PONTÍFICA UNIVERSIDADE CATÓLICA DE CAMPINAS

FERNANDO DE FACIO ROSSETTI

AMBIENTE EXPERIMENTAL PARA OTIMIZAÇÃO MULTICRITÉRIO E HIERÁRQUICA DE PROBLEMAS DE LOCALIZAÇÃO DE FACILIDADES

CAMPINAS - SP

2025

PONTÍFICIA UNIVERSIDADE CATÓLICA DE CAMPINAS

ESCOLA POLITÉCNICA

FACULDADE DE ENGENHARIA DE COMPUTAÇÃO

CURSO DE CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

AMBIENTE EXPERIMENTAL PARA OTIMIZAÇÃO MULTICRITÉRIO E HIERÁRQUICA DE PROBLEMAS DE LOCALIZAÇÃO DE FACILIDADES

Trabalho/Projeto para o componente curricular "Projetos Empreendedores B" apresentado à Faculdade de Engenheria de Computaçao, do Curso de Ciência de Dados e Inteligência Artificial da Escola Politécnica da Pontifícia Universidade Católica de Campinas.

Orientador:
Prof. Me. Fernando Soares de Aguiar Neto

CAMPINAS - SP

2025

Sumário

Sumário **3**
1 Introdução **4**
1.1 Objetivos 6
1.1.1 Objetivo Geral 6
1.1.2 Objetivos Específicos 6
1.1.2.1 Objetivos Específicos de Alta Prioridade 6
1.1.2.2 Objetivos Específicos de Baixa Prioridade 7
1.2 Estrutura da dissertação 8
2 Metodologia **9**
2.1 Aplicação 11
2.1.1 Modelagem do problema 11
2.1.2 Algoritmos e critérios utilizados 11
2.1.3 Fluxo de Execução 11
2.2 Arquitetura 12
2.2.1 Servidor 12
2.2.2 Cliente 12
2.2.3 Serviços 13

REFERÊNCIAS **14**

1. INTRODUÇÃO

Grafos são estruturas matemáticas usadas para representar relações entre diferentes dados e são utilizadas para modelar problemas reais em diferentes domínios (MAJEED; RAUF, 2020). Muitas situações do mundo real podem ser convenientemente descritas por meio de um diagrama que consiste em um conjunto de pontos juntamente com linhas que unem certos pares desses pontos. Como exemplo, é possível conectar cidades pela distância, pessoas pelo nível de afinidade, computadores por redes de internet, átomos por ligações químicas, neurônios por sinapses, entre outros. Uma das principais vantagens de se utilizar grafos é a possibilidade de realizar análises tanto discretas quanto topológicas, permitindo modelar dados em uma dimensão adicional.

Em diversos campos de aplicação, a otimização combinatória consiste na busca por soluções ótimas ou aproximadas dentro de um conjunto de soluções possíveis, de maneira a maximizar ou minimizar determinados critérios específicos do domínio. Muitos desses problemas podem ser naturalmente formulados em termos de grafos. Um exemplo clássico é o Problema do Caixeiro Viajante, que busca encontrar a menor rota que parte de uma cidade, visita todas as demais e retorna à sua origem. Tais problemas costumam ser computacionalmente intensivos, exigindo cuidado especial na projeção e implementação de algoritmos eficientes.

O Problema de Localização de Facilidades, conhecido como *Facility Location Problem* (FLP), é um tipo de problema de otimização combinatória que consiste em decidir o melhor local para alocar uma facilidade, de forma a minimizar uma determinada função de custo. O problema foi inicialmente idealizado por Weber (1909), consistindo em encontrar pontos em um espaço euclidiano que minimizem a soma dos custos de transporte entre estes pontos e as facilidades existentes.

Ao longo do tempo, surgiram diversas variações do problema, que introduzem restrições ou ampliam o domínio de aplicação. Entre elas, destacam-se as variantes *capacitated*, que consideram que as facilidades possuem um número finito de recursos e podem não conseguir atender a todos os clientes, as variantes hierárquicas, que adicionam uma cadeia de facilidades, por exemplo, uma fábrica que atende uma facilidade intermediária, que por sua vez atende o cliente, e as variantes que impõem restrições de domínio, definindo se o espaço de alocação é discreto ou contínuo. Existem ainda muitos outros tipos que consideram questões específicos do domínio do problema mas todas seguem o princípio básico de alocação de maneira a minimizar uma função objetivo.

Além disso, o problema pode ser estendido para atender situações de decisão multicritério, *Multi-Criteria Decision Making* (MCDM), considerando diversos fatores na alo-

cação, tais como o tempo de deslocamento entre a facilidade e o cliente, o custo de abertura das facilidades, o número de serviços oferecidos, entre outros, podendo ser adaptados conforme o domínio do problema. Farahani, SteadieSeifi e Asgari (2010) destacam diversas produções acadêmicas que aplicam o FLP em diferentes áreas, como saúde, química, física, economia, psicologia, entre outras, evidenciando o potencial e a versatilidade desta formulação.

Existem diversos tipos de algoritmos e modelos matemáticos que podem ser usados para abordar o FLP, cuja escolha depende tanto da complexidade do domínio quanto da formulação inicial do problema. Um algoritmo que apresenta bom desempenho em um contexto específico nem sempre terá a mesma performance em outro, devido às diferentes restrições, dimensões e critérios presentes em cada domínio. Além disso, problemas multicritério e hierárquicos introduzem ainda mais variáveis e complexidade, tornando difícil prever qual abordagem será mais eficiente.

Sendo assim, torna-se relevante a criação de um ambiente interativo, flexível e genérico, capaz de acomodar diferentes representações do problema, permitir a implementação e comparação de múltiplos algoritmos, e fornecer métricas de desempenho que auxiliem na análise da eficácia e adequação das soluções em distintos cenários. Dessa forma, é possível realizar experimentos em um ambiente controlado, que posteriormente pode ser adaptado a domínios específicos pelo usuário, após a análise da viabilidade, fornecendo uma exploração acadêmica e pedagógica, e contribuindo para a validação de métodos em contextos diversos, promovendo uma compreensão mais profunda das forças e limitações de cada técnica.

Além dos algoritmos tradicionais de otimização, o ambiente proposto permite a aplicação de técnicas avançadas de análise de grafos, que podem ser utilizadas para reduzir a dimensionalidade do problema, extrair métricas topológicas relevantes, como centralidade e intermediação, ou mesmo para gerar agrupamentos de clientes em clusters, facilitando a definição de locais estratégicos para instalação de facilidades. Dessa forma, o usuário pode explorar diferentes representações do problema, combinando múltiplos critérios, estruturas hierárquicas e análises de grafos, tornando a plataforma extremamente versátil e adaptável a múltiplos domínios e contextos.

É importante destacar que embora o foco principal seja o FLP, a plataforma será implementada de forma genérica, permitindo a modelagem e análise de problemas em grafos de maneira geral, abrindo espaço para constantes evoluções nas ferramentas disponíveis e possibilitando aplicações em outros problemas de otimização combinatória que envolvam grafos.

Esta pesquisa busca contribuir para a área de otimização combinatória e modelagem matemática, fornecendo um ambiente genérico o suficiente para poder simular

problemas que podem ser representados por grafos, em principal, o problema da alocação das facilidades, de forma a atender diferentes domínios. A implementação permite que o usuário final crie e selecione os critérios, de forma a adequar o ambiente aos requisitos específicos de seu problema, além de fornecer diversas ferramentas para análise de grafos e simulação do problema, com dados e comparações que irão facilitar a otimização. O trabalho proporciona ao aluno o estudo de uma área com um grande número de pesquisas, abrindo espaço para possíveis desenvolvimentos acadêmicos futuros e contribuindo para o desenvolvimento de conceitos matemáticos e computacionais que são muito importantes para enriquecimento acadêmico e oportunidades profissionais.

1.1 Objetivos

1.1.1 Objetivo Geral

Implementar um programa interativo e flexível para o estudo do problema das localização das facilidades, permitindo que o usuário modele diferentes representações do problema, defina critérios personalizados de otimização e aplique múltiplos algoritmos e obtenha resultados que possibilitem a comparação entre técnicas distintas e a análise das métricas geradas.

1.1.2 Objetivos Específicos

Os objetivos de alta prioridade são os que fazem parte da aplicação e de seu funcionamento e são necessários estar prontos até o fim do projeto. Já os de baixa prioridade são objetivos opcionais que agregam no que já está feito, fornecendo funcionalidades extras que não são necessárias para o funcionamento da aplicação.

1.1.2.1 Objetivos Específicos de Alta Prioridade

- Interatividade: O sistema deve oferecer uma experiência amigável ao usuário, permitindo que ele interaja de forma intuitiva com os componentes do ambiente. Isso inclui seleção de nós, criação de grafos, ajustes de parâmetros e visualização dinâmica de resultados. A interatividade garante a facilidade de uso do problema, permitindo uma experimentação mais dinâmica, e servindo também como uma ferramenta pedagógica;
- Abundância de critérios: O usuário deve ter flexibilidade para definir quais critérios deseja otimizar, como distância, custo ou tempo de atendimento, e escolher qual algoritmo será executado entre os disponíveis. Essa liberdade permite explorar cenários variados e comparar resultados, aumentando a capacidade analítica do ambiente;

- Mudança de estrutura: O ambiente deve permitir que o usuário selecione o tipo de estruturação do problema a ser tratado, seja hierárquico, multicritério ou padrão. Isso garante que a plataforma seja adaptável a diferentes contextos e níveis de complexidade;
- Entrada: O sistema deve possibilitar que o usuário crie grafos manualmente de forma interativa ou importe grafos a partir de arquivos padronizados. Essa funcionalidade facilita a experimentação com dados reais ou simulados;
- Saída: O sistema deve fornecer informações claras e de fácil acesso e interpretabilidade sobre os resultados obtidos a partir da simulação, de modo que permita-o a comparar os resultados e métricas de forma facilitada;
- Persistência do grafo: o usuário pode salvar o estado de seu trabalho atual em um arquivo em seu computador e depois usa-lo como entrada no programa para resumir o uso de onde parou.
- Otimização: O programa deve entregar resultados próximos do ótimo, aplicando estratégias de modelagem e implementação eficientes. Isso assegura que o ambiente não seja apenas visual, mas também útil para análise de desempenho de algoritmos;
- Escalabilidade: O sistema deve lidar tanto com pequenos quanto grandes conjuntos de dados, mantendo desempenho consistente e sem erros. Essa característica é fundamental para que o ambiente possa ser utilizado com dados reais e portado em diferenças contextos independentemente do volume de dados.

1.1.2.2 Objetivos Específicos de Baixa Prioridade

- Entrada avançado: a existência de outras ferramentas que trabalham em cima de grafos implica em diferentes formas de serializar um grafo. Fazer com que a aplicação aceite arquivos em diferentes formatos;
- Saída avançado: geração de gráficos e análises automáticas a partir dos resultados obtidos;
- Persistência do usuário: salvar dados e ações de usuário para garantir recuperação de dados implica na manipulação de dados sensíveis que devem ser lidados com extrema cautela. Implementar gerenciamento de usuários em banco de dados de forma segura;
- Simulação passo a passo: inicialmente, a aplicação processará os algoritmos de forma automática, sem exibir cada etapa. A implementação de uma funcionalidade que permita ao usuário acompanhar, visualmente, cada passo executado pelo algoritmo

na interface é complexa, exigindo camadas adicionais de abstração e controle sobre o fluxo de execução.

1.2 Estrutura da dissertação

Está dissertação está organizada em quatro diferentes tópicos. O Capítulo 1, de caráter introdutório, apresenta uma contextualização de grafos e sua relação com otimização combinatória, especificando o problema-alvo e descrevendo oque será de forma breve assim uma justificativa.

O Capítulo 2 irá apresentar um referencial teórico, explorando os fundamentos teóricos de teoria de grafos, as várias interpretações do problemas de localização das facilidades, e detalhes técnicos sobre a implementação do programa, como os paradigmas de renderização usados, modelo de arquitetura, entre outros.

O Capítulo 3 trata dos procedimentos metodológicos que serão utilizados no desenvolvimento do ambiente, descrevendo quais serão os algoritmos implementados bem como detalhes da arquitetura e do fluxo de uso da aplicação.

O Capítulo 4 tem como objetivo a discussão dos resultados obtidos, focando em mostrar como os objetivos foram alcançados.

Por fim, a conclusão recapitulada o problema que está sendo abordado neste trabalho, sintetiza as principais contribuições e apresenta possíveis caminhos para pesquisas futuras e extensões do programa.

2. METODOLOGIA

A presente pesquisa adota uma abordagem de caráter exploratório, em virtude da ampla variedade de possibilidades de modelagem e implementação relacionadas ao problema de localização de facilidades. O processo de desenvolvimento seguirá a estratégia *bottom-up*, iniciando-se pela construção de uma versão mínima da aplicação, em seu estado mais elementar, e avançando gradualmente por meio de incrementos sucessivos no escopo do projeto. Tal estratégia busca garantir uma evolução controlada, permitindo a ampliação das funcionalidades de forma contínua e sistemática. Para viabilizar esse processo, foi previamente estabelecida uma base estrutural sólida, que assegura a estabilidade do sistema e reduz a necessidade de alterações profundas no código durante a inserção de novos módulos e serviços.

O andamento do projeto será orientado por metas semanais e pela dedicação contínua ao desenvolvimento. Devido à diversidade de componentes do projeto e à abordagem exploratória adotada, é esperado que surjam dificuldades ao longo do caminho, podendo ser necessário desconsiderar ou remover certas funcionalidades e, eventualmente, ajustar o fluxo de trabalho. Por essa razão, a elaboração de um cronograma rígido se torna praticamente inviável devido a natureza líquida da abordagem, já que mesmo que seja estabelecido, seu seguimento estrito pode não ser possível.

Diante disso, torna-se preferível estabelecer uma ordem de prioridade na implementação das funcionalidades, conforme descrito a seguir:

- Interface: A interface do programa é considerada a camada mais importante, sendo, portanto, a primeira a ser desenvolvida. O foco inicial está na criação de grafos de forma interativa, com atenção à escalabilidade e usabilidade. Ao longo do projeto, as demais funcionalidades serão integradas à interface, garantindo uma evolução contínua.

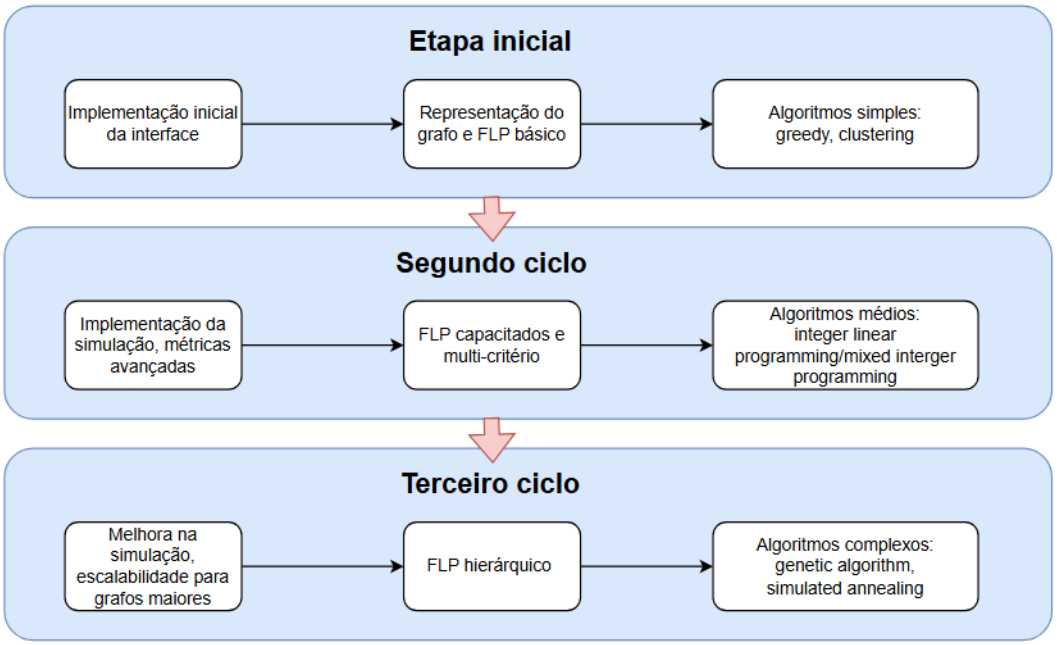
- Representação: Após a definição de uma interface funcional, a próxima etapa é consolidar a representação interna dos dados no back-end, com atenção à eficiência e modularidade. Isso inclui decisões sobre a melhor forma de estruturar grafos (por exemplo, listas ligadas ou matrizes), definição de interfaces e contratos, e organização hierárquica das entidades no contexto do FLP. Embora esta etapa seja relativamente rápida, ajustes podem ser necessários à medida que novas funcionalidades forem implementadas ou otimizações de código forem identificadas.

- Algoritmos: Com a interface e a representação interna bem estabelecidas, inicia-se a implementação dos algoritmos que estarão disponíveis para o usuário, permitindo a

realização de comparações e experimentos no ambiente.

A ideia central é adotar um modelo próximo ao *waterfall*, em que cada etapa deve estar suficientemente definida antes da seguinte ser iniciada. Entretanto, não é necessário que uma fase esteja completamente concluída para avançar, já que uma base sólida garante a continuidade do desenvolvimento, permitindo progressos de forma controlada e iterativa. A Figura 1 ilustra o processo em ciclos, onde o avanço para o próximo ciclo necessita de estabelecimento de uma base sólida no anterior

Figura 1 – Ciclo de desenvolvimento



Fonte: Elaboração própria.

É importante ressaltar que o ambiente de experimentação desenvolvido não possui caráter educacional. Seu objetivo não é o de ensinar conceitos relacionados a FLP e algoritmos correlatos, mas sim disponibilizar a pesquisadores, profissionais, estudantes e usuários familiarizados com o tema uma ferramenta prática para simulação. A aplicação contará com recursos para ajudar o processo de experimentação, acompanhado por um guia de utilização que orientará o usuário quanto às funcionalidades do programa e ao seu correto manuseio.

Assim como ocorre em problemas de otimização combinatória, a adoção de abordagens exploratórias nem sempre garante a obtenção da solução ótima. O desenvolvimento deste projeto por meio de múltiplas experimentações pode acarretar riscos e desafios, como tempo de execução elevado, dificuldades de escalonamento e eventuais problemas arquiteturais. Entretanto, é importante destacar que a aplicação não tem como

objetivo inovar ou resolver todos os problemas do domínio, funcionando apenas como um ambiente de implementação de técnicas já conhecidas. Limitações inerentes a essas técnicas estarão refletidas na aplicação, sendo responsabilidade do usuário compreender essas restrições. Caso algum método se mostre inviável, sua substituição é facilitada pela ampla disponibilidade de alternativas e recursos na literatura, permitindo a adaptação e continuidade do estudo de forma flexível e controlada.

2.1 Aplicação

2.1.1 Modelagem do problema

Esta seção visa descrever como os dados, como por exemplo o grafo, e os problemas estão estruturados no algoritmo. Poderão ser utilizados trechos de códigos ou diagramas UML para facilitar a visualização da representação.

2.1.2 Algoritmos e critérios utilizados

É fundamental destacar que, em virtude do caráter genérico da aplicação proposta, torna-se necessário avaliar diferentes critérios e algoritmos de forma conjunta, de modo a identificar aqueles que apresentam maior adequação ao domínio em questão. A utilização de certas técnicas depende das propriedades estruturais e matemáticas do problema tratado. Por exemplo, um algoritmo baseado em agrupamento contínuo não pode ser aplicado diretamente sobre um grafo cuja natureza é intrinsecamente discreta. Sendo assim, os algoritmos e critérios utilizados serão definidos e implementados ao longo do tempo de desenvolvimento do projeto

Ao longo do desenvolvimento do projeto, esta seção será utilizada para descrever os algoritmos e critérios disponíveis na aplicação.

2.1.3 Fluxo de Execução

O usuário final irá entrar no site a partir de um navegador, na qual terá a opção de importar um grafo a partir de um arquivo ou criar um grafo do zero a partir das ferramentas interativas disponíveis. O usuário poderá criar nós e conectar estes usando arestas, e caso clique em um nó, uma janela mostrando suas propriedades será exibida e nela o usuário poderá editar configurações sobre o nó, como marcar ele como um produtor ou consumidor, definir o peso da conexão com outras arestas, definir algumas propriedades relacionadas ao FLP como custo de implementação, entre outras. Tendo seu grafo pronto, o usuário poderá iniciar uma simulação, na qual será responsável por estruturar como será o problema de localização, isto é definir os critérios e o algoritmo a serem utilizados. Ao terminar a simulação, será exibido os resultados obtidos para o usuário, dando a possibilidade de ele

salvar estas informações. É importante lembrar que como um dos focos é comparação e otimização, o usuário pode definir diversas estruturas para o problema, selecionando conjuntos de critérios diferentes ou algoritmos diferentes para gerar todos os resultados de uma única vez.

Esta seção será posteriormente populada com várias imagens demonstrando um fluxo de uso padrão da aplicação, detalhando mais como os processos descritos acimas ocorrem de fato.

2.2 Arquitetura

2.2.1 Servidor

O back-end do programa será implementado utilizando o *Spring Framework*, um conjunto de ferramentas para desenvolvimento de aplicações Java que facilita a criação de sistemas robustos e escaláveis, com ênfase nos módulos *Spring Boot*, que permite configurações e inicialização rápida da aplicação, garantindo deploy eficiente, e *Spring Web*, que fornece suporte ao gerenciamento de requisições HTTP, garantindo comunicação eficiente entre o servidor e a interface do usuário. Essa escolha permite a construção de um back-end escalável, modular e de fácil manutenção, adequado à natureza genérica e experimental do ambiente proposto, sendo que a ampla utilização do Spring em projetos e sua performance superior em comparação com outros *frameworks* e bibliotecas que realizam funções similares reforçam sua adequação (CHOMA; CHWALEBA; DZIEńKOWSKI, 2023).

A linguagem utilizada, como mencionado anteriormente, será Java, cuja natureza orientada a objetos facilita a modelagem de relações hierárquicas, a definição de contratos entre estruturas e a reutilização de código por meio de herança e polimorfismo.

2.2.2 Cliente

A interface do programa será implementada utilizando o *framework React.js*, que se destaca por permitir o desenvolvimento de aplicações *Single Page Application* (SPA) de forma modular, por meio da reutilização de componentes. Essa abordagem torna o código mais organizado, facilita a manutenção e possibilita a criação de interfaces interativas e responsivas. O React também oferece atualização eficiente do DOM por meio do Virtual DOM, garantindo desempenho elevado mesmo em aplicações com grande quantidade de elementos gráficos.

Uma das principais vantagens do React, é a utilização de uma linguagem *JavaScript XML* (JSX) que é uma extensão da linguagem JavaScript que permite escrever sintaxe parecida com HTML dentro do código, de forma a escrever componentes de forma facilitada. Para aproveitar os benefícios da tipagem forte, será usada a variante *TypeScript*

XML (TSX). Como o navegador padrão não consegue interpretar estas linguagens, é necessário passar por um processo de compilação convertendo estas linguagens para JavaScript puro. Para isso será utilizado o *build tool Vite* que automatiza e otimiza todo o processo, incluindo a transformação de TSX em JavaScript, gerenciamento de módulos e atualização instantânea da interface durante o desenvolvimento.

A renderização dos grafos será realizada utilizando componentes *Scalable Vector Graphics* (SVG), que criam elementos diretamente no DOM do navegador, facilitando a interação e a manipulação individual dos objetos. Diferentemente da renderização baseada em *canvas*, que utiliza uma abordagem de *bitmap*, uma matriz de pixels coloridos, o SVG mantém o conhecimento do navegador sobre cada componente, permitindo atualizações e interações mais simples. Embora a renderização em *canvas* possa ser mais rápida, já que não precisa atualizar o DOM, ela exige que o desenvolvedor implemente grande parte da lógica de interação manualmente, o que aumenta a complexidade do código. Em sistemas com grande número de elementos gráficos, técnicas de renderização baseadas em *bitmap*, como *WebGL*, que aproveita a GPU para cálculos, tendem a apresentar melhor desempenho. Uma abordagem híbrida, que combina diferentes técnicas de renderização de acordo com o nível de zoom ou complexidade dos dados, é ideal em cenários onde o volume de informações pode variar entre simples e complexo.

2.2.3 Serviços

Conforme o desenvolvimento do programa, é possível a necessidade de implementar serviços para lidar com algoritmos complexos ou até mesmo garantir modularização de algum componente da aplicação. Um exemplo de serviço possível é uma API em linguagem Python que recebe dados do Servidor e utiliza-os para análises estatísticas ou computação de algum algoritmo. Os serviços implementados serão descritos nessa seção, dando ênfase no tipo de entrada e saída, bem como processamento interno.

REFERÊNCIAS

CHOMA, D.; CHWALEBA, K.; DZIEńKOWSKI, M. The efficiency and reliability of backend technologies: Express, django, and spring boot. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, v. 13, p. 73–78, 12 2023. Citado na página 12.

FARAHANI, R. Z.; STEADIESEIFI, M.; ASGARI, N. Multiple criteria facility location problems: A survey. *Applied Mathematical Modelling*, v. 34, n. 7, p. 1689–1709, 2010. ISSN 0307-904X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0307904X09003242>. Citado na página 5.

MAJEED, A.; RAUF, I. Graph theory: A comprehensive survey about graph theory applications in computer science and social networks. *Inventions*, v. 5, n. 1, 2020. ISSN 2411-5134. Disponível em: <https://www.mdpi.com/2411-5134/5/1/10>. Citado na página 4.

WEBER, A. *Über den Standort der Industrien*. Tübingen: J.C.B. Mohr, 1909. Citado na página 4.