiramente, no capítulo 1 tem-se a introdução, que contém o objetivo; no capítulo 2, a revisão bibliográfica, com a justificativa e relevância; o capítulo 3 a fundamentação teórica, com a revisão da pesquisa para este trabalho; o capítulo 4, a metodologia utilizada ; o capítulo 5 contém os resultados; no capítulo 6, a conclusão; após, as REFERÊNCIAS; e por fim, o Anexo A, com o código utilizado para obter resultados.

1.1 Objetivos

O objetivo geral deste TCC é verificar a viabilidade de se utilizar a RNA com a otimização de hiperparâmetros feita pela meta-heurística GRASP, para fazer análises de pressão de falha

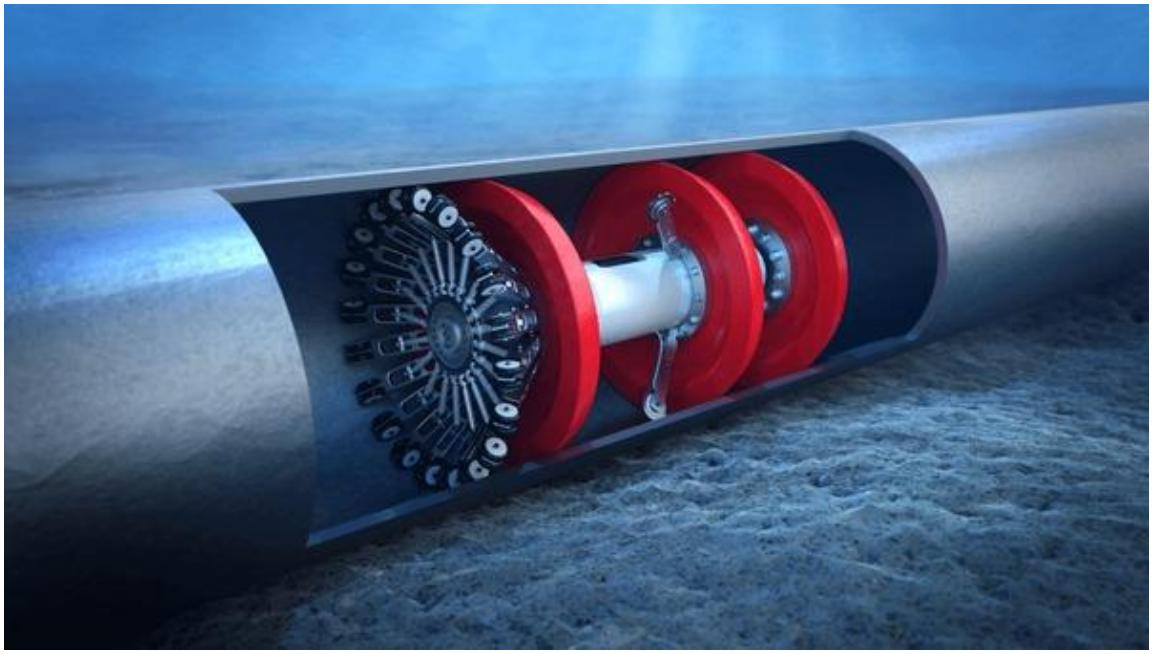
em dutos corroídos. Para atingir este resultado, os seguintes objetivos foram seguidos os seguintes objetivos:

1. Coletar e pré-processar dados para treino e validação da RNA profunda (FERREIRA, 2022).
2. Implementar uma rotina computacional que integre o algoritmo da RNA (FERREIRA, 2022), com o algoritmo do GRASP, i.e., RNA+ GRASP.
3. Proceder a uma análise minuciosa acerca da aplicabilidade do modelo proposto, focalizando especialmente na qualidade das soluções apresentadas e na eficiência temporal para a obtenção dos resultados almejados.
4. Realizar uma comparação entre os resultados obtidos por meio deste modelo e os resultados descritos por Ferreira (2022), assim como os resultados alcançados por Brandão (2023).

2 REVISÃO BIBLIOGRÁFICA

Os pesquisadores Xie e Titan (XIE; TIAN, 2018) propuseram um programa para monitorar a análise de integridade de dutos usando *In Line Inspection* (ILI) (i.e., consiste em um dispositivo chamado *pig* (Figura 2), que é inserido na tubulação e, faz a coleta de informações como comprimento de corrosão e espessura remanescente da parede do duto) que segue três passos: (i) detectar e identificar defeitos; (ii) prever o aumento do defeito (usando métodos estatísticos); e (iii) gerenciar falhas (i.e., política de reparos e atividades que limitem os danos). Importante destacar neste estudo o uso do ILI, que é um método muito utilizado por outros autores e também o uso dos métodos estatísticos para modelagem da corrosão.

Figura 2 - *Pig* de *In Line Inspection*.



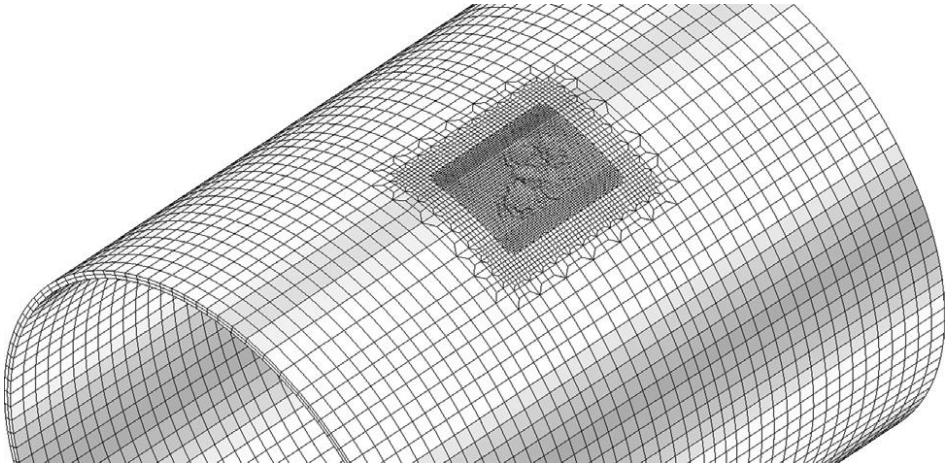
Fonte: (VISMAR UK, 2023)

Na mesma linha de Xie e Tian (XIE; TIAN, 2018) o estudo de Amaya-Gómez et al (AMAYA-GÓMEZ et al., 2019) modelaram a corrosão por um processo estatístico chamado de Processo Lévy (i.e., processo probabilístico que representa o movimento de um ponto cujos deslocamentos sucessivos não são idênticos em mesma variação de tempo (ZOLOTAREV, 1986)), que foi baseado na ILI e usou a avaliação de tempo médio de falha (i.e., tempo total necessário para o reparo dividido do número total de reparos em um período de tempo definido

(MENDES, 2019)) para a tomada de decisões (e.g., realização de manutenção; substituição de dutos; não interferências).

Foi estudado por Sun e Cheng (SUN; CHENG, 2018) a interação de múltiplos defeitos de corrosão e cálculo da pressão de falha da tubulação através de análise por EF (i.e., que consiste em discretizar o contínuo em um número finitos de variáveis). A grande vantagem do MEF (Figura 3) se deve ao fato de ser menos conservador nos valores de pressão de falha do que os métodos analíticos (e.g., ASME B31G, DNV RP-F101, RSTRENG Área Efetiva, Tipos Misto de Interação — TMI) e mais acessível que métodos experimentais.

Figura 3 - Simulação de corrosão em um modelo de elementos finitos.



Fonte: (PIMENTEL et al., 2020)

A modelagem manual do MEF requer muito tempo de trabalho dos engenheiros (chegando a dias) e acaba levando a erros por repetição desse processo e como alternativa o grupo PADMEC em parceria com a Petrobras construíram a ferramenta computacional PIPEFLAW, a qual usa os programas pagos Patran (MSC.PATRAN, 2022) e Ansys (ANSYS, 2022) para gerar modelos de corrosão e fazer análise de Elementos Finitos (EFs) de forma automática através dos dados obtidos por ILI.

Bruère et al (BRUÈRE et al., 2019) usaram o PIPEFLAW para fazer a análise da previsão de falha de dutos corroídos combinados de pressão interna e força de compressão longitudinal, considerando mudança de temperatura e verificou que a pressão de falha diminui com o aumento da força longitudinal e obteve a maior diferença em comparação com o experimental sendo menor que 13%. Pimentel e outros (PIMENTEL et al., 2020) também utilizaram a automatização pelos programas pagos *Patran* (MSC.PATRAN, 2022) e *Ansys* (ANSYS, 2022),

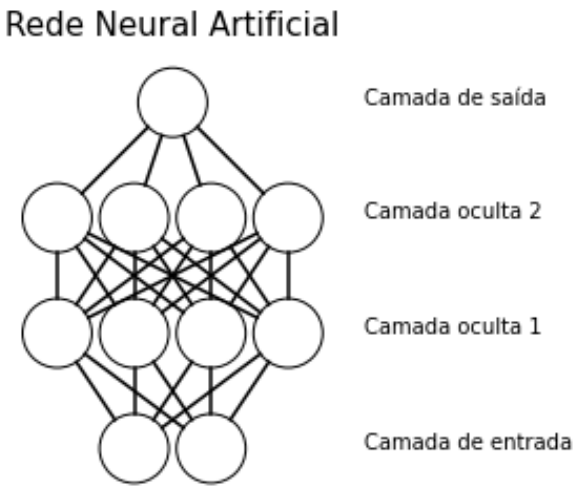
que cria a malha de EF dos defeitos de formatos reais e os dados de entrada também são obtidos da ILI, a maior diferença entre o valor experimental e o do MEF, foi menor a 16%.

O MEF obtém resultados muito próximos dos valores experimentais, mas mesmo automatizado demora um tempo considerável (horas ou dias, dependendo do refinamento da malha), principalmente quando a corrosão apresenta geometria complexa. Para resolver o problema da demora para analisar a pressão de falha, (FERREIRA et al., 2021) reduz os dados obtidos por ILI utilizando a Transformada *Wavelet* Discreta (TWD) (i.e., um filtro que mantém as espessuras com dimensões parecidas e reduz a quantidade de dados) e treina uma Rede Neural Artificial (RNA) para prever as pressões de falha em segundos, após treinada a RNA.

A RNA é um conjunto de neurônios que recebem uma entrada na primeira camada (camada de entrada), processa a entrada na (s) camada (s) intermediária (s) (camadas ocultas) e dá a solução na última camada (camada de saída), quando uma RNA tem muitas camadas na camada oculta, ela é chamada de RNA profunda. A Figura 4 representa a arquitetura de uma RNA profunda.

Para treinar a RNA, Ferreira et al (FERREIRA et al., 2021) utilizou dados estatísticos dos perfis de corrosão real para criar modelos de elementos finitos com corrosões aleatórias e processou eles no programa Code Aster (CODE_ASTER, 2021), que é um programa de código aberto. Assim, ele obteve uma diferença menor que 5%, se comparado com o MEF.

Figura 4 -Arquitetura de uma rede neural.



Fonte: Autora, 2024.

Brandão (BRANDÃO, 2023) continua o trabalho de Ferreira et al (FERREIRA et al., 2021) e otimiza hiperparâmetros de uma RNA profunda com o *Particle Swarm Optimization* (PSO). O PSO é um algoritmo que tenta imitar o movimento de animais e permite com que se ache soluções ótimas através de um espaço de busca (BRANDÃO, 2023). Os hiperparâmetros para serem otimizados por Brandão (BRANDÃO, 2023) foram: (i) número de neurônios; (ii) função de ativação e; (iii) número de camadas ocultas, visto que Ferreira et al (FERREIRA et al., 2021) fez por tentativa e erro a escolha dessas variáveis.

E em Angel-Bello et al (2006), eles propõem otimização de hiperparâmetros de uma RNA utilizando a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), que é uma meta-heurística de algoritmo de busca de vizinhança, o que difere da proposta de Brandão (BRANDÃO, 2023), que é um algoritmo evolucionário.

2.1 Justificativa e relevância

O estudo de dutos corroídos é importante no cenário atual devido ao crescimento do modal dutoviário e corrosão é uma das principais formas de acidente em dutos. Sendo necessários resultados rápidos e precisos em relação à pressão de falha, visto que as falhas comprometem a integridade estrutural.

Com a agilidade nos resultados da previsão da pressão de falha através do uso de RNA, pode-se aplicar o método para dar suporte em modelos de gêmeos digitais, que devido ao grande volume de dados produzidos por sensores demandam uma avaliação rápida dos dutos, além disso os engenheiros poderão usar seu tempo para outras finalidades (e.g., manutenção da linha de dutos; formas de limitar os danos causados pela corrosão).

A avaliação da pressão de falha é um tema extremamente importante nas indústrias de óleo e gás, mas seus dados devem ser precisos, rápidos e não conservadores (i.e., no sentido de não subestimar a chance de falhas, de forma que o duto seja removido prematuramente). Percebe-se então que usar uma RNA é uma proposta inovadora, assim como otimizar os hiperparâmetros da RNA, com objetivo de diminuir o tempo de resposta e recursos computacionais, além de ajustar a RNA para predizer de maneira mais precisa a pressão de falha em novos casos. Portanto, este trabalho contribuirá na área de corrosão de dutos fazendo a otimização de hiperparâmetros de uma RNA, com intuito de acelerar o tempo de resposta e a garantir a qualidade da previsão da pressão de falha.

3 FUNDAMENTAÇÃO TEÓRICA

O prognóstico de falhas em dutos corroídos, especialmente aqueles afetados por defeitos axissimétricos, é um desafio que demanda abordagens específicas. No âmbito do método GRASP, o processo de otimização revela-se fundamental para antecipar com precisão tais falhas. Diferentemente de técnicas convencionais, o GRASP emprega estratégias adaptativas e aleatórias durante a busca pela solução ótima, constituindo um componente vital na ferramenta computacional utilizada.

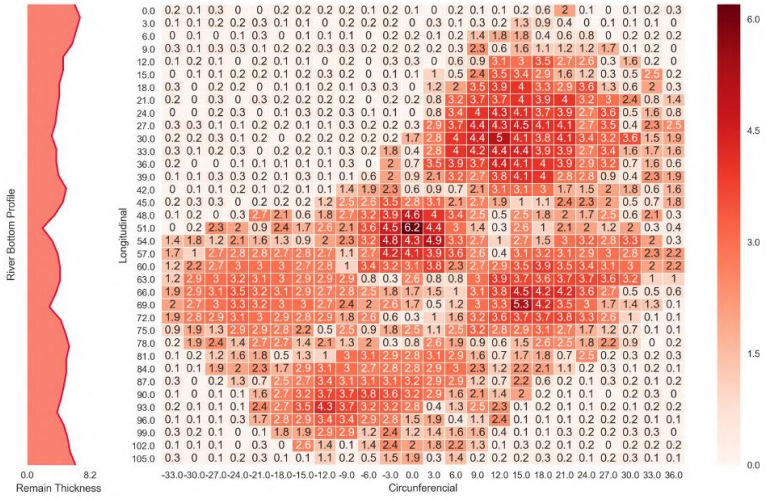
Neste contexto, esta seção destina-se a elucidar as teorias e conceitos relacionados aos métodos que serão utilizados para construção deste trabalho. As estratégias adaptativas e aleatórias implementadas no GRASP desempenham um papel crucial nas análises e movimentações realizadas para identificar com eficácia potenciais pontos de falha.

3.1 River Bottom Profile

Na área de dutos, a utilização de modelos axissimétricos com perfil *River Bottom Profile* (RBP) é uma prática difundida. O RBP é construído a partir das espessuras remanescentes, ou seja, as mínimas, obtidas por meio do *pig* de inspeção ILI. Este processo transforma a superfície tridimensional (3D) em uma representação bidimensional (2D), realizando a seleção das espessuras mínimas em cada linha circunferencial da grade total, conforme detalhado por Ferreira et al (2021).

A predominância de modelos de corrosão com perfil axissimétrico é notável, caracterizando-se pela simetria em relação a um eixo específico. Esta abordagem simétrica facilita análises e avaliações, contribuindo para uma compreensão mais eficaz dos padrões de corrosão em dutos. A consideração frequente desse perfil axissimétrico é um reflexo da relevância e praticidade que tal simetria proporciona na modelagem e predição de falhas em dutos, especialmente quando se busca compreender os efeitos corrosivos ao longo do tempo. A Figura 5, apresentada a seguir, proporciona uma visualização ilustrativa de um RBP.

Figura 5 - Representação de um r*iver bottom profile*.



Fonte: (FERREIRA et al., 2021).

3.2 Rede Neural Artificial

As RNAs se baseiam no cérebro humano, que consistem em densas redes de neurônios que se conectam entre si, realizando sinapses (HAYKIN, 2000). Uma RNA é dividida em camadas (neurônios).

Uma RNA mais simples se chama *perceptron*, nela um único neurônio recebe várias entradas (x_1, x_2, x_3, ... x_n) e produz uma saída única e que são ponderadas por hiperparâmetros de entrada, exibido em (1).

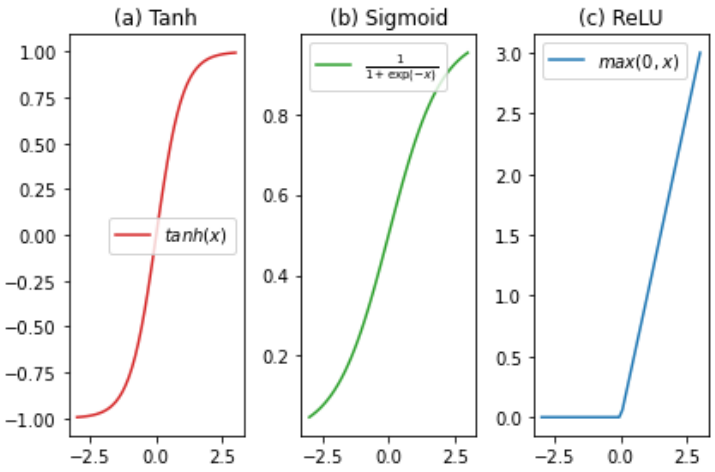
$$\sum w_{kj} * x_j + b_k \tag{1}$$

Onde "b_k" é chamado de viés, que é o valor mínimo para ativar um neurônio e w_{kj} são os pesos (valores dados a "relevância" de cada entrada) (HAYKIN, 2000).

Uma RNA profunda (Figura 4) tem mais neurônios na camada de entrada, diferente do *perceptron* e várias camadas ocultas, mas pode produzir várias saídas. As principais características (*features*) de uma rede neural são definidas pela camada de entrada. As camadas ocultas processam os dados da camada de entrada e a camada de saída apresenta a solução (BRANDÃO, 2023).

A ativação dos neurônios é feita através da função de ativação que decide se o neurônio vai ser ativado ou não. Camadas ocultas costumam ter a mesma função de ativação (e.g., função logística; função linear), mas a camada de saída costuma ser diferente (e.g., *Adam*; *Softmax*; *Sigmoid*) (HAYKIN, 2000). A Figura 6 mostra algumas funções de ativação, como a Tanh, Sigmoid e ReLU.

Figura 6 - Funções de ativação: (a) Tanh, (b) Sigmoid e (C) ReLU.



Fonte: Autora, 2024.

A retropropagação é um método fundamental no treinamento de redes neurais artificiais (RNAs), utilizado para ajustar os pesos e os vieses de forma a minimizar o erro da rede. O processo pode ser dividido em duas etapas principais (BRANDÃO, 2023).

Na primeira etapa, os dados de entrada são alimentados na RNA, onde são processados através das conexões ponderadas pelos pesos fixos. Essa etapa é conhecida como propagação direta, durante a qual as entradas são processadas em camadas sucessivas da rede até que a saída final seja gerada.

Na segunda etapa, a retropropagação propriamente dita, os pesos são ajustados com o objetivo de minimizar o erro entre a saída predita pela rede e o valor desejado. Para isso, é utilizado um algoritmo de otimização que se baseia no cálculo das derivadas parciais do erro em relação aos pesos e aos vieses. Essas derivadas indicam a direção e a magnitude das mudanças necessárias nos pesos para reduzir o erro (BRANDÃO, 2023).

A equação de minimização de erro é essencial nesta etapa, permitindo quantificar a discrepância entre a saída predita e o valor desejado. Ao subtrair a saída real da desejada, obtém-se o erro, que serve como guia para o ajuste dos pesos e vieses (BRANDÃO, 2023).

Os novos pesos calculados durante a retropropagação são então utilizados pelo algoritmo de otimização, que busca iterativamente encontrar os valores ideais para minimizar o erro global da rede. Esse processo é repetido ao longo de múltiplas iterações até que a rede atinja um desempenho satisfatório, ajustando-se de maneira adaptativa aos padrões presentes nos dados de treinamento (BRANDÃO, 2023).

Dessa forma, a retropropagação desempenha um papel crucial no aprendizado de RNAs, permitindo que elas se adaptem e generalizem a partir dos dados disponíveis, melhorando gradualmente seu desempenho ao longo do tempo.

Além dos pesos e vieses, nas redes neurais artificiais (RNAs), outros hiperparâmetros desempenham um papel crucial no processo de treinamento. Dois desses hiperparâmetros fundamentais são as épocas e o tamanho do lote.

As épocas referem-se ao número de vezes que a RNA realiza ciclos completos de treinamento, processando todo o conjunto de dados de treinamento. Durante cada época, a RNA ajusta seus pesos e vieses com base no algoritmo de retropropagação, buscando minimizar o erro global. O critério de parada determina quando o treinamento deve ser interrompido, podendo ser definido com base na convergência do erro ou em um número predeterminado de épocas (FERREIRA, 2022).

O tamanho do lote, por sua vez, representa a quantidade de amostras de dados que são utilizadas em cada passo de atualização dos pesos durante uma época. Em vez de ajustar os pesos após cada amostra individualmente (abordagem conhecida como gradiente estocástico), o treinamento pode ser mais eficiente computacionalmente ao agrupar várias amostras em lotes. Este processo é conhecido como gradiente em lote. O tamanho do lote influencia a estabilidade do treinamento e a utilização eficiente dos recursos computacionais (FERREIRA, 2022).

Esses hiperparâmetros, junto com a retropropagação, desempenham um papel em conjunto no treinamento eficaz de RNAs. O ajuste adequado das épocas e do tamanho do lote é crucial para garantir que a RNA aprenda de forma robusta, evitando tanto a subestimação da capacidade de generalização quanto o ajuste excessivo aos dados de treinamento.

Ao considerar esses hiperparâmetros, o processo de treinamento torna-se uma tarefa complexa de otimização, onde a interação entre eles impacta diretamente no desempenho final da RNA. A combinação adequada de épocas e tamanho do lote será determinada empiricamente, muitas vezes exigindo experimentação para encontrar a configuração ótima para o problema específico em questão.

Assim, ao integrar a retropropagação com a manipulação cuidadosa de hiperparâmetros como épocas e tamanho do lote, as RNAs podem ser treinadas de maneira mais eficiente e eficaz, adaptando-se de forma dinâmica aos padrões complexos presentes nos dados de treinamento (FERREIRA, 2022).

3.3 Greedy Randomized Adaptive Search Procedure

A meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) gera soluções que conseguem escapar de ótimos locais após o procedimento de busca local. Geralmente começa de uma solução vazia e a cada passo, escolhe um elemento para compor a sua solução, que é definida como Lista Restrita de Candidatos (LRC). O GRASP é um algoritmo guloso, visto que a cada passo, ele escolhe o melhor candidato para função de avaliação (GASPAR CUNHA; TAKAHASHI; HENGGELER ANTUNES, 2012).

O GRASP depende de dois hiperparâmetros e é simples de ser implementado. Ele é facilmente paralelizável. Cada interação é diferente das outras. Embora, por não ter memória e inteligência ao longo do processo de busca, ele pode repetir soluções (GASPAR CUNHA; TAKAHASHI; HENGGELER ANTUNES, 2012).

Os hiperparâmetros deste algoritmo é $\alpha \in [0,1]$, que controla a cardinalidade da LRC e número de iterações ou limite de tempo computacional. O parâmetro α igual a zero, representa uma busca aleatória e α igual a 1 é um algoritmo guloso. Uma busca gulosa escolhe apenas os melhores valores para compor a LRC. O Algoritmo 1 apresenta a construção do GRASP.

Algoritmo 1 Pseudo-código do GRASP padrão.

```
Hiperparâmetros: α, número_iterações:
enquanto i ≤ número_iterações faça
    Fase de construção
    Algoritmo de melhoria
    Atualiza melhor solução encontrada
    i = i+1
```

Algoritmo 1 Pseudo-código do GRASP padrão.

fim-do-enquanto

A calibração do parâmetro α, pode ser feito pelo GRASP reativo que é definido por (2) e (3), ele atualiza os valores de α ao longo das atualizações (GASPAR CUNHA; TAKAHASHI; HENGGELER ANTUNES, 2012).

$$q_i = \left(\frac{f^*}{A_i}\right)^{\delta} \tag{2}$$

$$p_i = \frac{q_i}{\sum_{j=1}^{j=k} q_j} \tag{3}$$

Onde, *f** é a melhor solução obtida até o momento; δ é um parâmetro de amplificação e A_i é a média das soluções, o parâmetro α passa a ser a probabilidade (3).

4 METODOLOGIA

São usados GRASP+RNA para otimizar de forma simultânea os hiperparâmetros da RNA, com intuito de prever a pressão de falha de dutos corroídos com defeitos axissimétricos. Para isso, foram considerados os hiperparâmetros: número de camadas (N_c), número de neurônios (N_n) e função de ativação (N_f). Embora, haja outros hiperparâmetros para serem otimizados (e.g., tamanho de lote; taxa de aprendizado).

4.1 Otimização GRASP

No código, foi criada uma LRC apenas para salvar as melhores soluções que já foram utilizadas. As funções de ativação utilizadas foram: (1) Sigmoid; (2) Tanh; e (3) ReLU (Figura 6); e foram definidas assim com esses números, de formato inteiro. O número de camadas e número de neurônios foram respectivamente 5 e 15. Utilizou-se uma LRC para salvar as melhores soluções e não se repetissem nas próximas iterações. E foram utilizadas 10 iterações.

4.2 RNA + GRASP reativo

Todas as inicializações começaram com o número de camadas, número de neurônios, função de ativação e parâmetro α respectivamente iguais a 5, 7, ReLU e 0,5. Eles poderiam começar como vazio, mas preferiu-se utilizar esses valores para garantir que ao menos a primeira iteração tenha o mesmo valor e facilite a comparação.

Na equação (2), foi utilizado o parâmetro δ igual a 10, visto que é o mais utilizado, para acelerar as respostas (GASPAR CUNHA; TAKAHASHI; HENGGELER ANTUNES, 2012). O critério de paradas é a apenas o número de iterações.

4.3 Dados para treino

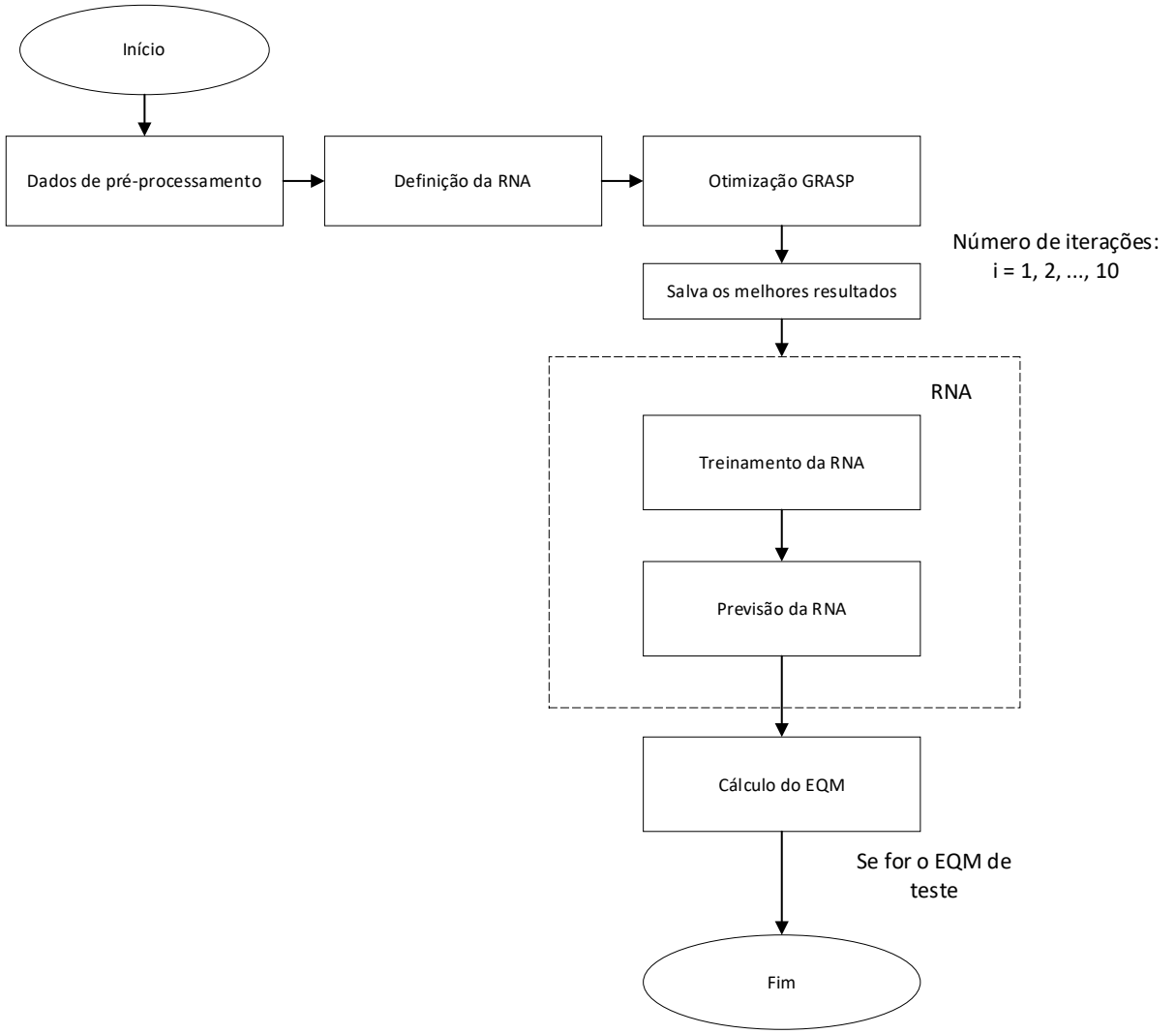
Os dados para treinamento utilizados para validação da RNA são os mesmos utilizados por Ferreira (FERREIRA, 2022). Os arquivos do RBP vêm em formato de texto com extensão "*.txt*" e são organizados em colunas em: (i) posição longitudinal (mm); (ii) espessura (mm).

Foram também utilizados 9612 casos sintéticos (i.e., são modelos que simulam defeitos de corrosão; eles foram criados a partir de campos estocásticos de perfis de espessura e esses

modelos têm suas pressões de falha obtida através de MEF (FERREIRA, 2022)), sua utilização se deve a escassez de dados reais para treinar uma RNA de forma adequada. Esses casos estão presentes em um arquivo *pickle* (.pkl), que é um formato de compressão, e é específico do PYTHON (PYTHON, 2024). Os dados foram divididos em 70% para treino, 20% para validação e 10% de teste (FERREIRA, 2022). A regressão da RNA é feita através de Erro Quadrático Médio (EQM), visto que é um dos melhores parâmetros para modelos de regressão (HAYKIN, 2000).

Em Ferreira (FERREIRA, 2022), foi usado a análise multiresolução, a Transformada *Wavelet* Discreta (TWD), como filtro para reduzir as variações das espessuras remanescentes dos dutos corroídos (i.e., manteve os dados de entrada com praticamente a mesma magnitude e reduziu a quantidade de dados). O que permitiu que os dados que serão utilizados neste trabalho, não apresentem muita variação e ainda reduziu a quantidade de dados, diminuindo a probabilidade de a RNA errar. Um fluxograma do que o código faz está presente na Figura 7.

Figura 7 - Fluxograma da metodologia GRASP+RNA.



Fonte: Autora, 2024.

4.4 Google Colaboratory

O código foi desenvolvido meticulosamente seguindo os procedimentos delineados nas subseções anteriores. A linguagem escolhida foi Python (PYTHON, 2024). A execução do código ocorreu no ambiente do *Google Colaboratory* (Colab), um serviço de *notebooks* hospedados que se baseia na tecnologia *Jupyter* (JUPYTER, 2024). Vale ressaltar que o Colab oferece uma notável facilidade de configuração, eliminando a necessidade de procedimentos complexos de instalação, além de proporcionar acesso gratuito a unidades de processamento gráfico (GPUs) (COLAB, 2024).

A organização estruturada do Colab segmenta o código em células, o que não apenas facilita a compartimentalização e compreensão, mas também permite que essas células sejam compartilhadas, editadas e executadas por colaboradores distintos. Essa abordagem colaborativa promove a interação entre membros da equipe, simplificando o compartilhamento de ideias e resultados, tornando o desenvolvimento do projeto mais dinâmico e acessível a uma audiência mais ampla.

5 RESULTADOS

Nesta seção são apresentados os resultados obtidos a partir das simulações e também, serão comparados com os resultados obtidos por Ferreira (2022) e Brandão (2023).

5.1 Problema original

Ferreira (2022) desenvolveu modelos para previsão de pressão de falhas em dutos corroídos a partir de dados do RBP. A Tabela 1 apresenta uma amostra de um desses arquivos "*.txt*", que contém os dados dos RBP, em uma determinada posição do duto.

Tabela 1 - Amostra de dados de um perfil *river bottom*.

Posição Longitudinal (mm)	**Espessura (mm)**
0	12,093
1	12,088
3	12,099
5	12,101
6	12,121
8	12,134
10	12,143
11	12,208
13	12,237
15	12,260

Fonte: (FERREIRA, 2022).

Foram utilizados 9612 casos sintéticos (casos criados a partir de campos estatísticos de perfis de defeitos; seu intuito é ser utilizado para treinar a RNA devido à escassez de dados reais), que correspondiam a 12 tipos de casos de defeitos de corrosão, totalizando aproximadamente 801 casos para cada tipo. Os dados são filtrados através de TWD e só então poderão ser utilizados na RNA (FERREIRA, 2022).

5.2 Simulação do RNA+GRASP reativo

Antes da realização das simulações do modelo RNA+GRASP reativo, foi necessário definir alguns hiperparâmetros gerais no código. Esses hiperparâmetros são: iterações, épocas, camadas ocultas, número de neurônios e variação do α.

Inicialmente a LRC será inicializada com os valores já descritos na seção (RNA + GRASP reativo). O parâmetro α está variando entre 0 e 1,0; o número de neurônios variando entre 1 e 15; número de camadas variando entre 1 e 5; as funções de ativação podendo ser Tanh, Sigmoid e ReLu. Os melhores valores se baseiam no Erro Quadrático Médio (EQM), visto que ele tem o objetivo de comparar a diferença de um resultado e seu valor inicial (i.e., compreende o erro de previsão). Então, as soluções que obtiverem os melhores valores não irão se repetir enquanto o código estiver realizando as 10 iterações e isso é importante, pois o GRASP não possui memória e evita dele repetir uma solução já considerada "melhor".

A escolha de 10 iterações se deve basicamente a altos custos computacionais. O número de épocas foi escolhido como 30 (onde foi encontrado por tentativa e erro), visto que já obtém bons resultados, com pouco tempo de execução. O algoritmo utilizado para a otimização dos hiperparâmetros da RNA é o *Adam*, o código utilizado está presente no Anexo A. O caso original está presente na Tabela 2, onde o melhor resultado está destacado na cor cinza.

Tabela 2 - Principais resultados da RNA+GRASP no caso original.

Rodada	**Camadas**	**Neurônios**	**Função de ativação**	**EQM**	**α**	**Tempo (s)**
1	3	1	Tanh	8,7933	0,40	62,92
2	2	13	ReLU	0,2484	0,20	56,75
3	5	14	Tanh	0,6313	0,90	56,44
4	3	2	ReLU	1,5262	0,90	57,59
5	5	7	Sigmoid	3,8570	0,30	62,49
6	4	10	ReLU	0,0397	0,90	63,44
7	4	4	Sigmoid	6,0158	0,50	53,72
8	1	9	ReLU	323,0952	0,20	59,64
9	2	8	ReLU	323,0923	0,70	59,99
10	5	4	Sigmoid	0,0617	0,50	711,26

Fonte: Autora, 2024.

Apesar do número de épocas ser um pouco alto (30 épocas), foi obtido um resultado muito satisfatório com erro quadrático médio de 3,97% e em 63,44 segundos. Apenas a última rodada levou um pouco mais de tempo para convergir. O melhor resultado foi obtido na 6ª rodada, com a seguinte configuração de RNA: 4 camadas com 10 neurônios; sua função de ativação foi a ReLU, o valor de α foi de 0,90.

5.3 Sensibilidade RNA+GRASP

O intuito de analisar a sensibilidade da RNA com o GRASP é analisar se ao reduzir os valores de número de camadas, neurônios e épocas, os resultados continuam bons, com a justificativa de utilizar menos recursos computacionais e encontrar a solução mais rapidamente. Já nos casos em que os valores de número de camadas, neurônios e épocas, são maiores, tem a justificativa de ver o quão preciso esse modelo pode ser. Os casos propostos neste trabalho estão presentes na Tabela 3.

Tabela 3 - Limites de variáveis para os quatro casos considerados.

Caso	Limites de Camadas	Limite de Neurônios	Épocas	Funções de ativação
C00	1-5	1-15	30	Sigmoid, Tanh e ReLU
C01	1-10	1-30	20	Sigmoid, Tanh e ReLU
C02	1-10	1-30	30	Sigmoid, Tanh e ReLU
C03	1-5	1-15	20	Sigmoid, Tanh e ReLU

Fonte: Autora, 2024.

Os valores originais dos limites são 5 camadas ocultas, 15 neurônios e 30 épocas. Que passam a ser 10 camadas ocultas, 30 neurônios e 20 épocas (as épocas variam, mas não são consideradas na otimização, apenas para fins de análise de sensibilidade). Assim, é possível analisar a viabilidade das soluções e do modelo.

5.3.1 Primeiro caso (C01): O dobro dos limites e 20 épocas

Neste caso os limites foram dobrados e o número de épocas foi reduzido a 20. Assim, foi obtido um resultado um pouco melhor que o anterior. A justificativa se deve a aleatoriedade dos valores, pode ser que o conjunto escolhido da vez, esteja com maior potencial para melhores soluções e também mais recursos de camadas e neurônios. A Tabela 4 apresenta os valores obtidos nestas dez rodadas do caso um. São então utilizadas 10 camadas, 14 neurônios e função de ativação Sigmoid, mas como o número de épocas é menor, o tempo também foi totalizando 48,59 segundos para chegar nessa solução.

Tabela 4 - Principais resultados da RNA+GRASP no caso um.

Rodada	**Camadas**	**Neurônios**	**Função de ativação**	**EQM**	**α**	**Tempo (s)**
1	7	24	ReLU	0,1516	0,8	55,71
2	7	14	Sigmoid	0,0814	0,7	40,89
3	5	18	Sigmoid	11,7381	0	56,76
4	6	28	Tanh	0,1844	0,5	53,93
5	9	22	Sigmoid	1,6803	0,4	49,84
6	4	11	Sigmoid	0,0634	0,8	46,91
7	10	5	Sigmoid	0,0652	0,4	55,40
8	10	14	Sigmoid	0,0373	0,4	48,59
9	4	9	ReLU	3,1981	0,6	54,74
10	6	11	ReLU	0,0585	0,6	610,23

Fonte: Autora, 2024.

5.3.2 Segundo caso (C02): O dobro dos limites e 30 épocas

Neste caso dois, os limites utilizados foram dobrados e apenas o número de épocas se manteve igual ao caso original. O resultado obtido foi o melhor entre todos os casos. A Tabela 5 apresenta os valores obtidos nestas dez rodadas do caso dois. No caso dois, utiliza-se 5 camadas, 23 neurônios e função de ativação Tanh, o tempo foi 62,87 segundos.

Tabela 5 - Principais resultados da RNA+GRASP no caso dois.

Rodada	Camadas	Neurônios	Função de ativação	EQM	α	Tempo (s)
1	5	9	Sigmoid	0,0436	0	60,76
2	7	5	Tanh	0,0729	0,2	65,27
3	4	10	Sigmoid	0,0814	0	63,93
4	4	15	Tanh	0,0506	0,8	61,61
5	5	9	Tanh	7,0052	0,9	66,97
6	5	1	Tanh	0,0997	0,1	64,69
7	1	22	Sigmoid	0,0514	0,5	77,59
8	5	23	Tanh	0,0234	0,4	62,87
9	1	12	Sigmoid	316,2938	0	67,71
10	1	28	ReLU	0,0419	0,8	747,80

Fonte: Autora, 2024.

5.3.3 Terceiro caso (C03): Limites originais e 20 épocas

Neste último caso, foram utilizados os limites originais e 20 épocas. O resultado obtido foi o pior entre os casos analisados. A Tabela 6 apresenta os valores obtidos nestas dez rodadas do caso três. No caso três o “melhor” resultado utiliza 6 camadas, 26 neurônios e função de ativação ReLU, o tempo foi 46,44 segundos.

Tabela 6 - Principais resultados da RNA+GRASP no caso três.

Rodada	Camadas	Neurônios	Função de ativação	EQM	α	Tempo (s)
1	9	22	ReLU	5,0147	0,1	58,95
2	6	26	ReLU	0,0943	0,4	46,44
3	3	17	Sigmoid	4,8487	0,4	55,33
4	7	17	Sigmoid	14,1307	0,2	61,27
5	3	8	Tanh	8,4367	0,3	53,08

Rodada	Camadas	Neurônios	Função de ativação	EQM	α	Tempo (s)
6	1	26	ReLU	53,3426	0,7	42,07
7	9	3	Tanh	1,0706	0,2	47,28
8	5	11	ReLU	2,0282	0	60,81
9	9	12	Sigmoid	16,5082	0,8	64,32
10	10	1	ReLU	16,6259	0,4	519,82

Fonte: Autora, 2024.

5.4 Escolha do melhor caso

Nota-se que não houve um padrão na escolha das funções de ativação, nem nos outros hiperparâmetros. Muito se deve ao fato do método GRASP sempre buscar um número aleatório de acordo com os limites que foram passados para ele. Diferente de outros métodos, que utilizam outras formas de se aproximar do valor ótimo. Por isso, não há padrão linear na melhora das soluções encontradas em nenhuma rodada de qualquer caso analisado.

Observa-se também que o número de épocas (o quanto a rede é treinada a mais) pode não ser tão relevante, se for feito a correta escolha do *tradeoff* (perda da qualidade em troca de outro aspecto) entre viabilidade de solução e tempo computacional. Mas ainda assim, a melhora das soluções advém do maior número de neurônios e camadas. O número de épocas permite maior refinamento no treino das RNA, mas não é tão fácil notar utilizando o método GRASP.

Para facilitar na visualização da melhora ou piora dos casos analisados, fez-se a comparação entre o EQM, a diferença percentual (equação (4)), a média e o Desvio Padrão (DP) das rodadas.

$$\Delta\,(\%) = 100 * \frac{EQM_{Ck} - EQM_{C00}}{EQM_{C00}}, \qquad k = 1, 2, 3 \tag{4}$$

A média é definida pela equação (5).

$$\overline{EQM} = \sum \frac{EQM_i}{n}, \qquad i = 1, 2, \dots, 10 \; e \; n = 10 \tag{5}$$

E o Desvio Padrão (DP) está presente na equação (6).

$$DP = \sqrt{\frac{\sum(EQM_i - \overline{EQM})^2}{n}}, \qquad i = 1,2,\dots,10 \; e \; n = 10 \tag{6}$$

A comparação descrita anteriormente está presente na Tabela 7.

Tabela 7 - Comparativo entre erros quadráticos médios para todos os casos.

Rodada	C00	C01		C02		C03	
	EQM	EQM	Δ (%)	EQM	Δ (%)	EQM	Δ (%)
1	8,7933	0,1516	-98,2760	0,0436	-99,5042	5,0147	-42,9714
2	0,2484	0,0814	-67,2303	0,0729	-70,6522	0,0943	-62,0370
3	0,6313	11,7381	1759,3537	0,0814	-87,1060	4,8487	668,0501
4	1,5262	0,1844	-87,9177	0,0506	-96,6846	14,1307	825,8747
5	3,8570	1,6803	-56,4351	7,0052	81,6230	8,4367	118,7374
6	0,0397	0,0634	59,6977	0,0997	151,1335	53,3426	134264,2317
7	6,0158	0,0652	-98,9162	0,0514	-99,1456	1,0706	-82,2035
8	323,0952	0,0373	-99,9885	0,0234	-99,9928	2,0282	-99,3723
9	323,0923	3,1981	-99,0102	316,2938	-2,1042	16,5082	-94,8906
10	0,0617	0,0585	-5,1864	0,0419	-32,0908	16,6259	26846,3533
Média	66,7361	1,7258	120,6091	32,3764	-35,4524	12,2101	16234,1772
DP	135,1432	3,6686	578,1319	99,7823	87,9556	15,7350	42312,7237

Fonte: Autora, 2024.

Além do EQM, é importante também comparar o tempo de resposta. A variações utilizadas, média e DP, são análogas às equações (3), (4) e (5).

Tabela 8 - Comparativo entre erros dos tempos para todos os casos.

Rodada	C00	C01		C02		C03	
	Tempo (s)	Tempo (s)	Δ (%)	Tempo (s)	Δ (%)	Tempo (s)	Δ (%)
1	62,92	55,71	-11,46	60,76	-3,43	58,95	-6,31
2	56,75	40,89	-27,95	65,27	15,01	46,44	-18,17
3	56,44	56,76	0,57	63,93	13,27	55,33	-1,97
4	57,59	53,93	-6,36	61,61	6,98	61,27	6,39
5	62,49	49,84	-20,24	66,97	7,17	53,08	-15,06
6	63,44	46,91	-26,06	64,69	1,97	42,07	-33,69
7	53,72	55,40	3,13	77,59	44,43	47,28	-11,99

Rodada	C00	C01		C02		C03	
	Tempo (s)	**Tempo (s)**	**Δ (%)**	**Tempo (s)**	**Δ (%)**	**Tempo (s)**	**Δ (%)**
8	59,64	48,59	-18,53	62,87	5,42	60,81	1,96
9	59,99	54,74	-8,75	67,71	12,87	64,32	7,22
10	711,26	610,23	-14,20	747,80	5,14	519,82	-26,92
Média	124,42	107,30	-12,99	133,92	10,88	100,94	-9,85
DP	206,22	176,78	10,47	215,75	13,03	147,36	13,86

Fonte: Autora, 2024.

5.5 Análise comparativa

Para comparar com Ferreira (2022) e Brandão (2023), será utilizado o caso dois, visto que o mesmo obteve o melhor resultado de EQM. Mas para isso, a RNA será treinada utilizando o mesmo número de épocas que foram utilizadas pelos outros autores (2000 épocas) e o resultado da comparação está apresentado na Tabela 9.

Tabela 9 - Comparação dos hiperparâmetros.

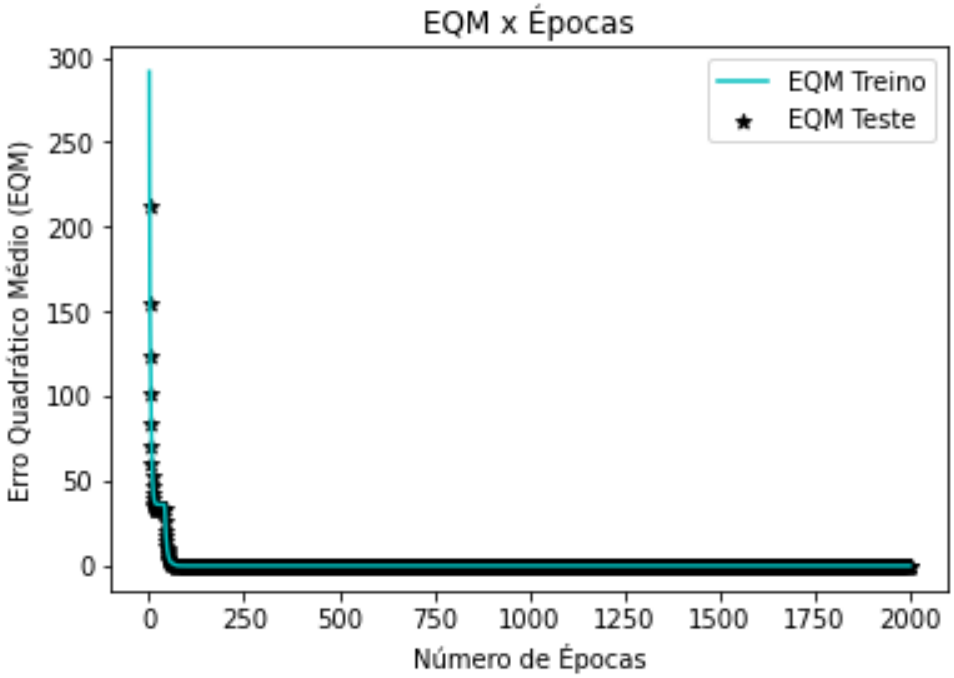
Hiperparâ-metros	C02	(BRANDÃO, 2023)	(FERREIRA, 2022)	Δ (%) (BRAN-DÃO, 2023)	Δ (%) C02
Camadas	5	8	3	166,6667	66,6667
Neurônios	23	28	15	86,6667	53,3333
Função de ativação	Tanh	ReLU	ReLU	-	-
EQM treino	0,0076	0,0146	0,0162	-9,8765	-53,3851
EQM teste	0,0058	0,0071	0,0111	-36,0360	-47,7817
R^2	0,9998	0,9998	0,9997	0,0100	0,0138

Fonte: Autora, 2024.

O parâmetro R^2 representa a qualidade do ajuste de um modelo de regressão, variando entre 0 e 1. Quando 0, significa que a regressão não está devidamente ajustada; quando 1, significa que está bem ajustada com o modelo (CHEIN, 2019).

O histórico do EQM com relação ao número de épocas no caso dois está apresentado na Figura 8.

Figura 8 - Histórico de Erro Quadrático Médio x Épocas, do caso dois.



Fonte: Autora, 2024.

Conforme a Tabela 9 e Figura 8, é observado que a arquitetura apresentada no caso 2 é melhor que a original (visto que, ela obteve um resultado mais preciso, com menor EQM e R^2) e mais complexa que a de Ferreira (2022), porém é menos complexa que a de Brandão (2023) e apresenta maior precisão. O que é possível observar que o GRASP reativo junto com a RNA buscou soluções mais complexas, que difere da proposta de Ferreira (2022) na busca de uma solução com menor custo computacional, embora, o GRASP reativo obteve um resultado mais preciso.

O coeficiente de determinação está ainda mais próximo de 1 que o trabalho de Ferreira (2022). Ele é idêntico ao valor do trabalho de Brandão (2023), evidenciando que a regressão é capaz de prever a pressão de falha de forma precisa. E fortalece a ideia de Brandão (2023), referente a estruturas mais complexas que resultam em melhor desempenho.

6 CONCLUSÃO

O presente trabalho teve o objetivo de aperfeiçoar o modelo de RNA que faz previsão de pressão de falha em dutos corroídos com defeitos axissimétricos. Para isso, foi utilizado a meta-heurística GRASP na sua variante reativa, com o intuito de otimizar hiperparâmetros da RNA. Sendo assim, passou-se a utilizar a RNA+GRASP reativo, para buscar a otimização dos hiperparâmetros: (i) número de camadas; (ii) número de neurônios; e (iii) função de ativação.

A função que foi minimizada na RNA+GRASP reativo foi o EQM, ela faz o comparativo entre o erro entre o valor real e o valor previsto, ou seja, o intuito era deixar o EQM o mais próximo de zero possível, sem demorar muito e sem consumir muitos recursos computacionais.

Como resultado, obteve-se uma RNA mais complexa que no trabalho de Ferreira (2022), porém, menos complexa que no de Brandão (2023). E foi obtida em menos tempo que a de Ferreira (2022); visto que, a de Ferreira (2022), foi feita através de tentativa e erro, onde resultou em um incremento ainda mais significativo no tempo necessário para gerar uma solução viável. E neste trabalho também se obteve o tempo menor para a solução viável que a obtida por Brandão (2023). Essa rede obtida também mostrou ter bastante capacidade de previsão de falha em dutos corroídos, visto que seu resultado foi superior aos dos trabalhos de Ferreira (2022) e Brandão (2023).

As vantagens observadas com a utilização do modelo proposto é de não precisar ficar testando manualmente os hiperparâmetros de otimização para poder encontrar um valor viável. Visto que, o modelo faz isso de forma automática e com valores precisos. Logo, faz-se indispensável a utilização da meta-heurística GRASP junto com a RNA para fazer estudos de previsão de falha em dutos corroídos, pois evita o desperdício de recursos computacionais e humanos. Outra vantagem desse modelo é ser possível otimizar outros hiperparâmetros das RNA (e.g., número de épocas), ou mesmo serem utilizados em RNAs que buscam prever outros resultados, não só apenas a pressão de falha.

Por outro lado, como o método GRASP busca valores de forma aleatória, não se torna viável definir um outro método de parada por ele, há não ser linear (podendo acabar se afastando de um valor ótimo) sendo mais prático utilizar como parada apenas o número de iterações. Outro ponto negativo é ele não ter memória, que pode acabar repetindo as soluções que ele já avaliou e pode acabar deixando o código mais lento do que deveria.

Embora haja algumas desvantagens no método GRASP+RNA, a utilização dele se faz imprescindível perante otimizações de hiperparâmetros da RNA, visto que, ao utilizar tentativa e erro, ocasiona uma perda de trabalho humana desnecessária e o GRASP consegue encontrar a otimização de hiperparâmetros de uma RNA em poucos segundos. Portanto, esse trabalho se mostra promissor para avaliação de problemas de otimização de hiperparâmetros de RNA, ou até mesmo de classificação.

6.1 Trabalhos futuros

Como sugestão para trabalhos futuros, sugere-se a utilização do GRASP antes do PSO (ou outra meta-heurística), com o intuito de não só acelerar o processo de busca pela otimização dos hiperparâmetros, mas também para o melhor refinamento dos hiperparâmetros. Outra opção seria a utilização de outras meta-heurísticas que melhor se adeque a esses problemas de mínimo local, e.g., Sewing *Training-Based Optimization* (STBO), que é baseado em aprendizado humanos; algoritmos meméticos; Busca Tabu.

REFERÊNCIAS

AMAYA-GÓMEZ, R. et al. Modeling of pipeline corrosion degradation mechanism with a Lévy Process based on ILI (In-Line) inspections. **International Journal of Pressure Vessels and Piping**, v. 172, p. 261–271, maio 2019.

ANGEL-BELLO, F. R.; GONZÁLEZ-VELARDE, J. L.; ALVAREZ, A. M. Greedy Randomized Adaptive Search Procedures. In: ALBA, E.; MARTÍ, R. (Eds.). . **Metaheuristic Procedures for Training Neutral Networks**. Boston, MA: Springer US, 2006. p. 207–223.

ANSYS. **Ansys**. Disponível em: <https://www.ansys.com/>. Acesso em: 21 jul. 2022.

BRANDÃO, B. C. **Otimização por enxame de partículas de rede neural artificial para previsão de pressão de falha em dutos corroídos**, 2023.

BRUÈRE, V. M. et al. Failure pressure prediction of corroded pipes under combined internal pressure and axial compressive force. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, v. 41, p. 172–182, 14 abr. 2019.

CETESB. **Principais Acidentes**. Disponível em: <https://cetesb.sp.gov.br/emergencias-quimicas/tipos-de-acidentes/dutos/principais-acidentes/>. Acesso em: 5 set. 2022.

CHEIN, F. **Introdução aos modelos de regressão linear: um passo inicial para compreensão da econometria como uma ferramenta de avaliação de políticas públicas**. [s.l: s.n.].

CODE_ASTER. **Code_Aster - Salome-Meca**. Disponível em: <https://www.code-aster.org/spip.php?article303>. Acesso em: 21 jul. 2022.

COLAB. **Colaboratory**. Disponível em: <https://research.google.com/colaboratory/intl/pt-BR/faq.html>. Acesso em: 15 jan. 2024.

FERREIRA, A. D. M. et al. Multiresolution analysis and deep learning for corroded pipeline failure assessment. **Advances in Engineering Software**, v. 163, p. 17–31, dez. 2021.

FERREIRA, A. D. M. **Análise multiresolução e redes neurais profundas para avaliação de dutos corroídos**. [s.l.] UFPE, 2022.

GASPAR CUNHA, A.; TAKAHASHI, R.; HENGGELER ANTUNES, C. **Manual de computação evolutiva e metaheurística**. [s.l: s.n.].

HAYKIN, S. **Redes Neurais: Princípios e Prática**. 2ª ed. [s.l: s.n.].

JUPYTER. **Jupyter**. Disponível em: <https://jupyter.org/>. Acesso em: 15 jan. 2024.

MENDES, G. **MTTR, MTBF e MTTF – O que são esses indicadores?** Disponível em: <https://www.fm2s.com.br/mtbf-mttr-mttf/#:~:text=O MTTR é calculado dividindo,um item ou equipamento específico.>. Acesso em: 2 set. 2022.

MSC.PATRAN. **Patran - Complete FEA Modeling Solution**. Disponível em: <https://www.mscsoftware.com/product/patran>. Acesso em: 29 jul. 2022.

NITAHARA, A. **Após 16 anos, pescadores ainda não foram compensados por vazamento da Reduc**. Disponível em: <https://agenciabrasil.ebc.com.br/geral/noticia/2016-01/apos-16-anos-pescadores-ainda-nao-foram-compensados-por-vazamento-da-reduc>. Acesso em: 20 set. 2022.

PIMENTEL, J. T. et al. New procedure of automatic modeling of pipelines with realistic shaped corrosion defects. **Engineering Structures**, v. 221, n. 111030, p. 0141–0296, 2020.

PYTHON. **Python**. Disponível em: <https://www.python.org/>. Acesso em: 15 jan. 2024.

RK CONSULTING ENGINEERS. **Pipeline Integrity Assessment**. Disponível em: <https://www.rkconsult.nl/services/pipeline-integrity/>. Acesso em: 20 set. 2022.

SUN, J.; CHENG, Y. F. Assessment by finite element modeling of the interaction of multiple corrosion defects and the effect on failure pressure of corroded pipelines. **Engineering Structures**, v. 165, p. 278–286, jun. 2018.

VISMAR UK. **3D rendering of an in-line inspection ILI tool inspecting a subsea pipeline**. Disponível em: <https://www.shutterstock.com/pt/image-illustration/3d-rendering-line-inspection-ili-tool-1250953828>. Acesso em: 15 jan. 2024.

XIE, M.; TIAN, Z. A review on pipeline integrity management utilizing in-line inspection data. **Engineering Failure Analysis**, v. 92, p. 222–239, 2018.

ZOLOTAREV, V. M. **One-dimensional stable distributions**. Providence: American Mathematical Society, 1986.

ANEXO A

```
from google.colab import drive
drive.mount ('/content/drive/')

#Importar as bibliotecas
import os, re
import matplotlib.pyplot as plt
import pandas as pd
import time
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import confusion_matrix
import tensorflow as tf
import sys
sys.path.append('/content/drive/MyDrive/TCC-Soraia')
import getDataFeatures as gdf
import pywt
import DrawNN as dn
import numpy as np
import seaborn as sns
from func_auxiliares import thick, press_eval, press_eval_ansys, eval_data, set_features
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam, SGD, Adagrad
import random
import csv
#from google.colab import files
#uploaded = files.upload()

from func_auxiliares import thick, press_eval, press_eval_ansys, eval_data, set_features
```

```
def preprocess(filename):
  # file_AI_full_results='full_inout_AI.pkl'
  AI_full_results = pd.read_pickle(filename)
  AI_full_results = AI_full_results.head(9612)  # excluindo o rpb1
  wvlt = 'rbio2.2'
  re = 304.8

  eval_features, y_eval_list, ft_add = eval_data(wvlt, re)  # eval_features coefs da wavelet
  max_len_evfeat = len(max(eval_features, key=len))
  for i in range(len(eval_features)):
    if len(eval_features[i]) < max_len_evfeat:
          eval_features[i] = np.pad(eval_features[i], int((max_len_evfeat - len(eval_features[i]))
/ 2), mode='constant')
    if len(eval_features[i]) < max_len_evfeat:
         eval_features[i] = np.append(eval_features[i], 0)
    eval_features[i] = np.append(eval_features[i], ft_add[i])

    y_eval = pd.DataFrame(y_eval_list, columns=['Ult Press'])
    X_eval = pd.DataFrame(eval_features, columns=[str(x) for x in range(max_len_evfeat + 1)])

  list_features, ft = set_features(AI_full_results, wvlt, re)
  max_len_feat = len(max(list_features, key=len))
  for i in range(len(list_features)):
     if len(list_features[i]) < max_len_feat:
          list_features[i] = np.pad(list_features[i], int((max_len_feat - len(list_features[i])) / 2),
mode='constant')
     if len(list_features[i]) < max_len_feat:
         list_features[i] = np.append(list_features[i], 0)
     list_features[i] = np.append(list_features[i], ft[i])

  lf_pd = pd.DataFrame(list_features, columns=[str(x) for x in range(max_len_feat + 1)])
  x_data = lf_pd
  y_val = AI_full_results['Ult Press']
```

```
  y_val.to_csv('press_output.csv')
  x_data.to_csv('ann_input.csv')

    X_train, X_test, y_train, y_test = train_test_split(x_data, y_val, test_size=0.3,
random_state=None)

  # scale the feature data
  scaler = MinMaxScaler()
  scaler.fit(X_train)
    X_train = pd.DataFrame(data=scaler.transform(X_train), columns=X_train.columns,
index=X_train.index)
     X_test = pd.DataFrame(data=scaler.transform(X_test), columns=X_test.columns,
index=X_test.index)
    X_eval = pd.DataFrame(data=scaler.transform(X_eval), columns=X_eval.columns,
index=X_eval.index)

  return X_train, X_test, y_train, y_test

# Função da RNA
def RNA_regressao(param, dados_pre_proc):
    nr_camadas, nr_neuronios, func_ativacao = param
    nr_camadas, nr_neuronios = int(nr_camadas), int(nr_neuronios)
    func_ativ_dict = {1: "sigmoid", 2: "tanh", 3: "relu"}
    func_ativ = func_ativ_dict[int(func_ativacao)]
    X_train, X_test, y_train, y_test = dados_pre_proc

    input_dim = X_train.shape[1]
    model = Sequential()
    model.compile(loss="mse", optimizer=Adam(), metrics=["mse"])

    for _ in range(nr_camadas - 1):
        model.add(Dense(nr_neuronios, activation=func_ativ))
    model.add(Dense(1, activation=func_ativ))
    model.compile(loss="mse", optimizer=Adam(), metrics=["mse"])
```

```
    history = model.fit(X_train, y_train,
                        batch_size=128, epochs=20, verbose=2)
    prediction = model.predict(X_test.values)
    mse = mean_squared_error(y_test.values, prediction)
    return mse

def grasp_optimization(X_train, y_train, X_test, y_test, alpha_max, alphas,
melhores_resultados):
    input_dim = X_train.shape[1]
    num_classes = 1

    melhor_modelo = None
    melhor_mse = float('inf')

    rcl_camadas = []  # Lista de candidatos recentes para número de camadas
    rcl_neuronios = []  # Lista de candidatos recentes para número de neurônios
    rcl_ativacao = []  # Lista de candidatos recentes para função de ativação

    for i in range(iteracoes_grasp):
        nr_camadas = random.randint(1, 5)
        nr_neuronios = random.randint(1, 15)
        func_ativacao = random.choice([1, 2, 3])

        func_ativ_dict = {1: "sigmoid", 2: "tanh", 3: "relu"}
        func_ativ = func_ativ_dict[func_ativacao]

        model = Sequential()
        model.add(Dense(nr_neuronios, activation=func_ativ, input_dim=input_dim))

        for j in range(nr_camadas - 1):
            model.add(Dense(nr_neuronios, activation=func_ativ))

        model.add(Dense(1, activation=func_ativ))
        model.compile(loss="mse", optimizer=Adam(), metrics=["mse"])
```

```
        history = model.fit(X_train, y_train, batch_size=128, epochs=20, verbose=2)

        mse_atual = model.evaluate(X_test, y_test)[0]

        if mse_atual < melhor_mse:
            melhor_modelo = model
            melhor_mse = mse_atual

            # Adicionar à RCL apenas se não estiver presente
            if nr_camadas not in rcl_camadas:
                rcl_camadas.append(nr_camadas)
            if nr_neuronios not in rcl_neuronios:
                rcl_neuronios.append(nr_neuronios)
            if func_ativacao not in rcl_ativacao:
                rcl_ativacao.append(func_ativacao)

    # Escolher aleatoriamente entre os candidatos na RCL
    nr_camadas = random.choice(rcl_camadas)
    nr_neuronios = random.choice(rcl_neuronios)
    func_ativacao = random.choice(rcl_ativacao)

    # Armazenar os melhores resultados
    melhores_resultados.add((nr_camadas, nr_neuronios, func_ativacao, melhor_mse))

    # Retornar o modelo correspondente ao melhor MSE
    return melhor_modelo

def save_results_to_csv(results, filename='results.csv'):
    with open(filename, 'w', newline='') as csvfile:
        fieldnames = ['Rodada', 'Numero_Camadas', 'Numero_Neuronios', 'Funcao_Ativacao',
'MSE', 'Alpha', 'Tempo']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
```

```
        writer.writeheader()
        for i, result in enumerate(results):
            writer.writerow({
                'Rodada': i + 1,
                'Numero_Camadas': result[0],
                'Numero_Neuronios': result[1],
                'Funcao_Ativacao': result[2],
                'MSE': result[3],
                'Alpha': result[4],
                'Tempo': result[5]
            })

def GRASP_RNA_reativo(X_train, y_train, X_test, y_test, iteracoes_grasp, alphas, alpha_max):

    melhor_mse_global = float('inf')
    melhor_modelo_global = None
    melhor_predicao_global = None

    f_star = float('inf')
    mse_final = float('inf')
    it = 1
    m = len(alphas)
    probabilidade_alphas = np.ones(m) / m

    melhores_resultados = set()
    resultados_por_rodada=[]
    nr_camadas = 5  # Initialize nr_camadas before the while loop
    nr_neuronios = 7  # Initialize nr_neuronios before the while loop
    func_activion = "relu"  # Initialize func_activion before the while loop
    alpha = 0.5

    print("Before the loop - it:", it)
    while it <= iteracoes_grasp:
```

```
tart_time = time.time()

nr_camadas = random.randint(1, 10)  # Move the initialization inside the loop
nr_neuronios = random.randint(1, 30)  # Initialize nr_neuronios inside the loop
func_activion = random.choice([1, 2, 3])  # Initialize func_activion inside the loop

mse_final = RNA_regressao((nr_camadas, nr_neuronios, func_activion), dados_pre_proc)

start_time = time.time()
alpha_index = np.random.choice(m, p=probabilidade_alphas)
alpha = alphas[alpha_index]

# Adicionar à lista de resultados
    resultados_por_rodada.append((nr_camadas, nr_neuronios, func_activion, mse_final,
alpha, time.time() - start_time))
     melhor_modelo_encontrado = grasp_optimization(X_train, y_train, X_test, y_test,
alpha_max, alphas, melhores_resultados)

mse_final = melhor_modelo_encontrado.evaluate(X_test, y_test)[0]

if mse_final < f_star:
    f_star = mse_final
    pressao_falha_pred = melhor_modelo_encontrado.predict(X_test.values)

if it % iteracoes_grasp == 0:
    delta = 10
     mi_values = [grasp_optimization(X_train, y_train, X_test, y_test, alpha_max, alphas,
melhores_resultados)
                .evaluate(X_test, y_test)[0] for a in alphas]
    q_values = [np.array((f_star / mi) ** delta) for mi in mi_values]
    probabilidade_alphas = q_values / np.sum(q_values)
if mse_final < melhor_mse_global:
    melhor_mse_global = mse_final
```

```
            melhor_modelo_global = melhor_modelo_encontrado
            melhor_predicao_global = pressao_falha_pred

        # Adicionar à lista de resultados
            resultados_por_rodada.append((nr_camadas, nr_neuronios, func_activion, mse_final,
alpha, time.time() - start_time))

    # Exibir informações
        print(f"\nRodada {it} - Informações:")
        print(f"Melhor mse: {mse_final}")
        print(f"Número de Camadas: {nr_camadas}")
        print(f"Número de Neurônios: {nr_neuronios}")
        print(f"Função de Ativação: {func_activion}")
        print(f"Alpha ajustado: {alpha}")
        print(f"Tempo decorrido: {time.time() - start_time} segundos")

        it += 1

    # Salvar resultados em um arquivo CSV
    save_results_to_csv(resultados_por_rodada)

    print(pressao_falha_pred)
    print(f"Melhor MSE Global: {melhor_mse_global}")
    return melhor_modelo_encontrado, pressao_falha_pred, melhor_predicao_global

melhor_modelo_rna = []
filename = "full_inout_AI.pkl"  # Substitua pelo nome do seu arquivo .pkl
iteracoes_grasp = 10
alphas = np.arange(0.0, 1.0, 0.1)
alpha_max = 1.0

# Preprocess the data
X_train, X_test, y_train, y_test = preprocess(filename)
dados_pre_proc = (X_train, X_test, y_train, y_test)
```

```
melhores_resultados = set()

melhor_modelo_encontrado, pressao_falha_pred, melhor_predicao_global = GRASP_RNA_reativo(X_train, y_train, X_test, y_test, iteracoes_grasp, alphas, alpha_max)

# Acesse as camadas do modelo
camadas_do_modelo = melhor_modelo_encontrado.layers

# Número de camadas
numero_de_camadas = len(camadas_do_modelo)
print(f"Número de Camadas: {numero_de_camadas}")

# Número de neurônios em cada camada
neuronios_por_camada = [camada.get_config()['units'] for camada in camadas_do_modelo]
print(f"Número de Neurônios em cada Camada: {neuronios_por_camada}")

# Função de ativação em cada camada
ativacoes_por_camada = [camada.get_config()['activation'] for camada in camadas_do_modelo]
print(f"Função de Ativação em cada Camada: {ativacoes_por_camada}")

# Imprimir ou usar o resultado conforme necessário
# print("Melhor MSE Final:", melhor_mse_global)
print(pressao_falha_pred)
```