UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DE SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO
Ciência da Computação

Daniel Gonçalves Mantuan

Desenvolvimento de Complemento para Software de Geoprocessamento na Otimização do Planejamento Florestal

ALEGRE-ES
2025

Daniel Gonçalves Mantuan

Desenvolvimento de Complemento para Software de Geoprocessamento na Otimização do Planejamento Florestal

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Marcelo Otone Aguiar

ALEGRE-ES

2025

Daniel Gonçalves Mantuam

Desenvolvimento de Complemento para Software de Geoprocessamento na Otimização do Planejamento Florestal

Trabalho de Conclusão de Curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em 21 de março de 2025.

COMISSÃO EXAMINADORA

Documento assinado digitalmente
MARCELO OTONE AGUIAR
Data: 21/03/2025 10:15:09-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Marcelo Otone Aguiar
Universidade Federal do Espírito Santo
Orientador

Documento assinado digitalmente
RODRIGO FREITAS SILVA
Data: 24/03/2025 13:55:09-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Rodrigo Freitas Silva
Universidade Federal do Espírito Santo

Documento assinado digitalmente
GERALDO REGIS MAURI
Data: 21/03/2025 10:29:59-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Geraldo Regis Mauri
Universidade Federal do Espírito Santo

Resumo

O Manejo Florestal Sustentável (MFS) enfrenta desafios significativos com a exploração inadequada da floresta causando danos à regeneração das espécies e ao meio ambiente. O planejamento das práticas de construção de infraestruturas como pátios, estradas e trilhas de arraste se mostram cruciais para minimizar a abertura da floresta e os custos monetários. Dada a relevância desse assunto, muitas pesquisas vêm disponibilizando resultados positivos na literatura acerca desse tema. No entanto, muitos desses estudos não oferecem soluções práticas aplicáveis ao planejamento real de empresas do manejo florestal. Diante disso, este trabalho buscou replicar um estudo importante recentemente publicado que obteve bons resultados na área e desenvolver um software implementando os métodos *Simulated Annealing* (SA) e o Dijkstra, visando disponibilizar uma ferramenta viável e útil para o MFS. O COMPFLOR demonstrou resultados competitivos com a literatura na eficiência da redução de custos e impactos ambientais, podendo auxiliar gestores na tomada de decisões estratégicas contribuindo para a redução de custos e preservação das áreas florestais.

Palavras-chave: Manejo florestal sustentável, *Simulated Annealing*, Dijkstra, planejamento otimizado, estruturas de exploração florestal.

Abstract

Sustainable Forest Management (SFM) faces significant challenges due to inadequate forest exploitation, causing damage to species regeneration and the environment. The planning of infrastructure practices such as yards, roads, and drag trails is crucial to minimize forest opening and monetary costs. Given the importance of this issue, many studies in the literature have provided positive results on this topic. However, many of these studies do not offer practical solutions applicable to the real planning of forestry companies. In this context, this paper aimed to replicate a significant study recently published, which achieved good results in the field, and to develop a software implementing Simulated Annealing (SA) and Dijkstra's methods, aiming to provide a viable and useful tool for SFM. COMPFLOR demonstrated competitive results compared to the literature in terms of cost reduction and environmental impact efficiency, potentially assisting managers in strategic decision-making, contributing to cost reduction and the preservation of forest areas.

Keywords: Sustainable forest management, Simulated Annealing, Dijkstra, optimized planning, forest exploitation structures.

Lista de figuras

Figura 1: Representação esquemática das etapas do manejo florestal..... 16
Figura 2: Abrangência das Sirenes em Dublin, Illinois, EUA..... 18
Figura 3: Algoritmo Dijkstra..... 22
Figura 4: Pseudocódigo do SA aplicado ao problema de alocação de pátios..... 25
Figura 5: Fluxograma das etapas para desenvolvimento do COMPFLOR..... 28
Figura 6: Função de leitura de vértices..... 31
Figura 7: Algoritmo SAStorageYard..... 32
Figura 8: Geração de vizinhança aplicado a esse problema..... 33
Figura 9: Cálculo da FO aplicado a esse problema..... 33
Figura 10: Exemplo da área de influência com cálculo de distância vetorial..... 34
Figura 11: Pseudocódigo roadsAprovTrecho..... 35
Figura 12: Pseudocódigo processarEstradasDeAcesso..... 36
Figura 13: Pseudocódigo conectarPatios..... 36
Figura 14: Hierarquia definição de trilhas..... 37
Figura 15: Pseudocódigo do algoritmo definição de trilhas..... 38
Figura 16: Área de estudo..... 40
Figura 17: Comparação solução pátios com o de Aguiar et al. (2020)..... 42
Figura 18: Arquivos gerados..... 46
Figura 19: Solução gerada para menor instância..... 46

Lista de tabelas

Tabela 1: Parâmetros SA.....41
Tabela 2: Alocação de pátios comparados com de Aguiar et al. (2020)..... 42
Tabela 3: Definição de estradas e trilhas comparada com Aguiar et al. (2023)..... 43

Lista de abreviaturas

ACO	Colônia de Formigas
APP	Áreas de Preservação Permanente
BILP	Binary Integer Linear Programming
COMFLOR	Complemento para a otimização do planejamento do manejo florestal
EIR	Exploração de Impacto Reduzido
Flona	Floresta Nacional
FO	Função objetivo
GRASP	Greedy Randomized Adaptive Search
HSTL	Heuristic Skid Trail Layout
MFS	Manejo florestal sustentável
PAH	Processo Analítico Hierárquico
PCMC	Problema do caminho mais curto
PMFS	Plano de manejo florestal sustentável
PMM	Modelo das p-medianas
PO	Pesquisa operacional
PPMC	Modelo de p-medianas capacitado
SA	Simulated Annealing
SIG	Sistema de Informações Geográficas
SYLH	Shortest Yard Link Heuristic
TS	Busca tabu
UMF	Unidade de Manejo Florestal
UP	Unidade de Produção
UPA	Unidade de Planejamento Anual
UT	Unidade de trabalho
VNS	Variable Neighborhood Search

Sumário

1. INTRODUÇÃO..... 10
1.1. Problema e sua importância.....11
1.2. Objetivos..... 12
1.2.1. Objetivos Gerais..... 13
1.2.2. Objetivos específicos.....13
2. REVISÃO LITERATURA.....14
2.1. Métodos de otimização..... 14
2.2. O problema da alocação de pátios de estocagem.....15
2.2.1. O modelo da p-mediana (PMM)..... 17
2.2.2. O modelo da p-mediana capacitado (PMMC).....19
2.3. Problema do traçado de estradas florestais.....20
2.3.1. O problema do caminho mais curto (PCMC)..... 21
2.4. Problemas NP-difícil.....22
2.5. Heurísticas e meta-heurísticas.....23
2.5.1. Simulated Annealing (SA).....24
2.6. Ferramentas utilizadas.....26
2.6.1. QGIS.....26
2.6.2. Python.....26
2.6.3. Qt Designer..... 26
3. METODOLOGIA..... 28
3.1. Obtenção dos dados..... 30
3.2. Leitura dos dados.....30
3.3. Definição dos pátios de estocagem..... 31
3.3.1. Distâncias entre pátios e árvores..... 34
3.4. Definição das estradas florestais..... 35
3.5. Definição das trilhas de arraste.....37
3.6. Criação dos shapefiles.....38
4. RESULTADOS..... 40
4.1. Instâncias..... 40
4.2. Parâmetros para o SA.....41
4.3. Ambiente de teste..... 41
4.4. Resultados computacionais..... 41
4.4.2. Otimização do cálculo da FO.....44
4.4.3. Arquivos gerados..... 45
5. CONCLUSÕES..... 47
6. REFERÊNCIAS BIBLIOGRÁFICAS.....49

7. APÊNDICES.....53
Apêndice A - SAStorageYard.....53
Apêndice B - gerarVizinhoPatio.....56
Apêndice C - CalculaFOPatio.....57
Apêndice D - RoadsAprovTrech.....59
Apêndice E - TrailsAprovTrecho.....65

1. INTRODUÇÃO

No Brasil a floresta representa uma fonte de renda por meio dos produtos provenientes dela (madeira, óleo, látex, entre outros.) (ESPADA et al., 2013). No entanto é notável a devastação descontrolada por diversos fatores sendo um deles o manejo florestal mal executado.

Neste aspecto, o manejo florestal sustentável (MFS) desempenha um papel importante, direcionando empresas do ramo na adoção de práticas de corte seletivo e exploração de impacto reduzido. Contudo, em florestas nativas a extração de toras envolve um complexo processo de tomada de decisão com inúmeras variáveis, como a tipologia florestal, relevo, hidrografia e solo, tornando assim complexo o planejamento da exploração, sobretudo, com a necessidade da observação das restrições impostas pelo regime do MFS (FIGUEIREDO; BRAZ; D'OLIVEIRA, 2007; AGUIAR et al., 2020).

Dentro dos critérios do MFS, as práticas que objetivam diminuir a abertura de áreas estão entre as mais cruciais. Amaral et al. (1998) expõe que sem essas práticas, a exploração resulta em florestas com grandes clareiras e numerosas árvores danificadas facilitando a entrada e a propagação do fogo, aumento do crescimento de espécies sem valor comercial e danificando a regeneração de espécies madeireiras.

A construção das infraestruturas essenciais para o manejo como pátios de estocagem, estradas e trilhas de arraste, estão entre as atividades mais dispendiosas e que causam mais abertura florestal. Além disso, comumente o profissional que está planejando a exploração conta principalmente com a experiência, gastando mais tempo alocando as infraestruturas do que verificando a viabilidade do projeto (AGUIAR et al., 2020).

Com isso, nos planos de manejo florestal, geralmente é possível notar a presença de estradas e pátios pouco eficientes (FIGUEIREDO; BRAZ; D'OLIVEIRA, 2007). Neste aspecto, a otimização da infraestrutura é crucial para reduzir a área aberta, o que resulta na diminuição de custos e impactos ambientais. Especificamente, otimizar os pátios de estocagem exerce uma influência significativa na quantidade de área

aberta, pois pode contribuir para a redução da quantidade de estradas e trilhas de arraste construídas (AGUIAR et al., 2020; BRAZ et al., 1998).

Dada a importância de reduzir a abertura florestal, a otimização do planejamento de infraestruturas tem sido objeto de muitos estudos, como em Silva et al. (2018), Arima (2008) e Søvde et al. (2013).

Dentre os estudos mais recentes, pode-se citar o estudo de Aguiar et al. (2023) que utilizou métodos heurísticos e exatos para alocar de forma combinando as três infraestruturas necessárias à exploração, pátios, estradas e trilhas. Dentre os achados obtidos no estudo, os autores chegaram a uma redução aproximada de 6,5% no custo e de 4% em área devastada para a alocação de pátios com o SA associado ao problema p-mediana e bons resultados para o traçado de estradas e trilhas com o algoritmo D'Esopo-Pape associado ao problema de caminho mínimo.

Este recente estudo traz uma importante contribuição para o problema da alocação de infraestruturas necessárias ao MFS. Contudo, o estudo se limita a propor os modelos e algoritmos associados ao problema, não disponibilizando o software como produto, que permita uma aplicação prática dessa otimização no planejamento de uma empresa do ramo.

Neste sentido, o presente trabalho visa reproduzir o estudo de Aguiar et al. (2023) e desenvolver o software COMPFLOR (Complemento para a otimização do planejamento do manejo florestal), capaz de executar o planejamento otimizado conforme proposto e que seja viável de implantação e uso em uma empresa do ramo de manejo florestal sustentável. Espera-se que o software proposto possa ser utilizado como suporte na tomada de decisões do planejamento do MFS e assim, contribua para reduzir os custos e os danos à floresta remanescente.

1.1. Problema e sua importância

O manejo florestal sustentável (MFS) no Brasil é vital para preservar e utilizar de forma consciente os recursos das florestas, porém enfrenta desafios consideráveis na alocação estratégica de pátios de estocagem, estradas e trilhas de arraste. Os pátios demandam uma alocação otimizada possibilitando facilitar o arraste e reduzir custos desnecessários influenciando na alocação das outras infraestruturas

(AMARAL et al., 1998; BRAZ et al., 1998; AGUIAR et al., 2020). Com isso a otimização pode reduzir a abertura da floresta e os custos monetários. Embora métodos matemáticos exatos consigam solucionar o Problema de p-medianas, sua aplicação em escalas maiores é ineficiente devido ao crescimento exponencial do tempo necessário para encontrar a solução ótima dentre as inúmeras soluções possíveis para o problema. Nesse sentido, meta-heurísticas, conforme sugerido por Aguiar et al. (2020), emergem como alternativa viável. Não tendo garantia de otimalidade, mas é capaz de obter soluções próximas à ótima em tempo hábil.

No âmbito das estradas, o traçado otimizado é essencial para minimizar custos de transporte e construção, os quais geralmente são maiores que os das outras infraestruturas de exploração (FFT, 2002). Características das florestas nativas como a heterogeneidade das espécies, distribuição espacial das árvores, volume, APP e árvores protegidas por lei levam esse modelo de planejamento a perder eficiência em relação ao planejamento e aumentos dos custos e dos danos causados a floresta remanescente (AGUIAR et al., 2021).

No contexto das trilhas de arraste, Aguiar et al. (2023) descreve uma heurística dividida em três etapas para minimizar custos e impactos ambientais. A metodologia busca identificar áreas de influência dos pátios, definir pontos de alta densidade entre árvores de interesse e, por fim, traçar o caminho mínimo entre esses pontos e os pátios de estocagem.

Essas infraestruturas, essenciais para a exploração florestal, demandam soluções práticas e eficientes para seu planejamento estratégico. É fundamental aprimorar métodos de otimização que considerem a complexidade ambiental e as restrições legais para promover um manejo florestal mais sustentável e economicamente viável.

1.2. Objetivos

Nesta seção são apresentados o objetivo geral, bem como os específicos deste trabalho.

1.2.1. Objetivos Gerais

Este trabalho tem como finalidade implementar o método de otimização proposto por Aguiar et al. (2023) por meio do software COMPFLOR, possibilitando planejar a alocação das infraestruturas necessárias à exploração em florestas nativas. Para isso, combina informações técnicas da área a ser explorada, provenientes de dados de campo e métodos computacionais, visando à otimização do uso dos recursos, à redução dos danos à floresta e à diminuição dos custos de operação. do uso dos recursos e a redução nos danos à floresta e nos custos de operação.

1.2.2. Objetivos específicos

Além do objetivo geral, pretende-se atingir os seguintes objetivos específicos:

a) Desenvolver o método para alocação otimizada de pátios de estocagem de madeira proposto por Aguiar et al. (2020), utilizando algoritmos de otimização e considerando variáveis como topografia e acessibilidade.

b) Implementar o sistema para alocação otimizada de estradas desenvolvido por Aguiar et al., (2021), baseado em técnicas de otimização de caminho mínimo, levando em consideração a minimização do impacto ambiental e a eficiência logística.

c) Implementar o algoritmo para alocação otimizada de trilhas de arraste de toras proposto por Aguiar et al. (2023), considerando fatores como distribuição das árvores, tipo de solo e redução de danos durante a extração.

d) Validar os resultados iniciais encontrados.

e) Corrigir e ajustar, se necessário, os componentes desenvolvidos.

f) Realizar testes finais com instâncias disponíveis na literatura.

g) Analisar os resultados obtidos comparando-os com resultados disponíveis na literatura.

2. REVISÃO LITERATURA

Nesta seção são apresentados os fundamentos teóricos que embasam o desenvolvimento da solução proposta.

2.1. Métodos de otimização

A otimização, uma das principais ferramentas da pesquisa operacional (PO) tem como objetivo de gerar a melhor solução para problemas de otimização que não podem ser resolvidos de forma eficiente com a computação, se tornando a principal forma de resolução dos problemas NP-Difíceis, ou seja, não se conhece um algoritmo determinístico que consiga resolver eficientemente esse tipo de problema. Nesse aspecto a PO pode ser usada para encontrar a melhor solução em termos de algum critério de desempenho (GONZALEZ, 2007; ANTONIOU E LU, 2007).

A área de gerenciamento de recursos naturais reconhece a implementação de PO como um recurso valioso por causa de cinco fatores: um ambiente complexo, um ou mais objetivos específicos, dúvida sobre o melhor curso de ação, decisões limitadas por recursos limitados e a possibilidade de quantificar problemas (NIEUWENHUIS, 1989).

Na década de 1980 e 1990 foram introduzidos os algoritmos de aproximação e meta-heurísticas desenvolvidos para fornecer soluções próximas a ótimas para os problemas NP-difíceis em tempo hábil como *Simulated Annealing* (SA), a Colônia de Formigas (ACO), a Busca Tabu (TS) entre outros (GONZALEZ, 2007). Essas técnicas têm sido amplamente aplicadas em diversos contextos, incluindo o planejamento de recursos naturais e o manejo florestal.

Por exemplo, Silva et al. (2018) busca otimizar a localização de pátios de estocagem na floresta amazônica, região do Brasil, considerando Áreas de Preservação Permanente (APP), zonas restritas e árvores remanescentes. O estudo examina essa distribuição utilizando ferramentas de SIG e técnicas de programação matemática baseadas no modelo clássico de p-medianas. Os autores concluíram que a aplicação do *Processo Analítico Hierárquico* (PAH) resultou em ganhos significativos na eficiência da colheita, com potencial para reduzir o número de estradas florestais e trilhas de arraste.

O estudo de Arima (2008) expande as descobertas apresentadas por Arima (2005) ao discutir a construção de estradas por empresas na Amazônia. Eles empregam métodos geoestatísticos em conjunto com Sistema de Informações Geográficas (SIG) para replicar um padrão comum de fragmentação encontrado em florestas tropicais, conhecido como dendrítico, um dos três tipos predominantes na bacia amazônica. Os autores ressaltam a importância de incorporar informações sobre o tipo de solo, vegetação e cursos d'água como parâmetros cruciais para futuros esforços de modelagem.

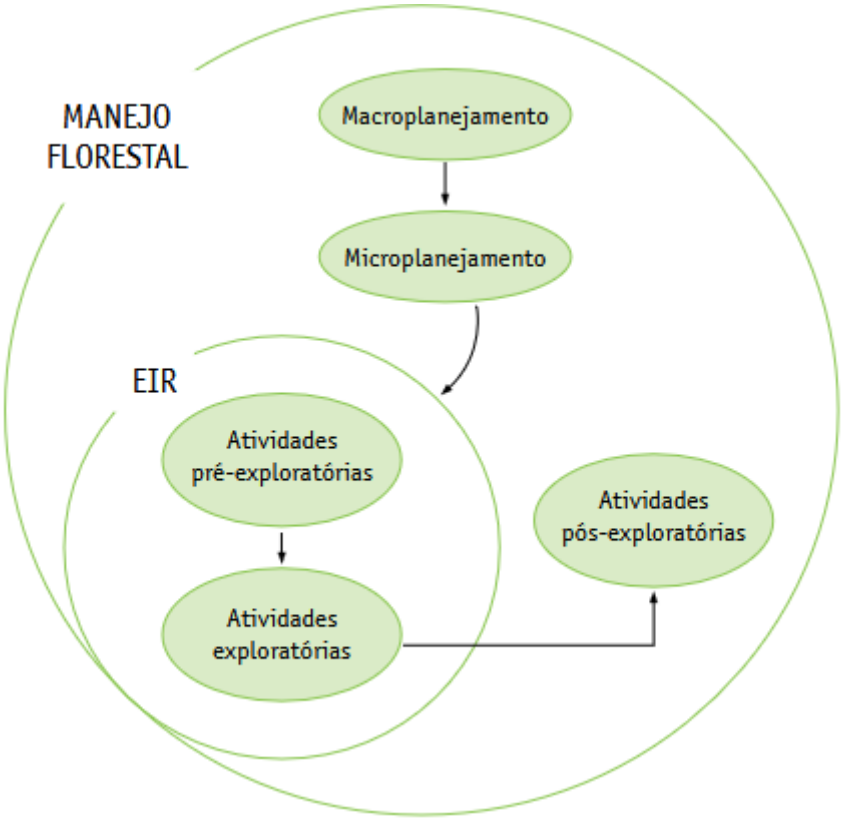
Søvde et al. (2013) avaliou a otimização do *layout* das trilhas de arraste levando em conta a topografia do terreno. Eles empregaram a meta-heurística *Greedy Randomized Adaptive Search* (GRASP). Ao comparar a solução proposta pelo GRASP com um método ganancioso, os autores concluíram que o GRASP pode ser uma ferramenta valiosa para planejar rotas viáveis no terreno, resultando em uma melhora de 5,6% no lucro líquido (FO) e redução de 30% no custo em comparação com a heurística gananciosa na instância mais irregular.

2.2. O problema da alocação de pátios de estocagem

O plano de manejo florestal sustentável (PMFS) baseia-se nas quatro etapas do manejo florestal (Figura 1): Macroplanejamento que é composto por exemplo das atividades de seleção das áreas aptas ao manejo florestal, quantificação do potencial da floresta para manejo florestal, avaliação da viabilidade econômica do empreendimento e definição e dimensionamento das infraestruturas gerais; Microplanejamento que abrange atividades pré-exploratórias, incluindo a delimitação da unidade de trabalho (UT), o levantamento detalhado do inventário e o planejamento das infraestruturas florestais, como pátios e estradas; Atividades exploratórias envolvem corte direcional das árvores, abertura do caminho de fuga para garantir a segurança da equipe a derrubada controlada das árvores por meio de técnicas específicas, visando minimizar o desperdício de madeira e direcionar a queda de forma segura. Esse processo busca evitar acidentes de trabalho, bem como reduzir danos às árvores remanescentes e às porta-sementes. Após essa etapa, realiza-se o planejamento do arraste, seguido do transporte das toras até o pátio de estocagem, empregando técnicas que minimizam impactos ao solo e

preservam a vegetação remanescente; Atividades pós-exploratórias incluem a manutenção das infraestruturas, a avaliação de danos e desperdícios resultantes da exploração, a realização de inventários contínuos, o tratamento silvicultural e a implementação de medidas de proteção floresta (BALIEIRO et al., 2010).

Figura 1: Representação esquemática das etapas do manejo florestal



Fonte: BALIEIRO et al. (2010).

Com isso, o planejamento da alocação de pátios de estocagem ocorre na etapa das atividades pré-exploratórias e a sua abertura ocorre na atividade exploratória. Vale salientar que as atividades pré-exploratórias juntamente com as atividades exploratórias são a base para a Exploração de Impacto Reduzido (EIR). Posteriormente, é feito o arraste de toras para o pátio. Dessa forma, é imprescindível para a EIR que os pátios sejam alocados eficientemente na etapa de planejamento visando diminuir a abertura de pátios, de estradas, de ramais e uma posição ótima em relação às árvores selecionadas, o que acarreta um menor custo monetário e dano ambiental desnecessário na exploração. Cabe salientar que a extração de madeiras a mais de 342,20m é economicamente inviável e leva a danos adicionais à fauna e flora da floresta (SILVA et al., 2018).

Como regra geral, os pátios de estocagem têm a dimensão de 25m x 35m, com faixas de circulação laterais para entrada e saída do Skidder. O tamanho deve permitir o trabalho sistemático das atividades de carregamento, traçamento e movimentação do trator de arraste (BRAZ et al., 1998). Além disso, de acordo com (SILVA et al., 2018), a distância ótima entre os pátios e as árvores escolhidas seria a de no máximo 258 metros.

2.2.1. O modelo da p-mediana (PMM)

O modelo das p-medianas (PMM) visa escolher p-medianas ou pontos de facilidades de modo a minimizar a soma ponderada das distâncias dos clientes, centrados em cada vértice, até as instalações mais próximas. Dessa forma, esse é o modelo matemático mais popular associado ao problema da localização de facilidades (PIZZALOTO et al., 2012).

O modelo de p-medianas busca otimizar a localização de instalações em um conjunto finito de pontos de demanda $N = \{1,..., n\}$. Nesse contexto, cada elemento $i \in N$ representa um cliente ou vértice, enquanto cada $j \in N$ corresponde a uma instalação potencial ou mediana. O parâmetro p define a quantidade de instalações que devem ser estabelecidas para atender à demanda de forma eficiente. Cada cliente i possui um peso associado w_i, refletindo sua importância na alocação. As distâncias entre clientes e instalações são representados pela matriz $[d_{ij}]_{n\,x\,n}$, onde os valores $d_{ii} = 0$ garantem que a distância de um cliente para si mesmo seja nula. A distribuição dos clientes entre as instalações é descrita pela matriz binária $[x_{ij}]_{n\,x\,n}$, onde $x_{ij} = 1$ indica que o cliente i está alocado à instalação j, e $x_{ij} = 0$ caso contrário. Além disso, $x_{jj} = 1$ significa que j foi selecionado como uma mediana, quanto $x_{jj} = 0$ indica que a instalação não foi escolhida (PIZZALOTO et al., 2012).

$$Min \qquad Z = \sum_{i \in N} \sum_{j \in N} w_i d_{ij} x_{ij} \tag{1}$$

$$Sujeto\ a: \qquad \sum_{j \in N} x_{ij} = 1 \qquad i \in N \tag{2}$$

$$\sum_{j \in N} x_{jj} = p \tag{3}$$

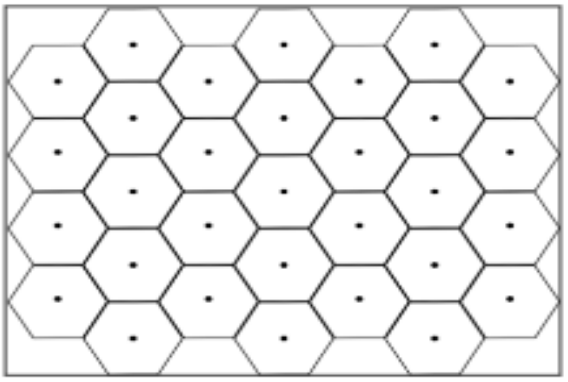
$$x_{ij} \leq x_{jj} \qquad i, j \in N \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad i, j \in N \tag{5}$$

No modelo acima a função objetivo (1) indica a minimização das distâncias ponderadas entre os clientes e os postos que oferecem serviços; as restrições em (2) indicam que cada cliente *i* é alocado a somente uma instalação *j*; a restrição (3) garante que somente *p* instalações oferecem o serviço proposto; as restrições em (4) afirmam que um cliente somente é atendido num local onde existe uma instalação que oferece o serviço, e as restrições em (5) impõem variáveis de decisão binárias.

A Figura 2 exemplifica uma solução usando o modelo da p-mediana em um problema real de alocação de sirenes em estados norte-americanos para alertar os moradores sobre os tornados, visando a disposição ótima das sirenes para cobrir a área urbana, dado o alcance de cada sirene.

Figura 2: Abrangência das Sirenes em Dublin, Illinois, EUA.



Fonte: Pizzolato et al. (2012).

Da mesma forma, a distribuição de pátios de estocagem está relacionada a um problema de alocação de facilidades envolvendo os pátios como vértices centrais e as árvores da instância como clientes a serem associados, objetivando a minimização da distância, ou seja, a minimização da distância de arraste da árvore até o pátio.

2.2.2. O modelo da p-mediana capacitado (PMMC)

No modelo de p-medianas capacitado (PPMC), uma variação do modelo de medianas, existe uma capacidade máxima de atendimento de cada instalação, por isso o critério de otimização passa a ser alocar as instalações de modo a minimizar a distância com os clientes e cada instalação deve ser alocada sem violar a restrição de capacidade (PIZZALOTO et al., 2012). Portanto, a aplicação do PPMC se faz necessária para resolver o problema de alocação de pátios de estocagem, especialmente considerando as restrições de capacidade máxima de armazenamento suportadas por esses pátios.

$$Min \qquad Z = \sum_{i \in N} \sum_{j \in N} w_i d_{ij} x_{ij} \tag{6}$$

$$Sujeto\ a: \qquad \sum_{j \in N} x_{ij} = 1 \qquad i \in N \tag{7}$$

$$\sum_{j \in N} x_{jj} = p \tag{8}$$

$$\sum_{j \in N} w_i x_{ij} \leq Q_i x_{jj} \qquad j \in N \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad i, j \in N. \tag{10}$$

A equação do PMMC supõe exclusividade entre cada centróide e cada facilidade, pois as variáveis x são binárias. A formatação do PMMC difere do PMM no que se refere à restrição que associa o cliente *i* à instalação *j*, ao substituir a restrição (4)

pela (9) considera-se a capacidade de atendimento Q da facilidade x (PIZZALOTO et al., 2012).

2.3. Problema do traçado de estradas florestais

O planejamento das estradas florestais ocorre na etapa pré-exploratória do PMFS, sendo geralmente as infraestruturas mais onerosas da exploração e são as responsáveis por dar acesso a área a ser explorada sendo essenciais para o manejo florestal devido ao uso em quase todas as etapas do manejo e utilização nas atividades pós-exploratórias, além de serem usadas para outras atividades como combate a incêndios. Os custos de construção e manutenção das estradas florestais são usualmente os maiores influenciadores no custo de produção da madeira (AKAY, 2006).

Akay (2006) considera, em seu estudo, algumas das principais especificações geométricas e exigências ambientais na construção de estradas florestais. Entre elas, destacam-se a inclinação máxima da estrada, a inclinação mínima para drenagem e a travessia mínima de riachos.

Além disso, Aguiar (2019) aponta problemas adicionais na construção de estradas em florestas tropicais. Fatores como o período prolongado de chuva intensa, a presença de brejos, áreas de preservação permanente e árvores protegidas por lei representam desafios significativos. Outro aspecto relevante é a colheita em sistema policíclico, comumente adotado na EIR, que pode resultar em baixa receita e, consequentemente, limitar os investimentos em estradas.

Em função do alto custo e as particularidades das estradas florestais, é essencial que sejam considerados estes aspectos para minimizar a abertura de estradas para acessar áreas com baixo potencial madeireiro, estradas planejadas em condições de relevo desfavorável e excessivo número de pontes (FIGUEIREDO; BRAZ; D'OLIVEIRA, 2007). Figueiredo, Braz e D'Oliveira (2007) ainda apontam que os danos desnecessários à floresta podem levar à redução do estoque de madeira comercial para o próximo ciclo. Existem três tipos de estradas florestais: as primárias, que são usadas para transportar a madeira até as instalações industriais;

as secundárias, que conectam os pátios de estocagem; e as trilhas de arraste, que permitem o acesso às árvores.

2.3.1. O problema do caminho mais curto (PCMC)

Como no caso do problema de localização no plano euclidiano, o problema do caminho mais curto (PCMC), também denominado como problema do caminho mínimo, é um clássico da literatura de otimização. Adicionalmente, o problema florestal de definir o traçado de estradas, seja para reduzir a distância ou o custo, pode ser tratado como um problema de caminho mais curto.

O problema do caminho mínimo pode ser formulado matematicamente usando grafos $G = (V, E)$, em que V representa o conjunto de vértices do grafo e E representa o conjunto de arestas. O caminho mais curto entre dois vértices *u* e *v* pertencentes ao grafo, é uma sequência de arestas que, passando por vértices distintos, conecta *u* a *v* acumulando a menor distância possível.

Dessa forma, para que exista um caminho mais curto entre u e v, é essencial que haja uma conexão entre esses vértices (Goldbarg; Luna, 2005).

Este problema está intimamente relacionado a diversos problemas combinatórios, como os de roteamento, programação e sequenciamento de tarefas, dentre outros (Goldbarg; Luna, 2005). No âmbito da programação matemática, existem os algoritmos com especialização do simplex e os algoritmos de fluxo.

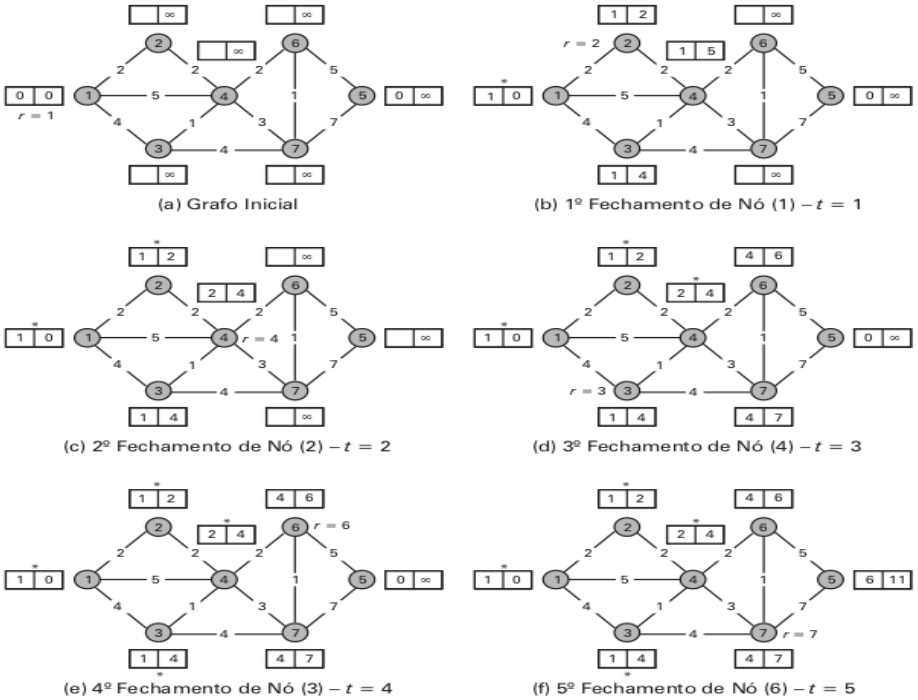
No entanto, atualmente, os algoritmos mais eficientes estão disponíveis por meio da abordagem em grafos (Goldbarg; Luna, 2005), como o algoritmo de Dijkstra, proposto em 1959, com complexidade $O(n^2)$, sendo um dos mais conhecidos para a solução do PCMC.

Goldbarg e Luna (2005) exemplificam em seu artigo a solução de um PCMC utilizando o algoritmo de Dijkstra (Figura 3), considerando um grafo com vários vértices: 1, 2, 3, 4, 5, 6, 7, onde cada aresta representa o custo de percurso entre os nós de origem e destino. O rótulo próximo a cada vértice contém, na primeira posição, a origem da rotulação e, na segunda, o valor acumulado até o vértice

correspondente. Inicialmente, todos os nós, exceto o nó inicial, são marcados com distância acumulada infinita.

A cada iteração, a distância acumulada é atualizada sempre que se encontra um caminho de menor custo. O contador de iterações *t* é incrementado, e o nó com a menor distância acumulada é fechado. O objetivo do algoritmo é calcular as menores distâncias do vértice inicial para todos os outros.

Figura 3: Algoritmo Dijkstra



Fonte: Goldbarg e Luna (2005)

2.4. Problemas NP-difícil

Na teoria da computação, a classificação dos problemas em classes de complexidade é fundamental para compreender os limites da computação. Essa classificação é baseada na dificuldade intrínseca de resolvê-los e inclui categorias como P, NP, NP-Completo e NP-Difícil.

Os problemas da classe P podem ser resolvidos de forma eficiente, ou seja, por um algoritmo determinístico em tempo polinomial. Já os problemas da classe NP não possuem, até onde se sabe, algoritmos que os resolvam em tempo polinomial. No

entanto, suas soluções podem ser verificadas eficientemente por um algoritmo determinístico.

Os problemas NP-Completos pertencem à classe NP e são, ao mesmo tempo, os mais difíceis dessa classe. Isso significa que, se houver um algoritmo em tempo polinomial para qualquer problema NP-Completo, todos os problemas em NP também poderão ser resolvidos eficientemente.

Por fim, os problemas NP-Difíceis são, no mínimo, tão complexos quanto os NP-Completos, mas não necessariamente pertencem à classe NP, ou seja, podem não admitir uma verificação eficiente de suas soluções.

2.5. Heurísticas e meta-heurísticas

Para problemas complexos da PO, os métodos heurísticos e meta-heurísticos surgem como uma alternativa mais eficiente que os métodos exatos para resolução de problemas, principalmente os problemas de grande complexidade, em tempo computacionalmente viável. É importante destacar que uma heurística é uma técnica que procura alcançar uma solução aceitável, empregando um esforço computacional considerado razoável (GOLDBARG; LUNA, 2005). Porém, não há garantia de que essa solução seja um ótimo global ou um ótimo local próximo do ótimo global.

Nos primórdios, as heurísticas foram desenvolvidas para solução de problemas específicos sendo as heurísticas clássicas de roteamento um ótimo exemplo disso, ou seja, não eram pensadas para ser uma estratégia universal de solução (GOLDBARG; LUNA, 2005). Entretanto, logo surgiram as chamadas meta-heurísticas que envolvem as heurísticas e possibilitam o seu uso em variados problemas com pouca ou nenhuma adaptação.

As heurísticas despertaram o interesse de toda a comunidade científica pela qualidade das soluções encontradas e pelo retorno dado ao compromisso do objetivo, tornando-se fortemente usadas em modelos combinatórios. Apesar do amplo uso de heurísticas e meta-heurísticas em uma variedade de problemas, a área florestal ainda tem sido carente dessas abordagens (AGUIAR, 2019).

Aguiar (2019) utiliza diferentes meta-heurísticas, como: *Simulated Annealing* (SA), *Variable Neighborhood Search* (VNS), Busca Tabu (TS) e *Greedy Randomized Adaptive Search* (GRASP). Todas foram avaliadas para o mesmo problema de alocação de pátios na otimização do PMFS e comparadas com o CPLEX, um método exato. As comparações revelaram que o SA foi a meta-heurística que obteve a melhor performance, além de ser a única a fornecer uma solução viável para uma das instâncias testadas.

2.5.1. Simulated Annealing (SA)

Proposto por Kirkpatrick et al. (1983) como uma meta-heurística de busca local para resolver problemas de otimização combinatória. O SA utiliza fenômenos naturais de recozimento de metais, como fonte de inspiração para encontrar soluções de qualidade. Ele é classificado como um dos muitos algoritmos surgidos na computação natural.

O SA é um algoritmo de busca local, o que significa que ele começa com uma solução inicial e em seguida pesquisa no espaço de soluções gerados, fornecendo iterativamente uma nova solução que está “próxima da solução atual” (GOLDBARG; LUNA, 2005). A estratégia de funcionamento pode ser descrita por quatro características: inicia com uma solução sendo ela aleatória ou de conhecimento prévio; geração de vizinhança na qual a partir da solução atual são geradas novas soluções por meio de pequenas alterações visando escapar de ótimos locais; redução de temperatura na qual a cada iteração a temperatura e reduzida que geralmente permite menos aceitação de soluções piores; critério progressivamente elitista para trocar a solução aleatória pela atual.

Figura 4: Pseudocódigo do SA aplicado ao problema de alocação de pátios

```
1  SA(tempoExec, S0, T0, Tc, SAmax, α)
2    horaTérmino ← horaAtual() + tempoExec
3    mS ← S0
4    IterT ← 0
5    T ← T0
6    enquanto T > Tc e horaAtual() < horaTérmino faça
7      enquanto IterT < SAmax e horaAtual() < horaTérmino faça
8          IterT ← IterT + 1
9          obter(vizinho aS ∈ N(S0))
10         Δ ← f(aS) - f(S0)
11         se Δ < 0 então
12             S0 ← aS
13             se f(aS) < f(mS) então
14                 mS ← aS
15             fim se
16         senão
17             random(x ∈ [0,1])
18             se x < e^(-Δ/T) então
19                 S0 ← aS
20             fim se
21         fim se
22     fim enquanto
23     T ← α x T
24     se T < (Tc + Tc * 2) e horaAtual() < horaTérmino então
25         T ← T0
26     fim se
27     IterT ← 0
28   fim enquanto
29   retorna mS
```

Fonte: AGUIAR (2019).

Na Figura 4 o algoritmo do *Simulated Annealing* (SA) opera com dois *loops* distintos. O primeiro loop (linha 6) mantém as iterações ativas até que a temperatura atinja um ponto de congelamento, o que é controlado pela taxa de resfriamento. O segundo loop (linha 7) é responsável por realizar iterações que buscam o estado de equilíbrio térmico, representando o fenômeno natural no SA. Para alcançar esse equilíbrio, primeiro, é gerado um vizinho *aS*. Em seguida, é calculada a variação. Se o valor da função objetivo (FO) para a solução vizinha for maior, a variação (Δ) será positiva; se for menor, será o oposto.

Se a variação é negativa, indica uma melhoria em um problema de minimização. A solução aS é aceita; em seguida, verifica-se se ela é melhor que a solução global. Se for o caso, a solução global é atualizada. Por outro lado, se a variação for positiva, indicando uma piora, ainda é possível aceitar essa solução, desde que uma condição específica seja atendida. Assim, a solução corrente será substituída pela solução vizinha, mesmo que ela seja pior. Esse processo permite que o SA escape de ótimos locais, explorando alternativas que inicialmente parecem piores, mas que podem levar a resultados mais vantajosos.

2.6. Ferramentas utilizadas

2.6.1. QGIS

O QGIS, Sistema de Informações Geográficas de código aberto, representa uma ferramenta fundamental para o COMPFLOR. Reconhecido por sua versatilidade e ampla gama de funcionalidades, o QGIS proporciona um ambiente poderoso para manipular, visualizar, analisar e interpretar dados geoespaciais. Sua *interface* intuitiva aliada à extensa variedade de complementos e recursos tornam-no uma escolha ideal para explorar e processar informações geográficas. Além disso, sua capacidade de lidar com diferentes formatos de dados e a comunidade ativa de desenvolvedores garantem flexibilidade e atualizações contínuas, contribuindo significativamente para a eficiência e precisão das análises realizadas neste trabalho.

2.6.2. Python

Para desenvolver componentes no QGIS, geralmente é utilizada a linguagem de programação Python. O Python é a linguagem predominante para criar plugins, ferramentas e complementos personalizados dentro do ambiente do QGIS. Ele oferece uma maneira flexível e poderosa de estender as funcionalidades do QGIS, permitindo aos usuários desenvolverem suas próprias ferramentas, realizar automações, processar dados geoespaciais e criar interfaces personalizadas para interação com os usuários dentro do software.

2.6.3. Qt Designer

O Qt Designer desempenha um papel crucial no desenvolvimento de interfaces de usuário dentro do ambiente do QGIS. Esta ferramenta facilita a criação de interfaces gráficas intuitivas, permitindo a construção de janelas interativas para entrada e visualização de dados no contexto geoespacial. Com sua interface amigável e recursos de arrastar e soltar, o QT Designer viabiliza a criação eficiente de interfaces personalizadas, simplificando a interação entre o usuário e as funcionalidades implementadas no ambiente do QGIS. Sua integração facilitada com o ambiente de

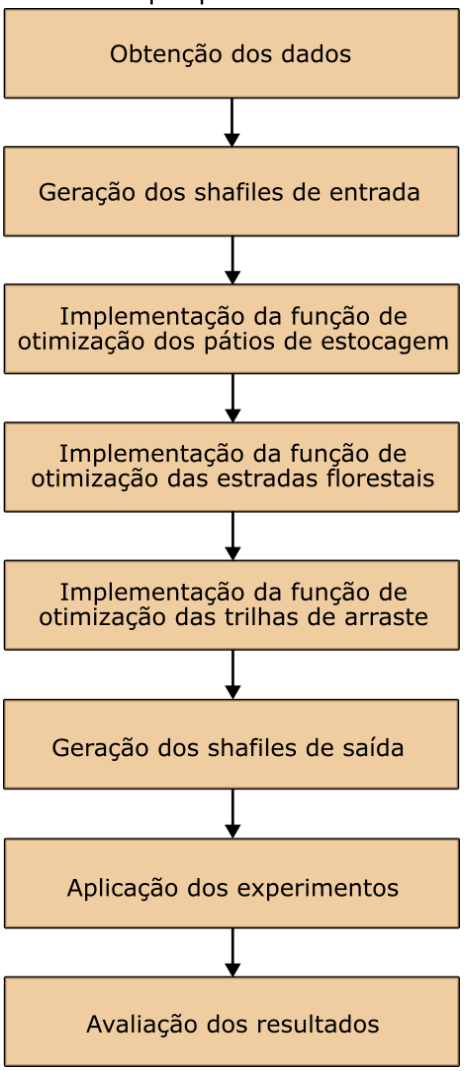
programação Python e o QGIS torna o QT Designer uma ferramenta valiosa para o desenvolvimento de soluções geoespaciais com interfaces intuitivas e funcionais.

3. METODOLOGIA

Este estudo foi desenvolvido com o objetivo de criar um software amplamente acessível ao público, capaz de auxiliar no plano de MFS. Para isso, foi adotado o QGIS como ferramenta de geoprocessamento, devido a sua ampla adoção por empresas do ramo de manejo florestal. Dessa forma, o método de otimização proposto em Aguiar et al. (2023) foi implementado como complemento no QGIS por meio do COMPFLOR.

Após a implementação, foram também realizados experimentos para comparação com base nos estudos de Aguiar et al. (2020), Aguiar et al. (2021) e Aguiar et al. (2023), foram estabelecidas as etapas para a realização desta pesquisa (Figura 5).

Figura 5: Fluxograma das etapas para desenvolvimento do COMPFLOR.



Fonte: Do autor

Seguindo o fluxo metodológico estruturado, primeiramente realizou-se a obtenção dos dados, que foram extraídos de três instâncias distintas, abrangendo informações geoespaciais como vértices, pontos de pátios, árvores exploráveis, Áreas de Preservação Permanente (APP), inclinação do terreno e zonas de inundação. Esses dados foram organizados e convertidos em shapefiles de entrada, garantindo a estrutura espacial necessária para as etapas subsequentes do processamento.

Com os dados estruturados, passou-se à definição dos pátios de estocagem, etapa fundamental para a logística da exploração. Para isso, foi implementado e aplicado o algoritmo *Simulated Annealing* (SA), que otimiza a localização dos pátios, minimizando a distância entre eles e as árvores exploráveis dentro da sua área de influência.

Em seguida, definiram-se as estradas florestais, que conectam os pátios entre si e aos pontos de entrada da Unidade de Produção (UP). A heurística *Shortest Yard Link Heuristic* (SYLH) foi empregada para reduzir a necessidade de abertura de novas vias, enquanto o algoritmo de Dijkstra foi utilizado para determinar as rotas mais curtas, garantindo a eficiência do sistema viário.

Após a definição das estradas, passou-se à determinação das trilhas de arraste, responsáveis pelo deslocamento da madeira até os pátios. A abordagem adotada considerou a área de influência dos pátios e aplicou novamente o *Simulated Annealing* para selecionar pontos de facilidade, conectando-os ao pátio por meio do menor caminho obtido pelo algoritmo de *Dijkstra*. Esse processo garantiu a minimização dos custos operacionais e do impacto ambiental. Concluída essa etapa, os resultados foram exportados em *shapefiles* de saída, permitindo sua visualização e análise em softwares de geoprocessamento como o QGIS.

Por fim, foram realizados experimentos para avaliar a eficiência do modelo, considerando as três instâncias e critérios como a distância total percorrida e os impactos ambientais. As simulações foram executadas sob diferentes configurações para verificar a robustez da solução proposta.

Os resultados foram analisados e comparados com a literatura, utilizando indicadores como a porcentagem de desvio em relação às melhores soluções conhecidas, o tempo de execução da melhor solução, o tempo médio de

processamento e o comprimento total das rotas geradas. Esse fluxo metodológico permitiu validar a eficácia do modelo e possibilitou o desenvolvimento de um software eficiente para o planejamento do MFS, contribuindo para a otimização da exploração florestal de forma sustentável.

3.1. Obtenção dos dados

Para avaliar o desempenho do COMPFLOR foram utilizadas três instâncias distintas, a fim de abranger uma gama maior de cenários. Assim, os arquivos (vértices, pontos de pátios, árvores exploráveis, APP, inclinação e inundação) das instâncias utilizadas neste estudo foram adquiridos de Aguiar et al. (2023) como arquivos de texto (TXT) e convertidos em *Shapefiles*.

3.2. Leitura dos dados

Neste trabalho, a leitura dos dados geoespaciais é realizada a partir de arquivos Shape (*shapefiles*), amplamente utilizados por profissionais da área florestal para representar informações espaciais, como pontos, linhas e polígonos.

Foi desenvolvida uma função para leitura de vértices (Figura 6), tendo como objetivo extrair os dados dos vértices de um arquivo *Shape* e armazená-los em uma estrutura de dados apropriada para facilitar o acesso e manipulação. A cada vértice lido, os atributos id, x, y e z são extraídos e utilizados para instanciar objetos da classe Area, representando um vértice no espaço no contexto do problema. A criação dessa função foi fundamental para padronizar e otimizar o processamento dos dados geoespaciais, permitindo que a estrutura gerada fosse diretamente utilizada em cálculos e algoritmos de otimização aplicados nas etapas subsequentes.

Figura 6: Função de leitura de vértices

```
def lerInstanciaVertices(self, vertices) -> list[Area]:
    vetor: list[Area] = []

    for verticeLayer in vertices.getFeatures():
        vertice = Area()

        vertice.id = int(verticeLayer.attribute("id"))
        vertice.x = float(verticeLayer.attribute("x"))
        vertice.y = float(verticeLayer.attribute("y"))
        vertice.z = float(verticeLayer.attribute("z"))

        vetor.append(vertice)

    return vetor
```

Fonte: Do autor

A fim de garantir um acesso eficiente aos dados, foi desenvolvido um vetor de instâncias da classe Area. Essa estrutura facilita a busca do pátio por seu ID, permitindo o acesso direto ao vetor por meio de seu identificador. Contudo, para que esse método seja eficiente, é necessário que os dados estejam ordenados pelo ID de cada vértice.

3.3. Definição dos pátios de estocagem

Para a fase de definição dos pátios de estocagem, foi desenvolvido o algoritmo SAStorageYard, cujo pseudocódigo é apresentado na Figura 7. O código-fonte completo pode ser encontrado no Apêndice A. Esse algoritmo utiliza a metaheurística SA com as funções auxiliares gerarVizinho (pseudocódigo na Figura 8, código no Apêndice B) e calculaFOPatios (pseudocódigo na Figura 9, código no Apêndice C). A escolha do SA deve-se ao seu desempenho superior na otimização do problema em questão, superando outras abordagens, como *Tabu Search* (TS), *Variable Neighborhood Search* (VNS) e *Greedy Randomized Adaptive Search Procedure* (GRASP), conforme concluído por Aguiar et al. (2021).

Figura 7: Algoritmo SAStorageYard

```
SAStorageYard(tempoExec, floresta, distancias, S0, restVolSup, T0, Tc, SAmax, α)
1.  horaTérmino ← horaAtual() + tempoExec
2.  mS ← S0
3.  atualS ← S0
4.  IterT ← 0
5.  T ← T0
6.  patios ← vetorZeros(NUM_VERTICES_PATIOS)
7.  iterT ← 0
8.
9.  para k de 1 até NUM_PATIOS faça
10.   patio ← S0.patios[k]
11.   patios[patio - 1] ← 1
12. fim para
13.
14. enquanto T > Tc e horaAtual() < horaTérmino faça
15.   enquanto IterT < SAmax e horaAtual() < horaTérmino faça
16.       iterT ← IterT + 1
17.       vizinhoS ← gerarVizinho(solucaoAtual, patios, restVolSup)
18.
19.       patios ← vetorZeros(NUM_VERTICES_PATIOS)
20.
21.       Δ ← vizinho.FO - solucaoAtual.FO
22.       se Δ < 0 então
23.           atualS ←vizinhoS
24.           para i de 1 até NUM_PATIOS faça
25.               patios[vizinho.patios[i] - 1] ← 1
26.           fim para
27.
28.           se vizinho.FO < melhorSol.FO então
29.               melhorSol ← vizinho
30.               melhorSol.tempoSol ← horaAtual() - horaInício
31.           fim se
32.       senão
33.           random(x ∈ [0,1000])
34.           x = x/1000
35.           se x  < e^(-Δ/T) então
36.               S0 ← vizinhoS
37.               para i de 1 até NUM_PATIOS faça
38.                  patios[vizinho.patios[i] - 1] ← 1
39.               fim para
40.           senão
41.               para j de 1 até NUM_PATIOS faça
42.                  patios[atualS.patios[i] -1] ← 1
43.               fim para
44.           fim se
45.       fim se
46.   fim enquanto
47.   T ← α * T
48.   se T < (Tc + Tc * 2) e horaAtual() < horaTérmino então
49.       T ←T0
50.   fim se
51.   IterT ← 0
52. fim enquanto
53. retorna mS
fim
```

Fonte: Adaptado de Aguiar (2019)

Figura 8: Geração de vizinhança aplicado a esse problema

```
gerarVizinho(floresta, distancias, sol, patios, restVolSup)
   solAtual ← sol
   vizinho = 0

   troca = aleatorio(1, NUM_PATIOS) - 1

   enquanto true
      vizinho = aleatorio(1, NUM_VERTICES_PATIOS)
      se patios[vizinho - 1] != 1
         solAtual.patios[troca] = vizinho
         break
      fim se
   fim enquanto

   calculaFOPatio(solAtual, restVolSup)

   return solAtual
fim
```

Fonte: Do autor

Figura 9: Cálculo da FO aplicado a esse problema

```
calculaFOPatios(solAtual, restVolSop)
   solAtual.volumes ← vetorZeros(NUM_PATIOS)
   solAtual.arvores ← vetor(vetor(), NUM_PATIOS)

   para j de 1 até NUM_ARVORES_EXPLORAVEIS faça
      distanciaMenor ← INFINITO
      patio ← 0
      para k de 1 até NUM_PATIOS faça
         distancia ← distancias[j][solAtual.patios[k]-1]
         se distancia < distanciaMenor então
            distanciaMenor ← distancia
            patio ← k
         fim se
      fim para

      se distanciaMenor > DISTANCIA_MAXIMA então
         diferencaDistancia ← diferencaDistancia + distanciaMenor - DISTANCIA_MAXIMA
      fim se
      distanciaTotal ← distanciaTotal + distanciaMenor

      solucaoAtual.arvores[patios].append(Arvore(floresta[j].id, floresta[j].numero))
      solucaoAtual.volumes[patio] ← solucaoAtual.volumes[patio] + floresta[j].volume
   fim para

   para w de 1 até NUM_PATIOS faça
      se res.volumes[w] > restVolSup então
         diferencaVolume ← diferencaVolume + (res.volumes[w] - restVolSup)
      senão
         diferencaVolume ← 0
      fim se
   fim para

   solAtual.distanciaTotal ← distanciaTotal
   distanciaTotal ← distanciaTotal + PENALIZACAO_VOLUME * (diferencaVolume + diferencaDistancia)

   solAtual.FO ← distanciaTotal
   res.viavel ← (diferencaVolume + diferencaDistancia) == 0
fim
```

Fonte: Do autor

Além disso, o SA destacou-se por sua capacidade de encontrar soluções próximas às obtidas pelo método exato *Binary Integer Linear Programming* (BILP) para instâncias pequenas e médias, e, para a maior instância, apresentou desempenho superior.

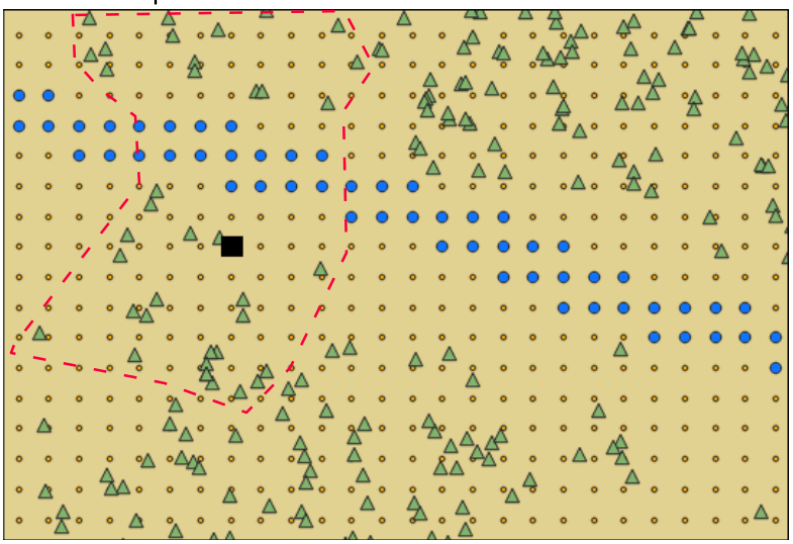
3.3.1. Distâncias entre pátios e árvores

O desenvolvimento do cálculo da Função Objetivo (FO) do pátio é fundamentado na distância entre o pátio e as árvores situadas dentro de sua zona de influência. Inicialmente, foi proposto o cálculo da distância d entre o pátio $P\left(X_1, Y_1, Z_1\right)$ e a árvore $A\left(X_2, Y_2, Z_2\right)$ por meio da fórmula vetorial da distância entre dois pontos:

$$d = \sqrt{\left(X_1 - X_2\right)^2 + \left(Y_1 - Y_2\right)^2 + \left(Z_1 - Z_2\right)^2}$$

No entanto, conforme ilustrado na Figura 10, o resultado da alocação dos pátios apresentava um viés, aproximando-se das áreas de APP para abranger árvores localizadas do outro lado da APP. Para contornar esse problema, optou-se por utilizar o arquivo de distâncias adaptado de Aguiar et al. (2023).

Figura 10: Exemplo da área de influência com cálculo de distância vetorial



Fonte: Do autor

3.4. Definição das estradas florestais

Com os pátios previamente definidos, foi desenvolvida a heurística de conexão dos pátios (pseudocódigo na Figura 11, código no Apêndice D) com base no algoritmo de *Storage yard linkage heuristics* (SYLH) proposta por Aguiar et al. (2023), com o objetivo de minimizar a abertura de novas estradas. O método segue uma abordagem sequencial, conectando inicialmente os pátios mais próximos aos pontos de entrada da Unidade de Produção (UP) (Figura 12). Em seguida, os demais pátios são interligados entre si ou a estradas previamente estabelecidas, buscando reduzir ao máximo a necessidade de novas aberturas (Figura 13). A escolha do Dijkstra deve-se ao seu bom desempenho na otimização do problema em questão e não apresenta perdas significativas em comparação ao D'Esopo-Pape conforme concluído por Aguiar et al. (2021).

Essa estratégia permite um planejamento mais eficiente da infraestrutura viária, garantindo a otimização da conectividade entre os pátios e reduzindo o impacto ambiental associado à construção de novas vias.

Figura 11: Pseudocódigo roadsAprovTrecho

```
roadsAprovTrecho(grafo, solPatios, estradaAcesso)
    NUM_ROADS ← NUM_PATIOS + 1

    patioProibido ← -1
    solRoad ← SolucaoRoad(NUM_VERTICES)
    sol ← SolucaoRoads(NUM_ROADS, NUM_VERTICES)
    vEstSecOrder ← vetorMenosUm(NUM_PATIOS)
    vDistEstradaSec ← vetorINFINITY(NUM_PATIOS)
    vetSelRoads ← vetorZeros(NUM_ROADS)
    vDistPat ← 0
    vDistEstPri ← 0

    preRods ← PreProcecamentoRoads(NUM_PATIOS, patios, solPatios)
    distanciaPatios ← preRods.distanciaPatios
    patiosOrder ← preRods.patiosOrder

    processarEstradasAcesso(grafo, solPatios, estradaAcesso, sol, vetSelRoads)
    conectarPatios(grafo, solPatios, estradaAcesso, sol, vetSelRoads, patiosOrder)

    se estradasAcesso.termino != 0 então
       para i de 1 até NUM_PATIOS faça
          patioAtual ← solPatios.patios[i]
          vPatioAtual ← patios[patioAtual - 1].vertice - 1
          vDistPat[0] ← calculaDistancia(area[vPatioAtual], area[estradasAcesso.termino])
          se vDistPat[0] < menor então
             pMenor ← vPatioAtual
             menor ← vDistPat[0]
          fim se
       fim para

       solRoad ← grafo.Dijkstra(pMenor, estradasAcesso.termino)
       atualizaMetricasSolucao(sol, solRoad)
       vetSelRoads[i] ← 1
    senão
       solRoad ← criarSolucaoVazia(estradasAcesso.termino)
       sol.typeRoad[NUM_ROADS - 1] ← 1
       sol.roads[NUM_ROADS - 1] ← solRoad
       vetSelRoads[NUM_ROADS - 1] ← 1
    fim se

   determinarTipoEstrada(sol, vetSelRoads)

   retorna sol
fim
```

Fonte: Adaptado de Aguiar et al. (2023)

Figura 12: Pseudocódigo processarEstradasDeAcesso

```
processarEstradasDeAcesso(grafo, solPatios, estradaAcesso, sol, vetSelRoads)
1.    para i de 1 até NUM_ACCESS_ROAD faça
2.       vEstradaPri ← estradasAcesso.inicio[i]
3.       pMenor ← -1
4.       patioCon ← -1
5.       menor ← INFINITY
6.
7.       para j de 1 até NUM_PATIOS faça
8.          patioAtual ← solPatios.patios[j]
9.          vPatioAtual ← patios[patioAtual - 1].vertice - 1
10.         vDistPat[0] ← calculaDistancia(area[vEstradaPri], area[vPatioAtual])
11.         se vDistPat[0] < menor então
12.            patioCon ← j
13.            pMenor ← vPatioAtual
14.            menor ← vDistPat[0]
15.         fim se
16.      fim para
17.
18.      solRoad ← grafo.Dijkstra(pMenor, vEstradaPri)
19.      typeRoad ← 1
20.      vetSelRoads[patioCon] ← 1
21.
22.      atualizaVerticesEstSecundarias(solPatios, solRoad, estradasAcesso, vEstSecOrder, vDistEstradaSec, 0, patioProibido)
23.      atualizaMetricasSolucao(sol, solRoad)
24.   fim para
fim
```

Fonte: Do Autor

Figura 13: Pseudocódigo conectarPatios

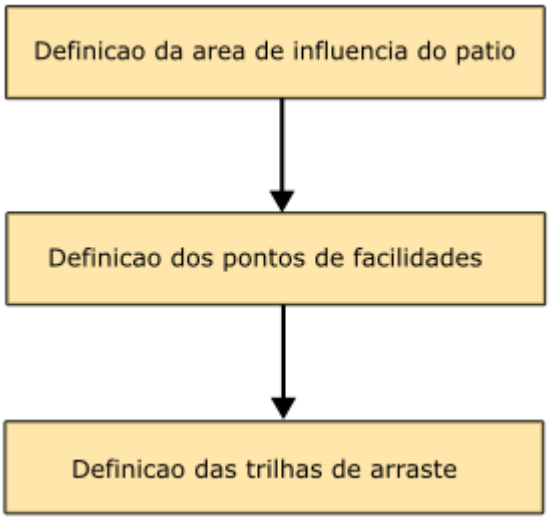
```
conectarPatios(grafo, solPatios, estradaAcesso, sol, vetSelRoads, patiosOrder)
1.    para i de 1 até NUM_PATIOS faça
2.       se vetSelRoads[i] = 0 então
3.          patioProibido ← -1
4.          patioAtual ← solPatios.patios[i]
5.          vPatioAtual ← patios[patioAtual - 1].vertice - 1
6.          vEstradaPri ← obterVerticeInicioProximo(area, estradasAcesso, vPatioAtual, vDistEstPri)
7.          vEstradaSec ← vEstSecOrder[i]
8.          vDistEstSec ← vDistEstradaSec[i]
9.
10.         se vDistEstPri[0] != 0 e vDistEstSec != 0 então
11.            patioCon ← obterPatioProximo(area, patios, vEstradaPri, vDistEstPri[0], i, distanciaPatios, patiosOrder, vDistPat)
12.
13.            se patioCon != -1 então
14.               se vDistEstPri[0] <= vDistEstSec e vDistEstPri[0] <= vDistPat[0] então
15.               solRoad ← grafo.Dijkstra(vPatioAtual, vEstradaPri)
16.               typeRoad ← 1
17.               vetSelRoads[i] ← 1
18.            senão se vDistEstSec <= vDistEstPri[0] e vDistEstSec <= vDistPat[0] então
19.               solRoad ← grafo.Dijkstra(vPatioAtual, vEstradaSec)
20.               typeRoad ← 2
21.               vetSelRoads[i] ← 1
22.            senão
23.               solRoad ← grafo.Dijkstra(vPatioAtual, patios[patioCon - 1].vertice - 1)
24.               typeRoad ← 2
25.               patioProibido ← patioCon
26.            fim se
27.         senão
28.            se vDistEstPri[0] <= vDistEstSec então
29.               solRoad ← grafo.Dijkstra(vPatioAtual, vEstradaPri)
30.               typeRoad ← 1
31.               vetSelRoads[i] ← 1
32.            senão
33.               solRoad ← grafo.Dijkstra(vPatioAtual, vEstradaSec)
34.               typeRoad ← 2
35.               vetSelRoads[i] ← 1
36.            fim se
37.
38.         atualizaVerticesEstSecundarias(solPatios, solRoad, estradasAcesso, vEstSecOrder, vDistEstradaSec, i+1, patioProibido)
39.         atualizaMetricasSolucao(sol, solRoad)
40.      fim se
41.   fim para
fim
```

Fonte: Do Autor

3.5. Definição das trilhas de arraste

Na etapa de definição de trilhas foi desenvolvido o algoritmo Heuristic Skid Trail Layout (HSTL) descrito por Aguiar et al. (2023), que envolve uma sequência de etapas interligadas (Figura 14).

Figura 14: Hierarquia definição de trilhas



Fonte: Do autor

No algoritmo implementado (pseudocódigo na Figura 15, código no Apêndice G) inicialmente define-se a área de influência do pátio, considerando sua localização e as árvores exploráveis adjacentes. A seguir, dentro dessa área de influência, utiliza-se a heurística SA para identificar e selecionar os pontos de facilidade, que representam locais ótimos para a conexão entre os pátios e as árvores. Por fim, o algoritmo de Dijkstra é empregado para calcular o caminho mínimo entre o pátio e os pontos de facilidade previamente definidos, garantindo a eficiência na determinação da rota mais curta e otimizada.

Figura 15: Pseudocódigo do algoritmo definição de trilhas

```
trails(solRoads, solPatios, arvoreSelPatios, qtdArvores, restVolSup, desvios, NUM_ROADS)
1.   solucaoTrilha ← vetorVaziaSolucaoTrilhas()
2.   grPatio ← Grafo()
3.   verticesProibidos ← vetorZeros(NUM_VERTICES)
4.
5.   se app != none entao
6.      marcarTodosAppComoProibidos()
7.   fim se
8.
9.   para i de 1 até NUM_ROADS
10.     marcarVerticesUtilizadosEmEstradasComoProibidos()
11.  fim para
12.
13.  para i de 1 até NUM_PATIOS
14.     solPtTrails ← SolucaoPtTrails()
15.     arvMaisDistante ← 0.0
16.     patioAtual ← solPatios.patios[i]
17.
18.     se qtdArvores[i] / NUM_ARV_TRILHA) >= 1 então
19.        qtdTrilhas ← (qtdArvores[i]/NUM_ARV_TRILHA)
20.     se nao
21.        qtdTrilhas ← 1
22.     fim se
23.
24.     vSubArvPatios ← vetorArvoresExploraveis(qtdArvores[i])
25.     vSubDistPatios ← vetorZeros(qtdArvores[i])
26.
27.     buscaArvoreMaisDistanteDoPatio()
28.
29.     areaPatio ← (arvMaisDistante/30) + 1
30.     qtdVerticesPatio ← pow((areaPatio * 2), 2)
31.     vertPatios ← vetorZeros(qtdVerticesPatio)
32.     qtdRealPatio ← localizaVizinhos(verticesProibidos, patioAtual, arvMaisDistante, var vertPatios)
33.
34.     vertRealPatio ← vetorZero(qtdRealVertPatio)
35.     para j de 1 até qtdRealPatio
36.        vertRealPatio[j] ← vertPatios[j]
37.     fim para
38.
39.     solPtTrilhas ← heuristicaConstrutivaTrilhas(vertRealPatios, qtdRealVertPatio, qtdTrilhas, qtdArvores[i], vSubArvPatios, restVolSup)
40.     solptTrilhas ← SATrilhas(vertRealPatios, qtdRealVertPatio, qtdTrilhas, qtdArvores[i], vSubArvPatios, solPtTrilhas, restVolSup)
41.
42.     qtdRealVertPatio++
43.     vertArestaPatios ← vetorZeros(qtdRealVertPatio)
44.     vertArestaPatios[0] ← patios[patioAtual - 1].vertice
45.
46.     grPatio ← criaGrado(qtdRealVertPatio, GRAU_MAX, PONDERADO)
47.     grPatio ← insereArestas(vertArestaPatios, qtdRealVertPatio)
48.
49.     sol ← trailsAprovTrilhas(grPatio, patioAtual, qtdTrilhas, solPtTrilhas)
50.
51.     transformaReferenciasVerticesParaVerticesReais()
52.
53.     solucaoTrilha.append(sol)
54.
55.  fim para
fim
```

Fonte: Do autor

3.6. Criação dos *shapefiles*

A criação de *shapefiles* é uma etapa crucial para exportar os resultados gerados na resolução do problema de otimização. Esses arquivos, utilizados em Sistemas de Informações Geográficas (SIG), representam dados espaciais como pontos, linhas e polígonos.

No caso deste trabalho, são gerados *shapefiles* para representar os pátios, árvores e as estradas/trilhas conectando-os. Cada ponto é associado a atributos como coordenadas e identificador único, enquanto as estradas e trilhas são representadas por linhas conectando múltiplos pontos.

Os *shapefiles* são salvos em diretórios específicos e seguem o formato ESRI Shapefile, amplamente compatível com ferramentas SIG. Esse processo permite a visualização dos resultados em plataformas como QGIS, facilitando a análise espacial dos dados.

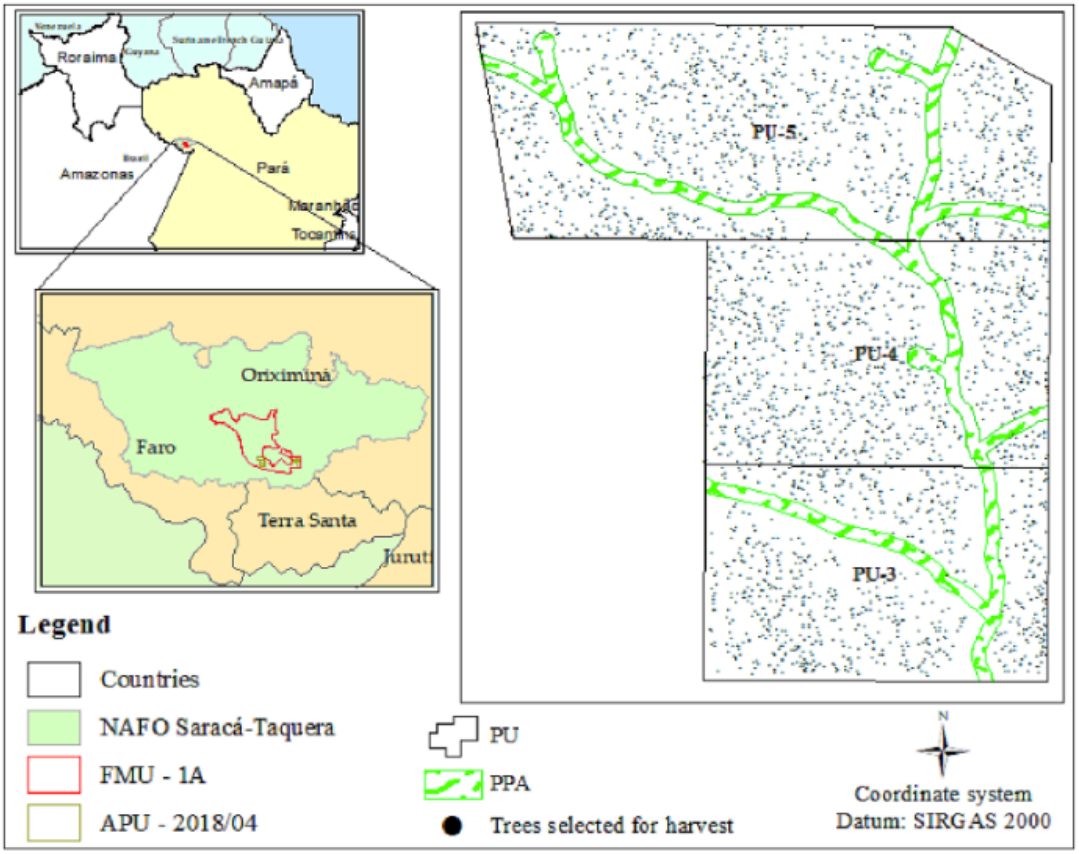
4. RESULTADOS

4.1. Instâncias

Para possibilitar a comparação com o estudo de Aguiar et al. (2023), este trabalho utiliza a mesma área de estudo: a Unidade de Manejo Florestal (UMF) 1A. Essa unidade abrange 26.897,96 hectares e está situada na Floresta Nacional (Flona) de Saracá-Taquera, nos municípios de Terra Santa e Oriximiná, Pará, nas coordenadas 1°45′23″ S e 56°34′21″ W.

A concessão florestal da UMF 1A foi concedida à empresa EBATA Produtos Florestais por meio do edital de concorrência nº 02/2012, organizado pelo Serviço Florestal Brasileiro, em conformidade com a Lei nº 11.284/2006 e o Decreto nº 6.063/2007. O estudo foca na Unidade de Planejamento Anual (UPA) 04/2018, abrangendo especificamente as Unidades de Produção (UP) 03, 04 e 05 (Figura 16).

Figura 16: Área de estudo



Fonte: Aguiar et al. (2023)

4.2. Parâmetros para o SA

Para avaliar o desempenho da aplicação do modelo proposto por Aguiar et al. (2023) no ambiente do QGIS foram utilizados os mesmos parâmetros (tabela 1) apresentados por ele.

Tabela 1: Parâmetros SA

Parâmetros	Valores
Temperatura de início	1.000
Temperatura de congelamento	0,001
Taxa de resfriação	0,985
Iterações de vizinhança	500
Iterações da solução inicial	20.000

Fonte: Adaptado de Aguiar et al. (2020).

4.3. Ambiente de teste

Todos os experimentos apresentados neste trabalho foram realizados no ambiente do QGIS 3.34.10 em um computador com processador i7 de 2,6 GHz com 6 núcleos e 12 threads, sistema operacional Windows 11 e 16GB de memória RAM. Inicialmente, o COMPFLOR foi desenvolvido na linguagem Python versão 3.12.5.

4.4. Resultados computacionais

A instância 1 (UP 03) foi executada 10 vezes, a instância 2 (UP 03 e 04) e 3 (UP 03, 04 e 05) foram executadas 4 vezes, cada uma das execuções teve um período de 16 horas (960 minutos) para garantir uma análise detalhada do desempenho do modelo. Esse período foi projetado para mitigar a perda computacional causada pela execução do algoritmo no ambiente do QGIS, permitindo uma comparação direta com os resultados apresentados por Aguiar et al. (2023).

Os resultados da alocação de pátios foram analisados considerando as mesmas métricas de Aguiar et al. (2023) como a melhor solução encontrada (Melhor Sol), a

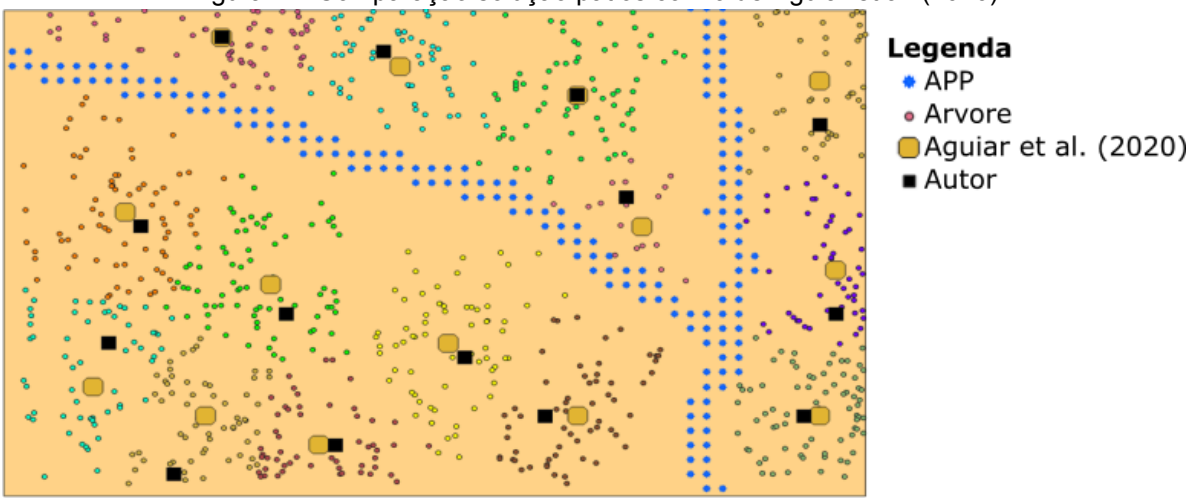
média das soluções (Média Sol), a porcentagem de desvio (Dev %), o tempo de execução da melhor solução (Tempo), o tempo médio de execução (Avg Tempo) e a relação entre a solução e a encontrada pelo CPLEX (GAP %). A Tabela 2 apresenta a comparação dos resultados obtidos para a alocação de pátios e a figura 17 a comparação dos pátios para a menor instância.

Tabela 2: Alocação de pátios comparados com de Aguiar et al. (2020)

Instância	Autor	Melhor Sol (m)	Media Sol (m)	Dev (%)	Tempo (min)	Avg Tempo (min)	GAP (%)
1	Aguiar et al. (2023)	105.602,37	105.936,26	0,32	05,22	14,23	1,73
	Autor	108.165,56	108.504,84	0,31	472,92	341,49	4,22
2	Aguiar et al. (2023)	261.498,34	263.705,69	0,84	4,30	18,69	2,38
	Autor	260.144,05	263.588,79	1,32	794,48	425,66	1,72
3	Aguiar et al. (2023)	442.211,64	446.283,81	0,92	18,18	18,32	- 5,03
	Autor	475.888,97	476.497,41	0,13	770,23	835,88	3,22

Fonte: Do autor

Figura 17: Comparação solução pátios com o de Aguiar et al. (2020)



Fonte: Do autor

Na Instância 1, o COMPFLOR obteve uma melhor solução de 108.165,56 m, com um GAP de 4,22%, enquanto Aguiar et al. (2023) alcançaram 105.602,37 m e um GAP de 1,73%. A diferença de 2,43% nas soluções sugere que a abordagem

proposta gera resultados competitivos, embora com um custo operacional ligeiramente superior.

Na Instância 2, o modelo superou o estudo de referência, alcançando uma melhor solução de 260.144,05 m, em comparação aos 261.498,34 m de Aguiar et al. (2023). Além disso, apresentou um GAP menor (1,72% contra 2,38%), sugerindo uma maior eficiência na otimização e uma melhor proximidade com a solução ideal.

Já na Instância 3, a solução obtida foi de 475.888,97 m, superior ao valor encontrado no estudo de referência (442.211,64 m). No entanto, o GAP positivo de 3,22% indica que a qualidade da solução do modelo foi inferior à do CPLEX. Em contraste, Aguiar et al. (2023) obteve um GAP negativo de - 5,03%, mostrando um desempenho melhor em relação ao método exato. Esse resultado sugere que o modelo precisa de ajustes para melhorar sua eficiência em cenários mais complexos.

Além da alocação de pátios, a Tabela 3 apresenta a comparação das estradas e trilhas de arraste, considerando o comprimento total (Comprimento) e a função objetivo (FO).

Tabela 3: Definição de estradas e trilhas comparada com Aguiar et al. (2023)

Instância	Infraestrutura	Aguiar et al. (2023)		Autor	
		Comprimento(m)	FO (m)	Comprimento(m)	FO (m)
1	Estrada	3.855,62	4.939,32	3.829,67	5.411,54
	Trilha de arraste	4.973,31	5.313,09	5.495,59	6.235,85
Total		**8.828,93**	**10.252,41**	**9.325,26**	**11.647,39**
2	Estrada	8.462,99	9.848,64	8.694,24	10.866,73
	Trilha de arraste	13.361,15	14.473,55	13.125,01	13.125,01
Total		**21.824,14**	**24.322,19**	**21.819,25**	**23.991,74**
3	Estrada	16.049,91	18.814,61	21.518,82	24.287,53
	Trilha de arraste	22.727,74	24.723,05	21.053,09	21.053,09
Total		**38.777,65**	**43.537,66**	**42.571,91**	**45.340,62**

Fonte: Do autor

Na Instância 1, o COMPFLOR resultou em um comprimento total de 9.325,26 m, representando um aumento de 5,61% em relação aos 8.828,93 m obtidos por Aguiar et al. (2023). A função objetivo (FO) foi de 11.647,39 m, 13,61% superior ao valor de 10.252,41 m do estudo de referência, indicando um custo operacional mais elevado.

Na Instância 2, os resultados obtidos foram bastante próximos aos da literatura, com um comprimento total de 21.819,25 m e uma função objetivo (FO) de 23.991,74 m. Esses valores são comparáveis a 21.824,14 m e 24.322,19 m, respectivamente, conforme Aguiar et al. (2023). A diferença na FO foi de 1,36% inferior ao valor do estudo de referência, sugerindo que o COMPFLOR obteve uma otimização ligeiramente mais eficiente dos custos. No entanto, essa aparente vantagem deve ser analisada com cautela, uma vez que a ausência de penalizações nas trilhas de arraste compromete a confiabilidade dos resultados, evidenciando a necessidade de uma reavaliação mais aprofundada do modelo.

Na Instância 3, o comprimento total das infraestruturas foi de 42.571,91 m, representando um aumento de 9,78% em relação aos 38.777,65 m apresentados por Aguiar et al. (2023). A FO também foi superior, alcançando 45.340,62 m, um incremento de 4,14% em comparação aos 43.537,66 m da literatura. Esses resultados indicam que, embora o modelo tenha gerado infraestruturas mais extensas, os custos associados permaneceram competitivos. No entanto, assim como na Instância 2, a falta de penalização nas trilhas de arraste levanta preocupações semelhantes sobre a confiabilidade dos resultados.

Embora promissora, essa performance confirma a perda de desempenho associada ao uso do Python no ambiente do QGIS. Isso evidencia a necessidade de otimizar as operações mais custosas por meio de módulos implementados em linguagens de baixo nível, como C++, a fim de melhorar a eficiência do modelo.

4.4.2. Otimização do cálculo da FO

Uma análise do tempo de execução do SA no QGIS mostrou que o cálculo da função objetivo (FO) é a operação mais onerosa. Essa complexidade decorre do fato de que, para cada iteração do algoritmo, é necessário calcular a distância de cada árvore explorável para todos os pátios disponíveis, resultando em uma

complexidade linear de $O(A * P)$, onde A representa o número de árvores exploráveis e P, o número de pátios. No contexto do problema, a menor instância apresenta 1.666 vértices de pátios e 820 árvores exploráveis, totalizando 1.366.120 operações. Esse custo computacional elevado limita a quantidade de iterações do SA e, consequentemente, sua capacidade de explorar o espaço de soluções de forma eficiente.

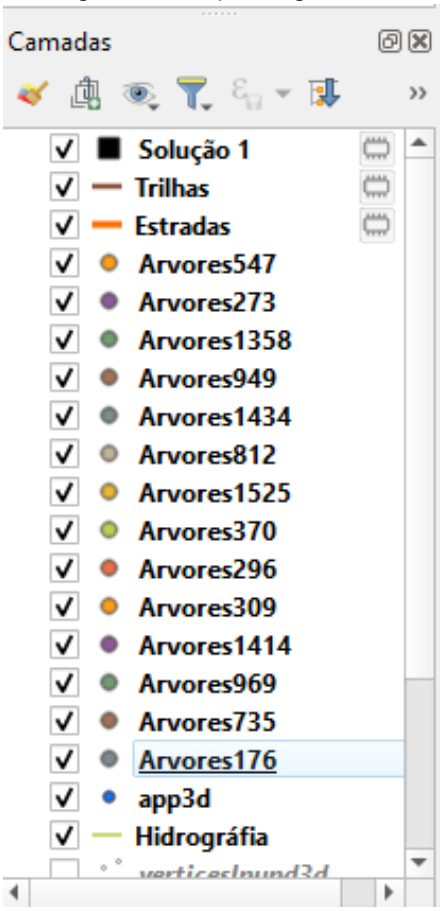
Para mitigar esse gargalo, foi desenvolvido um módulo de otimização em C++ integrado à API do Python. O objetivo era reduzir o tempo de processamento da FO, permitindo que o SA avaliasse mais soluções em um determinado período. A decisão de implementar o módulo em C++ se baseou na sua eficiência em cálculos computacionais intensivos, especialmente em operações matriciais, como exigido pelo problema de alocação de pátios de estocagem e definição de trilhas de arraste.

A integração com Python, através da API, visava manter a flexibilidade e a facilidade de uso da linguagem para o restante do sistema. No entanto, desafios inesperados na comunicação entre C++ e Python impediram a implementação efetiva do módulo na versão final do sistema. Ainda assim, a análise do impacto da complexidade do cálculo da FO reforça a necessidade de utilizá-la para melhorar o desempenho do sistema.

4.4.3. Arquivos gerados

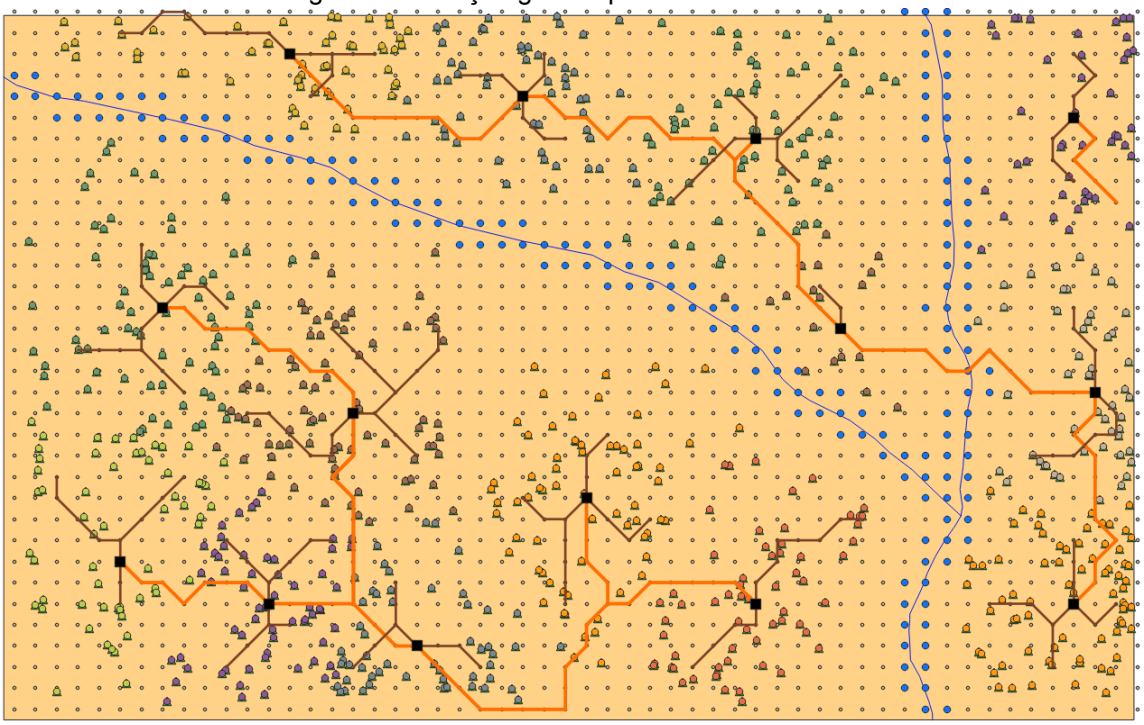
Como resultado, o COMPFLOR gera uma solução otimizada para a alocação de pátios, estradas e trilhas, facilitando a análise do profissional. Para isso, são criados *shapefiles* específicos para os pátios, estradas, trilhas e árvores associadas a cada pátio (Figura 18). Dessa forma, a solução gerada permite uma visualização clara da infraestrutura planejada, auxiliando na tomada de decisões e no planejamento do manejo florestal.

Figura 18: Arquivos gerados



Fonte: Do autor

Figura 19: Solução gerada para menor instância



Fonte: Do autor

5. CONCLUSÕES

A otimização do planejamento de infraestruturas no Manejo Florestal Sustentável (MFS) desempenha um papel fundamental na preservação ambiental e na eficiência operacional das empresas do setor. A definição estratégica de pátios de estocagem, estradas e trilhas de arraste permite reduzir custos, minimizar o impacto ambiental e aprimorar a gestão dos recursos florestais. Ao integrar algoritmos de otimização como *Simulated Annealing* (SA) e Dijkstra, o COMPFLOR propõe uma abordagem computacional viável para o planejamento de exploração florestal, combinando técnicas heurísticas e ferramentas de geoprocessamento.

O presente trabalho teve como objetivo principal desenvolver e validar o COMPFLOR, um software para alocação otimizada das infraestruturas do MFS, com base na metodologia proposta por Aguiar et al. (2023). Considerando que empresas do ramo de manejo florestal habitualmente adotam softwares de geoprocessamento em seu planejamento, entendemos que tais softwares são o melhor local para disponibilizar o recurso para a otimização da alocação das infraestruturas. Neste aspecto, a escolha do QGIS como software de geoprocessamento para o COMPFLOR, se deve ao fato de que, o mesmo tem sido amplamente adotado pelas empresas do ramo devido à sua principal característica de ser um software aberto e gratuito.

A implementação do COMPFLOR dentro do ambiente QGIS possibilitou a criação de uma ferramenta acessível e adaptável ao planejamento real de empresas do setor. Os testes realizados demonstraram que o software desenvolvido apresentou desempenho competitivo em relação a estudos anteriores, garantindo otimizações na disposição das infraestruturas e contribuindo para a mitigação dos impactos ambientais da atividade madeireira.

Em suma, o COMPFLOR forneceu uma solução prática para otimizar a alocação das infraestruturas no MFS, utilizando técnicas de otimização e integração computacional. Os resultados obtidos indicam que o COMPFLOR pode auxiliar gestores e engenheiros florestais na tomada de decisões estratégicas, promovendo um manejo mais sustentável e eficiente. Dessa forma, espera-se que este trabalho contribua para avanços na área de planejamento florestal, possibilitando a

implementação de práticas mais responsáveis e alinhadas com a preservação ambiental.

Para trabalhos futuros, recomenda-se o aprimoramento dos algoritmos utilizados, explorando outras técnicas de otimização e reavaliação dos resultados das instâncias 2 e 3. Além de expandir os testes do complemento para uma gama mais ampla de instâncias, incluindo áreas florestais com características distintas (ex.: diferentes densidades de árvores, topografias mais acidentadas ou florestas em outros biomas além da Amazônia). Essa validação adicional ajudaria a verificar a robustez e a generalização da ferramenta.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AGUIAR, Marcelo Otone. **Otimização do planejamento integrado de infraestrutura na exploração florestal na região amazônica**. 2019. Tese (Doutorado em Ciências Florestais) - Universidade Federal do Espírito Santo, Centro de Ciências Agrárias e Engenharias, Jerônimo Monteiro - ES, 2019.

AGUIAR, M. O. et al. **Integrated planning of forest exploration infrastructures in an Amazonian sustainable forest management area**. *Forest Ecology and Management*, Volume 549, 2023, p. 121265. ISSN 0378-1127. Disponível em: https://doi.org/10.1016/j.foreco.2023.121265.

.AGUIAR, M. O. et al. **Metaheuristics applied for storage yards allocation in an Amazonian sustainable forest management area**. *Journal of Environmental Management*, Volume 271, 2020, p. 110926. ISSN 0301-4797. Disponível em: https://doi.org/10.1016/j.jenvman.2020.110926.

AGUIAR, M. O. et al. **Optimizing forest road planning in a sustainable forest management area in the Brazilian Amazon**. *Journal of Environmental Management*, Volume 288, 2021, p. 112332. ISSN 0301-4797. Disponível em: https://doi.org/10.1016/j.jenvman.2021.112332.

AKAY, A. E. **Minimizing total costs of forest roads with computer-aided design model**. *Sadhana*, Bangalore, Karnataka, India, v. 31, n. 5, p. 621–633, out. 2006. ISSN 0973-7677. Disponível em: http://dx.doi.org/10.1007/BF02715918.

AMARAL, P.; VERÍSSIMO, A.; BARRETO, P.; VIDAL, E. **Floresta para sempre: um manual para a produção de madeira na Amazônia**. Belém, PA: Imazon, 1998. 130 p.

ARIMA, E. Y.; WALKER, R. T.; SALES, M.; SOUZA JÚNIOR, C.; PERZ, S. G. **The fragmentation of space in the Amazon Basin**. *Photogramm. Eng. Remote Sens.*, v. 74, n. 6, p. 699–709, 2008. Disponível em: https://doi.org/10.14358/PERS.74.6.699.

.

ARIMA, E. Y.; WALKER, R.; MESSINA, J.; SOARES-FILHO, B.; PERZ, S.; VERGARA, D.; SALES, M.; PEREIRA, R.; CASTRO, W. **Modeling spatial decisions with graph theory: logging roads and forest fragmentation in the Brazilian Amazon**. *Ecological Applications*, v. 23, n. 1, p. 239–254, 2013. ISSN 1051-0761. Disponível em: https://doi.org/10.1890/11-1800.1.

ARORA, J. S. **Introduction to optimum design**. 2. ed. San Diego, California, USA: Elsevier Academic Press, 2004. 728 p. ISBN 0-12-064155-0.

ANTONIOU, A.; LU, W. **Practical Optimization: Algorithms and engineering applications**. Victoria, British Columbia, CA: Springer, 2007. 670 p. ISBN 978-0-387-71106-5.

BALIEIRO, M. R.; ESPADA, A. L. V.; NOGUEIRA, O.; PALMIERI, R.; LENTINI, M. **As concessões de florestas públicas na Amazônia Brasileira: um manual para pequenos e médios produtores florestais**. 2. ed. Piracicaba, SP: Imaflora, 2010. 204 p. ISBN 978-85-98081-36-6.

BRAZ, E. M.; D'OLIVEIRA, M. V. N.; ARAÚJO, H. J. B. de; MIRANDA, E. M. de. **Plano de exploração sob critérios de manejo florestal de baixo impacto**. Rio Branco, AC: Embrapa-CPAF/AC, 1998. 20 p. Circular Técnica, 27.

BRAZ, E. M.; D'OLIVEIRA, M. V. N.; ARAÚJO, H. J. B. de; MIRANDA, E. M. de. **Planejamento da Exploração em Florestas Naturais**. 1. ed. Colombo, PR: Embrapa Florestas, 2005. 32 p. ISBN 1517-526X.

EPSTEIN, R.; WEINTRAUB, A.; SAPUNAR, P.; NIETO, E.; SESSIONS, J. B.; SESSIONS, J.; BUSTAMANTE, F.; MUSANTE, H. **A combinatorial heuristic approach for solving real-size machinery location and road design problems in forestry planning**. *Operations Research*, v. 54, n. 6, p. 1017–1027, 2006. Disponível em: https://doi.org/10.1287/opre.1060.0331.

ESPADA, A. L. V.; PIRES, I. P.; LENTINI, M. A. W.; BITTENCOURT, P. R. G. **Manejo Florestal e Exploração de Impacto Reduzido em Florestas Naturais de Produção da Amazônia: Informativo técnico 1**. Altamira, PA: Instituto Florestal Tropical, 2013. 31 p.

FFT – FUNDAÇÃO FLORESTA TROPICAL. **Manual de procedimentos técnicos para condução de manejo florestal e exploração de impacto reduzido**. Belém, PA: Fundação Florestal Tropical, 2002. 89 p.

FIGUEIREDO, E. O.; BRAZ, E. M.; D'OLIVEIRA, M. V. N. **Manejo de precisão em florestas tropicais: modelo digital de exploração florestal**. 1. ed. Rio Branco, AC: Embrapa Acre, 2007. 183 p. ISBN 978-85-99190-04-3.

GOLDBARG, M. C.; LUNA, H. P. L. **Otimização combinatória e programação linear: modelos e algoritmos**. 2. ed. Rio de Janeiro, RJ: Elsevier, 2005. 518 p. ISBN 978-85-352-1520-5.

GONZALEZ, T. F. **Handbook of approximation algorithms and metaheurististics**. 1. ed. Boca Raton, Florida, USA: Chapman & Hall, 2007. 1427 p. ISBN 978-1-58488-550-4.

HOLMES, T. P. et al. **Financial and ecological indicators of reduced impact logging performance in the eastern Amazon**. *Forest Ecology and Management*, Amsterdam, Netherlands, v. 163, n. 1, p. 93–110, mar. 2002. ISSN 0378-1127. Disponível em: https://doi.org/10.1016/S0378-1127(01)00530-8.

LACHTERMACHER, G. **Pesquisa operacional na tomada de decisões**. 5. ed. Rio de Janeiro, RJ: LTC, 2018. 185 p. ISBN 978-85-216-3048-7.

MARTINHAGO, A. Z. **Otimização para alocação de pátios de estocagem para exploração de impacto reduzido na Amazônia Brasileira**. 2012. 162 f. Tese (Doutorado em Engenharia Florestal) — Universidade Federal de Lavras, Lavras, MG, 2012.

MARCATTI, G. E. **Otimização do traçado geométrico de estradas florestais**. 2017. 47 f. Tese (Doutorado em Ciência Florestal) — Universidade Federal de Viçosa, Viçosa, MG, 2017.

NIEUWENHUIS, M. **Operations research in forestry**. *Irish Forestry*, Dublin, Ireland, v. 46, n. 1, p. 51–58, 1989. Disponível em: https://journal.societyofirishforesters.ie/index.php/forestry/article/viewFile/9654/8776.

.

ORLIN, J. B.; MADDURI, K.; SUBRAMANI, K.; WILLIAMSON, M. **A faster algorithm for the single source shortest path problem with few distinct positive lengths**. *Journal of Discrete Algorithms*, v. 8, n. 2, p. 189–198, 2010. ISSN 1570-8667. Disponível em: https://doi.org/10.1016/j.jda.2009.03.001.

PIZZOLATO, Nelio Domingues; RAUPP, Fernanda M. P.; ALZAMORA, Guina Sotomayor. **REVISÃO DE DESAFIOS APLICADOS EM LOCALIZAÇÃO COM BASE EM MODELOS DA p-MEDIANA E SUAS VARIANTES**. *Pesquisa Operacional para o Desenvolvimento*, [S. l.], v. 4, n. 1, p. 13–42, 2012. Disponível em: https://www.podesenvolvimento.org.br/podesenvolvimento/article/view/174. Acesso em: 6 dez. 2023.

RIBEIRO, C. A. A. S.; MARCATTI, G. E.; MUNARETTI, A. M.; SILVA, K. E. da; ALMEIDA, D. R. A. de; VASCONCELOS, R. dos S.; NETO, E. L. V. **Otimização do traçado de trilhas para acesso a recursos florestais naturais**. Manaus, AM: Embrapa Amazônia Ocidental, 2017. 99 p. ISSN 1517-3135.

SABOGAL, C.; LENTINI, M.; POKORNY, B.; SILVA, J. N. M.; ZWEEDE, J.; VERÍSSIMO, A.; BOSCOLO, M. **Manejo florestal empresarial na Amazônia Brasileira: Restrições e oportunidades**. Belém, PA: Center for International Foresty Research - Cifor, 2006. 72 p. ISBN 979-24-4635-4.

SILVA, E. F. da; SILVA, G. F. da; FIGUEIREDO, E. O.; BINOTI, D. H. B.; MENDONÇA, A. R. de; TORRES, C. M. M. E.; PEZZOPANE, J. E. M. **Allocation of storage yards in management plans in the Amazon by means of mathematical programming**. *Forests*, Basel, Switzerland, v. 9, n. 3, p. 1–20, mar. 2018. ISSN 1999-4907. Disponível em: https://doi.org/10.3390/f9030127.

SØVDE, N. E.; LøKKETANGEN, A.; TALBOT, B. **Applicability of the GRASP metaheuristic method in designing machine trail layout**. *Forest Sci. Technol.*, v. 9, p. 187–194, 2013. Disponível em: https://doi.org/10.1080/21580103.2013.839279.

7. APÊNDICES

Apêndice A - SAStorageYard

```
def SAStorageYard(
    self, floresta: list[ArvoreExploravel],
    distancias: list[list[float]],
    solInicial: SolucaoStorageYard,
    restVolSup: float,
    txResfriamento: float,
    iteracoesVizinhanca: float,
    tempInicial: float,
    tempCongelamento: float
) -> SolucaoStorageYard:

    # inicializando variáveis
    tfInicio = time.time() # get_time
    i = 0
    j = 0
    k = 0
    patio = 0
    cont = 0
    contViaveis = 0
    contInviaveis = 0
    tInicio = time.time() # inicializando o tempo atual
    tFim = tInicio + self.TEMPOEXEC

    melhorSol = copy.deepcopy(solInicial)
    solucaoInicial = copy.deepcopy(solInicial)
    vizinho = SolucaoStorageYard()
    melhorSol.numViaveis = 0
    melhorSol.numInviaveis = 0
    patios = [0 for _ in range(self.NUM_VERTICES_PATIOS)]
    tempoTotalCalculoFOSA = 0.0

    # Marcando com 1 os elementos da solução inicial
    for k in range(self.NUM_PATIOS):
        patio = solucaoInicial.patios[k]
        patios[patio - 1] = 1

    # Inicializando variáveis
    IterTemp = 0
```

```
39      temp = tempInicial
40      x = 0.0
41      variacao = 0.0
42
43      while temp > tempCongelamento and time.time() < tFim:
44          while IterTemp < iteracoesVizinhanca and time.time() < tFim:
45              IterTemp += 1
46              cont += 1
47              vizinho = self.gerarVizinhoPatio(floresta, distancias,
                ↪  solucaoInicial, patios, restVolSup)
48              tempoTotalCalculoFOSA += vizinho.t
49              variacao = vizinho.FO - solucaoInicial.FO
50
51              if vizinho.viavel:
52                  contViaveis += 1
53              else:
54                  contInviaveis += 1
55
56              patios = [0 for _ in range(self.NUM_VERTICES_PATIOS)]
57
58              if variacao < 0:
59                  solucaoInicial = vizinho
60                  for i in range(self.NUM_PATIOS):
61                      patio = vizinho.patios[i]
62                      patios[patio - 1] = 1
63
64                  if vizinho.FO < melhorSol.FO:
65                      melhorSol = vizinho
66                      melhorSol.tempoSol = time.time() - tfInicio
67              else:
68                  x = randint(0, 1000) / 1000
69                  if x < math.pow(math.e, (-variacao / temp)):
70                      solucaoInicial = vizinho
71                      for i in range(self.NUM_PATIOS):
72                          patio = vizinho.patios[i]
73                          patios[patio - 1] = 1
74                  else:
75                      for j in range(self.NUM_PATIOS):
```

```
                        patio = solucaoInicial.patios[j]
                        patios[patio - 1] = 1

        temp *= txResfriamento

        if temp < (tempCongelamento + (tempCongelamento * 2)) and
        ↪  time.time() < tFim:
            temp = tempInicial

        IterTemp = 0

    melhorSol.tempo += time.time() - tfInicio
    melhorSol.numIteracoes = cont
    melhorSol.numViaveis = contViaveis
    melhorSol.numInviaveis = contInviaveis
    melhorSol.tempoCalculoFO_SA = tempoTotalCalculoFOSA
    melhorSol.tempoSA = time.time() - tfInicio
    return melhorSol
```

Apêndice B - gerarVizinhoPatio

```
1  def gerarVizinhoPatio(
2      self,
3      floresta: list[ArvoreExploravel],
4      distancias: list[list[float]],
5      sol: SolucaoStorageYard,
6      patios: list[int],
7      restVolSup: float
8  ):
9
10     solAtual = copy.deepcopy(sol)
11     vizinho = 0
12
13     # obtendo a posicao aleatoria de troca
14     # troca = randint(1, 32767) % NUM_PATIOS
15     troca = randint(1, self.NUM_PATIOS) - 1
16
17     while True:
18         # vizinho = 1 + (randint(1, 32767) % NUM_VERTICES_PATIOS)
19         vizinho = randint(1, self.NUM_VERTICES_PATIOS)
20         if patios[vizinho - 1] != 1:
21             solAtual.patios[troca] = vizinho
22             break
23
24     # depois de selecionar um número, marca ele como já tendo sido
       ↪ selecionado
25     patios[vizinho - 1] = 1
26
27     heuristica = Heuristicas(self.NUM_PATIOS,
       ↪ self.NUM_ARVORES_EXPLORAVEIS, self.DISTANCIA_MAXIMA,
       ↪ self.NUM_VERTICES_PATIOS, self.PENALIZACAO_VOLUME)
28     solAtual = heuristica.calculaFOPatio(floresta, distancias, solAtual,
       ↪ restVolSup)
29
30     return solAtual
```

Apêndice C - CalculaFOPatio

```
def calculaFOPatio(
    self,
    floresta:
    list[ArvoreExploravel],
    distancias: list[list[float]],
    solAtual: SolucaoStorageYard,
    restVolSup: float
) -> SolucaoStorageYard:
    tInicio = time.time()
    patio = 0
    distancia = 0
    distanciaMenor = 0
    diferencaVolume = 0
    diferencaDistancia = 0
    distanciaTotal = 0
    res = copy.deepcopy(solAtual)
    res.t = 0.0

    res.volumes = [0 for _ in range(self.NUM_PATIOS)]
    res.arvores = [[] for _ in range(self.NUM_PATIOS)]

    #Comparando as distancias dos pontos aleatorios
    for j in range(self.NUM_ARVORES_EXPLORAVEIS):
        distanciaMenor = math.inf
        patio = 0
        for k in range(self.NUM_PATIOS):
            distancia = distancias[j][res.patios[k] - 1]
            if distancia < distanciaMenor:
                distanciaMenor = distancia
                patio = k

        if distanciaMenor > self.DISTANCIA_MAXIMA:
            diferencaDistancia += distanciaMenor - self.DISTANCIA_MAXIMA
        distanciaTotal += distanciaMenor

        nova_arvore = {'id': floresta[j].id, 'numero':
        ↪  floresta[j].numero}
        res.arvores[patio].append(nova_arvore)
```

```
38          res.volumes[patio] += floresta[j].volume
39
40      for w in range(self.NUM_PATIOS):
41          diferencaVolume += (res.volumes[w] - restVolSup) if
            ↪  (res.volumes[w] > restVolSup) else 0
42
43      res.distanciaTotal = distanciaTotal
44      #penalizar solucao inviavel
45      distanciaTotal +=  self.PENALIZACAO_VOLUME * (diferencaVolume +
        ↪  diferencaDistancia) # penalizacao é o theta e diferença é g()
46
47      res.FO = distanciaTotal
48      res.viavel = (diferencaVolume + diferencaDistancia) == 0
49
50      res.t = time.time() - tInicio
51      return res
```

Apêndice D - RoadsAprovTrech

```
1  def roadsAprovTrecho(
2      self,
3      grafo: Grafo,
4      area: list[Area],
5      patios: list[Patio],
6      solPatios: SolucaoStorageYard,
7      estradasAcesso: AccessRoad,
8      num_patios: int,
9      num_vertices: int,
10     num_access_road: int
11 ):
12     global NUM_PATIOS, NUM_VERTICES, NUM_ROADS, NUM_ACCESS_ROAD
13     NUM_PATIOS= num_patios
14     NUM_VERTICES = num_vertices
15     NUM_ROADS = NUM_PATIOS + 1
16     NUM_ACCESS_ROAD = num_access_road
17
18     patioProibido = -1
19     solRoad = SolucaoRoad(NUM_VERTICES)
20     sol = SolucaoRoads(NUM_ROADS, NUM_VERTICES)
21     vEstSecOrder = [-1 for _ in range(NUM_PATIOS)]
22     vDistEstradaSec = [INFINITY for _ in range(NUM_PATIOS)]
23     vetSelRoads = [0 for _ in range(NUM_ROADS)]
24     vDistPat = [0.0]
25     vDistEstPri = [0.0]
26
27     #executando o preprocessamento das roads
28     preRods = PreProcecamentoRoads(NUM_PATIOS, patios, solPatios)
29     distanciaPatios = preRods.distanciaPatios
30     patiosOrder = preRods.patiosOrder
31
32     #ligando os patios com as estradas de acesso
33     for i in range(NUM_ACCESS_ROAD):
34         vEstradaPri = estradasAcesso.inicio[i]
35         pMenor = -1
36         patioCon = -1
37         menor = INFINITY
38
```

```
39          #verificando qual patio esta mais proximo
40          for j in range(NUM_PATIOS):
41              patioAtual = solPatios.patios[j]
42              vPatioAtual = patios[patioAtual - 1].vertice - 1
43              vDistPat[0] = calculaDistancia(area[vEstradaPri],
                ↪  area[vPatioAtual])
44              if (vDistPat[0] < menor):
45                  patioCon = j
46                  pMenor = vPatioAtual
47                  menor = vDistPat[0]
48
49          #Realiza a ligacao ate a estrada primaria
50          solRoad = grafo.Dijkstra(pMenor, vEstradaPri)
51          typeRoad = 1
52          vetSelRoads[patioCon] = 1
53
54          #atualiza os vertices das estradas secundarias em relacao aos
            ↪  patios
55          self.atualizaVerticesEstSecundarias(area, patios, solPatios,
            ↪  solRoad, estradasAcesso, vEstSecOrder, vDistEstradaSec, 0,
            ↪  patioProibido)
56          sol.roads[patioCon] = solRoad
57          sol.typeRoad[patioCon] = typeRoad
58          sol.tempoTotal += solRoad.tempo
59          sol.distanciaTotal += solRoad.distanciaTotal
60          sol.FOTotal += solRoad.FO
61
62      pMenor = -1
63      menor = INFINITY
64
65      #ligando os patios as estradas e outros patios
66      for i in range(NUM_PATIOS):
67          #apenas patios que nao foram ligados ainda
68          if vetSelRoads[i] == 0:
69              patioProibido = -1
70              patioAtual = solPatios.patios[i]    #equivale ao id dos
                ↪  patios
71              vPatioAtual = patios[patioAtual - 1].vertice - 1
```

```
72              #estrada primaria
73              vEstradaPri = self.obterVerticeInicioProximo(area,
                ↪  estradasAcesso, vPatioAtual, vDistEstPri)
74              #estrada secundaria
75              vEstradaSec = vEstSecOrder[i]
76              vDistEstSec = vDistEstradaSec[i]
77
78              #verifica se o patio esta em algum vertice da estrada
                ↪  principal
79              if vDistEstPri[0] != 0 and vDistEstSec != 0:
80                  patioCon = self.obterPatioProximo(area, patios,
                    ↪  vEstradaPri, vDistEstPri[0], i, distanciaPatios,
                    ↪  patiosOrder, vDistPat)
81
82                  if patioCon != -1:
83                      if vDistEstPri[0] <= vDistEstSec and vDistEstPri[0]
                        ↪  <= vDistPat[0]:
84                          #realiza a ligacao ate a estrada primaria
85                          solRoad = grafo.Dijkstra(vPatioAtual,
                            ↪  vEstradaPri)
86                          typeRoad = 1
87                          vetSelRoads[i] = 1
88                      elif vDistEstSec <= vDistEstPri[0] and vDistEstSec
                        ↪  <= vDistPat[0]:
89                          #realiza a ligacao ate a estrada secundaria
90                          solRoad = grafo.Dijkstra(vPatioAtual,
                            ↪  vEstradaSec)
91                          typeRoad = 2
92                          vetSelRoads[i] = 1
93                      else:
94                          #realiza a ligacao ate outro patio
95                          solRoad = grafo.Dijkstra(vPatioAtual,
                            ↪  patios[patioCon - 1].vertice - 1)
96                          typeRoad = 2
97                          #cria uma ligacao inversa proibida are os dois
                            ↪  patios
98                          patioProibido = patioCon
99                          for j in range(i+1, NUM_PATIOS):
```

```
100                             if patioCon == solPatios.patios[j]:
101                                 #marcando o proximo patio na busca
102                                 for k in range(1, NUM_PATIOS):
103                                     if patiosOrder[j][k] == patioAtual:
104                                         patiosOrder[j][k] = -1
105                                         break
106                                 break
107                 else:
108                     if vDistEstPri[0] <= vDistEstSec:
109                         #realiza a ligacao ate a estrada primaria
110                         solRoad = grafo.Dijkstra(vPatioAtual,
                            ↪  vEstradaPri)
111                         typeRoad = 1
112                         vetSelRoads[i] = 1
113                     else:
114                         #realiza a ligacao ate a estrada secundaria
115
116                         solRoad = grafo.Dijkstra(vPatioAtual,
                            ↪  vEstradaSec)
117                         typeRoad = 2
118                         vetSelRoads[i] = 1
119                 #atualiza os vertices das estradas secundaria em
                    ↪  relacao aos patios
120                 self.atualizaVerticesEstSecundarias(area, patios,
                    ↪  solPatios, solRoad, estradasAcesso, vEstSecOrder,
                    ↪  vDistEstradaSec, i+1, patioProibido)
121                 sol.roads[i] = solRoad
122                 sol.typeRoad[i] = typeRoad
123                 sol.tempoTotal += solRoad.tempo
124                 sol.distanciaTotal += solRoad.distanciaTotal
125                 sol.FOTotal += solRoad.FO
126             else:
127                 #gera uma solucao vazia
128                 solRoad.estrada.inicio = vPatioAtual
129                 solRoad.estrada.termino = vPatioAtual
130                 solRoad.antecessor[vPatioAtual] = -1
131                 solRoad.sucessor[vPatioAtual] = -1
132                 solRoad.distancia[vPatioAtual] = 0
```

```
133                 solRoad.peso[vPatioAtual] = 0
134                 solRoad.numVerticesRota = 0
135                 solRoad.FO = 0
136                 solRoad.distanciaTotal = 0
137                 solRoad.tempo = 0
138                 solRoad.tempoSol = 0
139                 sol.typeRoad[i] = 1
140                 sol.roads[i] = solRoad
141                 vetSelRoads[i] = 1;
142     #ligando o vertice da saida da UT
143     if estradasAcesso.termino != 0:
144         for i in range(NUM_PATIOS):
145             patioAtual = solPatios.patios[i]
146             vPatioAtual = patios[patioAtual - 1].vertice - 1
147             vDistPat[0] = calculaDistancia(area[vPatioAtual],
                ↪  area[estradasAcesso.termino])
148             if vDistPat[0] < menor:
149                 pMenor = vPatioAtual
150                 menor = vDistPat[0]
151         #construindo o tracado final
152         solRoad = grafo.Dijkstra(pMenor, estradasAcesso.termino)
153         sol.typeRoad[i] = 1
154         sol.roads[NUM_ROADS - 1] = solRoad
155         sol.tempoTotal += solRoad.tempo
156         sol.distanciaTotal += solRoad.distanciaTotal
157         sol.FOTotal += solRoad.FO
158         vetSelRoads[i] = 1
159     else:
160         #gera uma soluo vazia
161         solRoad.estrada.inicio = estradasAcesso.termino
162         solRoad.estrada.termino = estradasAcesso.termino
163         solRoad.antecessor[0] = -1
164         solRoad.sucessor[0] = -1
165         solRoad.distancia[0] = 0
166         solRoad.peso[0] = 0
167         solRoad.numVerticesRota = 0
168         solRoad.FO = 0
169         solRoad.distanciaTotal = 0
```

```
170         solRoad.tempo = 0
171         solRoad.tempoSol = 0
172         sol.typeRoad[NUM_ROADS - 1] = 1
173         sol.roads[NUM_ROADS - 1] = solRoad
174         vetSelRoads[NUM_ROADS - 1] = 1
175     self.determinarTipoEstrada(sol, vetSelRoads)
176     return sol
```

Apêndice E - TrailsAprovTrecho

```
1  def trailsAprovTrilhas(
2      grPatio: Grafo,
3      area: List[Area],
4      desvios: Optional[List[Desvio]],
5      patioAtual: int,
6      qtdTrilhas: int,
7      solPtTrilhas: SolucaoPtTrails,
8      NUM_VERTICES: int
9  ):
10     solTrilha = SolucaoRoad(NUM_VERTICES)
11     tInicio = time.time()
12     sol = cria_SolTrails(qtdTrilhas, NUM_VERTICES)
13     sol.patio = patioAtual
14     sol.distanciaTotal = 0
15     sol.FOTotal = 0
16     sol.tempoTotal = 0
17
18     #Tracado de trilhas de arraste
19     pInicioTrilha = 0
20
21     vTrilhaOrder = [-1 for _ in range(qtdTrilhas)]
22     vDistTrilha = [INFINITY for _ in range(qtdTrilhas)]
23
24     for i in range(qtdTrilhas):
25         tInicioTrilha = time.time()
26         pTerminoTrilha = solPtTrilhas.patios[i]
27         #obtendo as distancias
28         temp = grPatio.Dijkstra(pInicioTrilha, pTerminoTrilha)
29         vDistanciaPonto = temp.FO
30         #trilha
31         vTrilha = vTrilhaOrder[i]
32         vDistanciaTrilha = vDistTrilha[i]
33
34         if vDistanciaTrilha != 0:
35             if vDistanciaPonto <= vDistanciaTrilha:
36                 solTrilha = grPatio.Dijkstra(pInicioTrilha,
                   ↪  pTerminoTrilha)
37             else:
```

```
38                  solTrilha = grPatio.Dijkstra(vTrilha, pTerminoTrilha)
39              atualizaVerticesTrilhas(grPatio, solPtTrilhas, solTrilha,
                ↪ pInicioTrilha, qtdTrilhas, i+1, vTrilhaOrder,
                ↪ vDistTrilha)
40          else:
41              #gera uma solucao vazia
42              solTrilha.estrada.inicio = vTrilha
43              solTrilha.estrada.termino = pTerminoTrilha
44              solTrilha.antecessor[vTrilha] = -1
45              solTrilha.sucessor[vTrilha] = -1
46              solTrilha.distancia[vTrilha] = 0
47              solTrilha.peso[vTrilha] = 0
48              solTrilha.numVerticesRota = 0
49              solTrilha.FO = 0
50              solTrilha.distanciaTotal = 0
51              solTrilha.verticesRoad = []
52
53          solTrilha.tempo = solTrilha.tempoSol = (time.time() -
            ↪ tInicioTrilha) / 100.0
54          sol.FOTotal = sol.FOTotal + solTrilha.FO
55          sol.distanciaTotal = sol.distanciaTotal +
            ↪ solTrilha.distanciaTotal
56          sol.roads[i] = solTrilha
57
58      sol.tempoTotal = (time.time() - tInicio) / 100.0
59      return sol
```