UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
CAMPUS DE ILHA SOLTEIRA

BRUNO BETTIOL STRIEDER

ANÁLISE DA TÉCNICA DE SOLUÇÃO PARA O PROBLEMA DE ALOCAÇÃO ÓTIMA DAS CHAVES SECCIONADORAS EM REDES DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA

Ilha Solteira - SP
2025

PROGRAMA DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

BRUNO BETTIOL STRIEDER

ANÁLISE DA TÉCNICA DE SOLUÇÃO PARA O PROBLEMA DE ALOCAÇÃO ÓTIMA DAS CHAVES SECCIONADORAS EM REDES DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia de Ilha Solteira - UNESP, para obtenção do título de bacharel em Engenharia Elétrica

Orientador(a): Prof. Dr. Jonatas Boas Leite

Ilha Solteira - SP

2025

FICHA CATALOGRÁFICA
Desenvolvida pela Diretoria Técnica de Biblioteca e Documentação

Strieder, Bruno Bettiol.
S917a Análise da técnica de solução para o problema de alocação ótima das chaves seccionadoras em redes de distribuição de energia elétrica / Bruno Bettiol Strieder. -- Ilha Solteira: [s.n.], 2025
61 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2025

Orientador: Jonatas Boas Leite
Inclui bibliografia

1. Algoritmo genético. 2. Chaves seccionadoras. 3. Custo de interrupção ao consumidor.

Elaborada por Raiane da Silva Santos - CRB-8/9999

IMPACTO POTENCIAL DESTA PESQUISA

Esta pesquisa demonstra que a escolha da codificação em algoritmos genéticos é determinante para a eficiência na otimização de redes de distribuição. No aspecto técnico e científico, valida a metodologia de codificação inteira como uma solução mais robusta e rápida para a alocação de chaves. No aspecto socioeconômico, a minimização do custo de interrupção ao consumidor (CIC) melhora a confiabilidade do fornecimento e reduz prejuízos operacionais, contribuindo para o desenvolvimento sustentável e para a modernização da gestão de ativos nas concessionárias de energia.

POTENTIAL IMPACT OF THIS RESEARCH

This research demonstrates that the choice of encoding in genetic algorithms is decisive for efficiency in distribution network optimization. From a technical and scientific standpoint, it validates the integer encoding methodology as a more robust and faster solution for switch allocation. From a socioeconomic perspective, minimizing the customer interruption cost (CIC) improves supply reliability and reduces operational losses, contributing to sustainable development and the modernization of asset management in power utilities.

UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Ilha Solteira

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos dois dias do mês de novembro do ano de dois mil e vinte e cinco, o discente ***Bruno Bettiol Strieder,*** matriculado sob o nº 191053295, tendo como banca examinadora o seu orientador, *o Prof. Dr. Jonatas Boas Leite, o Doutorando Vagner Antonio de Moraes da Cruz e o Mestrando Sergie Josue Peña Cruz,* apresentou o Trabalho de Graduação intitulado **"Análise da Técnica de Solução para o Problema de Alocação Ótima das Chaves Seccionadoras em Redes de Distribuição de Energia Elétrica",** obtendo a nota 10,0 (DEZ) e conceito APROVADO.

Prof. Dr. Jonatas Boas Leite
- Orientador-

Bruno Bettiol Strieder
- Discente-

Doutorando Vagner Antonio de Moraes da Cruz
- Membro da Banca -

Mestrando Sergie Josue Peña Cruz
- Membro da Banca -

Dedico este trabalho à …

AGRADECIMENTOS

Agradeço a Deus, que me sustentou e renovou minhas forças para que pudesse persistir e vencer os obstáculos encontrados ao longo desta jornada.

Dedico este trabalho e minha gratidão aos meus pais, Amelia e Carlos. Obrigado por acreditarem em mim desde o começo, por me aconselharem e por me proporcionarem a oportunidade de me dedicar aos estudos sem preocupações. O amor e o apoio de vocês serviram como motivação para que eu continuasse lutando pelos meus objetivos.

Ao meu orientador, Jonatas, agradeço pela parceria, pela dedicação e pelas instruções durante a elaboração deste trabalho. Seu suporte foi indispensável para a conclusão desta etapa.

Estendo meus agradecimentos à UNESP e a todos os professores do curso. Sou grato pelo conhecimento transmitido e pelo auxílio prestado durante minha graduação, fundamentais para o meu amadurecimento acadêmico e para a construção da minha carreira.

“As universidades serão o que são suas bibliotecas” (Gelfand, 1968, p. 19, tradução nossa).

RESUMO

A alocação ótima de chaves seccionadoras é fundamental no planejamento de redes de distribuição, visando aumentar a confiabilidade do sistema e a qualidade da energia entregue. A reconfiguração da rede, habilitada por estas chaves, caracteriza um problema de otimização complexo, modelado como programação não-linear inteira mista. Para redes extensas, o emprego de metaheurísticas, como o algoritmo genético (AG), se mostra uma solução eficiente. O sucesso da implementação de um AG, no entanto, exige uma análise criteriosa dos fatores que afetam seu desempenho, com foco especial na forma de representação (codificação) das possíveis soluções. Este trabalho tem como objetivo central implementar uma análise crítica da técnica de codificação do AG para o problema de alocação/realocação de chaves, buscando minimizar o custo de interrupção ao consumidor. O estudo detalha o modelo matemático e algoritmo, e compara duas abordagens: a codificação binária; e uma codificação decimal (inteira). São avaliados os impactos de cada método na robustez, no desempenho computacional e na estrutura dos operadores genéticos, dado um conjunto predeterminado de ramais candidatos e um número fixo de chaves. A validação é realizada com testes em alimentadores típicos da literatura.

Palavras-chave: algoritmo genético; chaves seccionadoras; custo de interrupção ao consumidor; metaheurística; otimização combinatória; sistemas de distribuição.

ABSTRACT

The optimal allocation of disconnector switches is fundamental in distribution network planning, aiming to increase system reliability and the quality of delivered energy. Network reconfiguration, enabled by these switches, characterizes a complex optimization problem, modeled as mixed-integer non-linear programming. For large-scale networks, the use of metaheuristics, such as the genetic algorithm (GA), proves to be an efficient solution. However, the successful implementation of a GA requires a careful analysis of the factors affecting its performance, with special focus on the form of representation (encoding) of possible solutions. The central objective of this work is to implement a critical analysis of the GA encoding technique for the switch allocation/reallocation problem, seeking to minimize the customer interruption cost. The study details the algorithmic model and compares two approaches: binary encoding and decimal (integer) encoding. The impacts of each method on robustness, computational performance, and the structure of genetic operators are evaluated, given a predetermined set of candidate branches and a fixed number of switches. Validation is performed through tests on typical feeders found in the literature.

Keywords: genetic algorithm; customer interruption cost; metaheuristic; combinatorial optimization; distribution systems.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

PNLIM	Programação Não Linear Inteira Mista
AG	Algoritmo Genético
CIC	Custo de Interrupção ao Consumidor
AE	Algoritmos Evolucionários

SUMÁRIO

1 INTRODUÇÃO

A garantia de um fornecimento de energia elétrica estável e de qualidade é um pilar da sociedade moderna, e a eficiência da rede de distribuição é um de seus componentes mais críticos. A confiabilidade do sistema, ou seja, a sua capacidade de minimizar interrupções, é um dos principais desafios de engenharia no planeamento e operação destas redes. Estima-se que a maioria das falhas percebidas pelo consumidor, frequentemente superando 70% da duração total das interrupções, tem origem em contingências ao nível da distribuição (Billinton & Allan, 1996). Em caso de falha, seções da rede são isoladas da subestação, e equipes de manutenção são mobilizadas para localizar e reparar o defeito, um processo que inevitavelmente deixa os consumidores a jusante sem energia, impactando negativamente os índices de confiabilidade.

É neste contexto que a instalação de chaves seccionadoras na rede assume um papel importante. Estes dispositivos permitem a divisão da rede, tornando possível isolar rapidamente a seção afetada e reconfigurar o sistema para restaurar o serviço ao maior número possível de consumidores no menor tempo possível. Em redes já em operação, surge o problema da realocação destes dispositivos, onde se estuda a modificação da sua posição física para otimizar a fiabilidade geral do sistema. É importante diferenciar que a realocação altera o local físico das chaves, enquanto a reconfiguração é o ato de manobrar (abrir ou fechar) das mesmas. O enfoque deste trabalho é o problema da realocação ótimo.

A realocação de chaves visa, portanto, determinar a configuração superior de instalação, com base num conjunto predefinido de ramais candidatos e um número fixo de chaves. O objetivo último é minimizar as interrupções de serviço na decorrência de falhas na seção. Esta formulação configura um clássico problema de otimização combinatória. Para determinar a configuração ideal, o modelo deve considerar múltiplos parâmetros, como as potências das cargas, os índices de falha e os tipos de consumidores em cada seção. A complexidade matemática resultante caracteriza o problema como um modelo de programação não linear inteiro misto (PNLIM).

A natureza não linear desse problema torna a sua resolução um desafio significativo. O espaço de busca é composto por todas as combinações possíveis de locais para as chaves, e o seu domínio pode crescer exponencialmente com o tamanho da rede. Consequentemente, métodos clássicos de otimização revelam-se frequentemente inadequados ou computacionalmente ineficientes para encontrar uma solução viável num tempo de processamento razoável.

Esta dificuldade justifica o uso de técnicas de busca adaptadas, conhecidas como metaheurísticas. Estes métodos, como o algoritmo genético (AG), são fundamentais por permitirem explorar o vasto espaço de soluções de forma inteligente, proporcionando soluções de alta qualidade em tempos de execução viáveis. O AG, em particular, é uma técnica robusta e amplamente testada, inspirada nos princípios da evolução biológica e seleção natural neodarwiniana.

Contudo, a simples aplicação de um AG não é suficiente. A sua eficácia está diretamente ligada às decisões de implementação. Para além da calibração de parâmetros de controle (como tamanho da população, taxas de mutação e recombinação), a decisão de design mais impactante é o sistema de codificação, ou seja, a forma como uma configuração de chaves é representada computacionalmente. Esta escolha define a estrutura do espaço de busca e condiciona toda a lógica dos operadores genéticos.

Para o problema em estudo, duas abordagens de codificação são apresentadas:

1. Codificação Binária: A representação tradicional, onde o cromossomo é um vetor de bits (0 ou 1) com um comprimento igual ao número total de posições candidatos da rede. Um '1' indica a presença de uma chave nessa posição, '0' a sua ausência. Embora simples, esta abordagem colide com a restrição de "número fixo de chaves", onde operadores genéticos podem gerar soluções-filhas com um número incorreto de chaves. Isto exige a implementação de operadores lógicos complexos que garantam a manutenção do número de chaves, o que pode adicionar sobrecarga computacional e complexidade ao algoritmo;

2. Codificação Decimal (Inteira): A alternativa analisada neste trabalho. O cromossomo tem um tamanho fixo, igual ao número de chaves a alocar (ex: 10 posições). Cada gene (posição no vetor) armazena o número de identificação do ramal que recebe a chave. Esta representação resolve o problema do "número fixo". Contudo, introduz um novo desafio: a restrição de unicidade, pois não podem existir duas chaves na mesma posição (genes duplicados). Tal como a abordagem binária, esta requer operadores genéticos adaptados (ex: baseados em permutação ou com "funções de reparo") para garantir a validade das soluções.

1.1 OBJETIVOS

Fundamentado na formulação do problema e na metodologia da codificação, este trabalho tem como seu objetivo a implementação e a análise crítica comparativa de duas abordagens (binária e decimal) do AG para a solução do problema de alocação/realocação de chaves seccionadoras, visando a minimização do custo de interrupção ao consumidor (CIC).

Para alcançar este propósito, será primeiramente detalhada a formulação matemática do problema, que estabelece o CIC como a função objetivo a ser otimizada. Em seguida, serão desenvolvidos e implementados os dois métodos de AG, demonstrando as diferenças entre a codificação binária tradicional com a decimal (inteira), o que necessitará a adaptação dos seus respectivos operadores genéticos para lidar com as restrições específicas de cada representação.

O desfecho do trabalho consiste na execução de um conjunto de testes computacionais em um alimentador de distribuição de referência. Esta análise de desempenho permite uma avaliação crítica e comparativa das duas metodologias, focando no impacto de cada codificação na qualidade final da solução, na velocidade de convergência, na robustez dos resultados e no tempo de execução, de modo a determinar a abordagem mais eficiente para o problema em estudo.

1.2 ESTRUTURA DO TRABALHO DE GRADUAÇÃO

Este documento está organizado da seguinte forma:

No Capítulo 1, são apresentadas a introdução e a motivação do trabalho, destacando a relevância da alocação de chaves, a complexidade do problema e a justificativa para a análise comparativa das técnicas de codificação do AG. Apresentam-se também os objetivos do projeto.

No Capítulo 2 é detalhada a formulação matemática do problema, com ênfase no cálculo do CIC, que serve como função objetivo para a otimização.

No Capítulo 3 são apresentados os fundamentos teóricos do AG, e, de forma central, a metodologia de implementação das duas arquiteturas de codificação (binária e decimal) e dos seus respetivos operadores genéticos.

No Capítulo 4 são descritos os procedimentos do estudo de caso, incluindo a caracterização do alimentador de testes (rede de 136 barras), os parâmetros de custo e a metodologia de configuração dos testes comparativos.

No Capítulo 5 são apresentados e analisados os resultados obtidos, utilizando gráficos e tabelas para comparar de forma direta a qualidade da solução e o desempenho computacional de ambas as codificações.

No Capítulo 6 são apresentadas as considerações finais e as conclusões do trabalho, sintetizando os achados e respondendo à questão central de pesquisa sobre a eficiência relativa de cada método de codificação para o problema em estudo.

2 DESENVOLVIMENTO

A alocação, ou realocação, de chaves seccionadoras é um problema central no planejamento e operação de redes de distribuição elétrica. Ele consiste em determinar os locais ótimos para a instalação de um número fixo de dispositivos, escolhidos a partir de um conjunto preestabelecido de possíveis posições. Esta decisão é considerada um problema de otimização combinatória, pois cada configuração possível de chaves redefine a segmentação da rede, alterando diretamente a confiabilidade. O objetivo principal é definir uma configuração que, na ocasião de faltas permanentes, permita a reconfiguração do sistema para que este opere em estado restaurativo, minimizando assim os danos causados aos consumidores pela interrupção do serviço.

Para quantificar esse prejuízo de forma objetiva, utiliza-se o CIC. Este parâmetro é a métrica fundamental para validar os impactos econômicos e sociais de uma interrupção no fornecimento. O CIC é um indicador que converte o prejuízo estimado para diferentes classes de consumidores (residenciais, comerciais e industriais) em um valor monetário. Desta forma, ele permite que o objetivo abstrato de "aumentar a confiabilidade" se transforme em um caso de otimização com base em um parâmetro calculável, de forma a obter o menor custo possível ao consumidor.

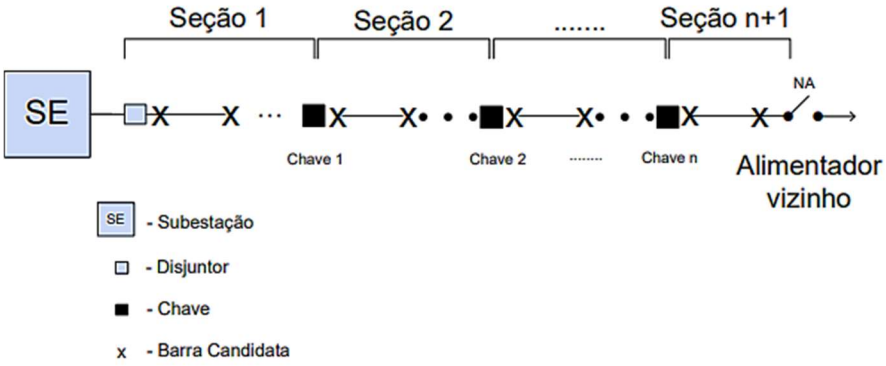
Para a determinação do CIC são utilizados dados históricos e estatísticos detalhados do alimentador para a realização do cálculo. É necessário considerar o registro de taxas de falha (frequência), os tempos médios para diagnóstico (identificação da falha), os tempos de manobra para isolamento da área afetada e os tempos de reparo para o restabelecimento do serviço. A combinação desses fatores permite estimar o custo associado à duração da interrupção para cada seção da rede.

O ponto chave deste modelo é o conceito de “seção”. Uma seção é definida como um trecho da rede elétrica delimitado por dispositivos de seccionamento, como disjuntores ou as próprias chaves seccionadoras. A alocação de uma nova chave em um ramo candidato divide uma seção existente em duas novas seções, alterando comprimentos, cargas e taxas de falha de cada trecho.

Para o modelo de otimização, cada seção é tratada como uma unidade de análise, considera-se que cada seção possui uma determinada carga e tipos específicos de clientes. Isso é essencial, pois o impacto econômico de uma interrupção em uma seção majoritariamente industrial é diferente do impacto em uma seção residencial ou comercial.

Para ilustrar o modelo matemático utilizado para o problema de alocação/realocação de chaves em alimentadores da rede de distribuição, apresenta-se o alimentador genérico de uma rede de distribuição na Figura 1 a seguir.

Figura 1 – Configuração de um alimentador típico de distribuição



Fonte: Leite & Mantovani (2012).

Na Figura 1, ilustram-se os elementos-chave do problema: a Subestação (SE), o Disjuntor (que define o início da Seção 1), as Chaves (que definem o início das seções subsequentes), e as barras candidatas (x), que são os possíveis pontos para alocação ou realocação.

A formulação matemática do CIC inicia-se com a análise de contingências. Utilizando os dados históricos do alimentador, como o tempo necessário para detecção da falha, isolamento da seção e restauração do serviço, modela-se o impacto de uma falha.

Quando uma falta permanente ocorre na seção *i*, ela causa uma interrupção. O custo dessa interrupção para os consumidores localizados na seção *j* (que pode ser

a própria seção *i* ou outra seção a jusante que também foi afetada) depende do tipo de consumidor e da duração da falha.

O custo de interrupção por kilowatt (kW) da seção *j*, devido a uma falha ocorrida na seção *i*, é expresso por (1):

$$C_{ij} = \left(Res_j(\%) \cdot fr(\gamma_{ij}) + Com_j(\%) \cdot fc(\gamma_{ij}) + Ind_j(\%) \cdot fi(\gamma_{ij})\right) \quad (1)$$

Em que:

- $\mathrm{Res}_j(\%), \mathrm{Com}_j(\%), \mathrm{Ind}_j(\%)$: porcentagens das cargas residenciais, comerciais e industriais na seção *j*;
- $fr(\gamma_{ij})$, $fc(\gamma_{ij})$, $fi(\gamma_{ij})$: funções de custo que convertem o tempo de interrupção γ_{ij}no custo monetário para cada tipo de carga.

A Equação (1) fornece o custo por kW de uma contingência específica. Para obter o custo total anual, considera se o custo de todas as falhas possíveis, ponderando pela taxa de falha e pela carga afetada.

A função objetivo que representa o CIC para o alimentador analisado é dada por (2):

$$\min CIC = \sum_{i=1}^{n+1} \lambda_i \, l_i \sum_{j=1}^{n+1} C_{ij} L_j \quad (2)$$

Onde:

- *i*: índice da seção onde a falta ocorre.
- *j*: índice da seção cuja carga é interrompida.
- *n*+1: número total de seções.
- λ_i: taxa de falhas permanentes da seção *i*.
- l_i: comprimento da seção *i* em km.
- C_{ij}: custo de interrupção unitário.
- L_j: carga total da seção *j*.

Essa função objetivo é o fator que o AG busca minimizar. Cada “indivíduo” representa uma proposta de alocação de chaves. Para avaliar o indivíduo, o AG deve:

1. Recalcular a divisão das seções com base na posição das chaves;
2. Redistribuir comprimentos (l_i) e cargas (L_j);
3. Calcular o CIC total usando (2).

O indivíduo com menor CIC será considerado o mais apto. A complexidade se encontra no fato de a função ser não linear e o espaço de busca possuir natureza combinatória e extensa.

3 TÉCNICA DE SOLUÇÃO

A natureza do problema de alocação de chaves, caracterizado como um modelo de PNLIM viabiliza o uso de técnicas metaheurísticas, para encontrar soluções de qualidade em um tempo razoável.

Este trabalho aborda a implementação e análise crítica do AG, uma técnica específica dentro da classe mais ampla dos algoritmos evolucionários (AE). A eficiência desta abordagem para o problema em estudo será avaliada em detalhes, particularmente em como as diferentes metodologias de codificação impactam no desempenho e na robustez do algoritmo.

3.1 ALGORITMO EVOLUCIONÁRIO

O termo AE, é aplicado a um conjunto de técnicas inspiradas na teoria neodarwiniana da evolução natural na biologia. Apesar do modelo padrão dos algoritmos genéticos, atualmente torna-se cada vez mais difícil diferenciar os mesmos, e há uma tendência no uso do termo mais amplo "algoritmos evolucionários (AE)" para se referir a qualquer técnica baseada no princípio de seleção natural (ou sobrevivência do mais apto) originalmente definida por Charles Darwin.

Em termos gerais, para simular um processo evolutivo em um computador, precisamos do seguinte (Coello et al., 2002):

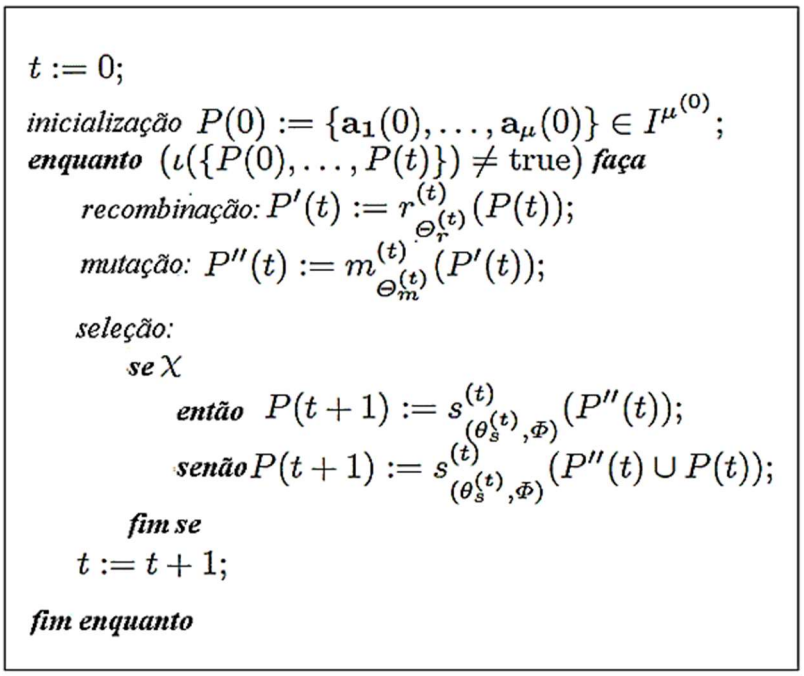
- Uma representação de soluções potenciais para o problema;
- Uma maneira de criar uma população inicial de soluções potenciais;
- Uma função de avaliação que desempenha o papel do meio ambiente e, soluções de elite em termos de sua "adequação";
- Operadores genéticos que alteram a composição da prole gerada, normalmente, recombinação e mutação;
- Valores para vários parâmetros que o algoritmo evolutivo usa (tamanho da população, probabilidades de aplicação de operadores genéticos, etc.).

Para definir um AE, seu modelo básico é descrito em termos matemáticos. Nesta estrutura, cada AE está associado a um conjunto não vazio I de espaço dos indivíduos. Cada indivíduo $a \in I$ normalmente representa uma solução candidata para o problema que está sendo resolvido. Os indivíduos são representados como um vetor (a), onde as dimensões do vetor são similares aos genes de um cromossomo.

Ao definir transformações populacionais, Bäck (1996) determina-se a coleção resultante de μ indivíduos via I^{μ}, e denota-se transformações populacionais pela seguinte relação: $T: I^{\mu} \rightarrow I^{\mu}$, onde $\mu \in \mathbb{N}$. No entanto, esta estrutura geral representa uma transformação populacional por meio da relação $T: I^{\mu} \rightarrow I^{\mu'}$, indicando que populações sucessivas podem conter o mesmo ou diferentes números de indivíduos. Dessa forma, um AE é definido como (Coello et al., 2002):

Seja I um conjunto não vazio do espaço de indivíduos, $\{\mu^{(i)}\}_{i\in\mathbb{N}}$ uma sequência em $\mathbb{Z}^{+}$ dos tamanhos da população pai, $\{\mu'(i)\}_{i\in\mathbb{N}}$ a sequência em $\mathbb{Z}^{+}$ dos tamanhos da população descendente, $\Phi: I \rightarrow \mathbb{R}$ uma função de aptidão, $l: \bigcup_{i=1}^{\infty}(I^{\mu})^{(i)} \rightarrow$ {verdadeiro, falso} o critério de parada, $\chi \in$ {verdadeiro, falso}, r uma sequência $\{r^{(i)}\}$ de operadores de recombinação $r^{(i)}: \mathbb{X}_r^{(i)} \rightarrow \mathcal{T}(\Omega_r^{(i)}, \mathcal{T}(I^{\mu^{(i)}, I^{\mu'^{(i)}}}))$, m uma sequência $\{m^{(i)}\}$ dos operadores de mutação $m^{(i)}: \mathbb{X}_m^{(i)} \rightarrow \mathcal{T}(\Omega_m^{(i)}, \mathcal{T}(I^{\mu'^{(i)}}, I^{\mu'^{(i)}}))$, s uma sequência $\{s^{(i)}\}$ dos operadores de seleção $S^{(i)}: \mathbb{X}_s^{(i)} \times \mathcal{T}(I, \mathbb{R}) \rightarrow \mathcal{T}(\Omega_s^{(i)}, \mathcal{T}(I^{\mu'^{(i)}} + \chi\mu^{(i)}, I^{\mu^{(i+1)}}))$, $\theta_r^{(i)} \in \mathbb{X}_r^{(i)}$ os parâmetros de recombinação, $\theta_m^{(i)} \in \mathbb{X}_m^{(i)}$ os parâmetros de mutação e $\theta_s^{(i)} \in \mathbb{X}_s^{(i)}$ os parâmetros de seleção. Então, o algoritmo mostrado na Figura 2 é chamado de AE.

Figura 2 – Pseudocódigo de um algoritmo evolucionário



Fonte: Leite & Mantovani (2012).

3.2 ALGORITMO GENÉTICO

Os AGs representam uma família de técnicas dentro da classe dos AEs, inspirando-se diretamente nos princípios da seleção natural e da genética. Eles possuem como diferença principal, os operadores genéticos específicos, a recombinação (*crossover*) e a mutação, aplicados a uma população de soluções. O AG é, por natureza, um método probabilístico que busca uma solução aproximada do valor ideal, adequada aos critérios esperados para o problema em análise.

3.3 A ANÁLISE CRÍTICA DA CODIFICAÇÃO

Conforme estabelecido nos objetivos deste trabalho, um dos pontos mais relevantes que influenciam o desempenho e a robustez da metaheurística é a codificação das possíveis soluções. Esta é a escolha mais impactante na concepção de um AG. A "codificação" refere-se a como uma solução do mundo real (o "fenótipo", por exemplo: "chaves alocadas nos ramais 5, 12 e 30") é representada computacionalmente (o "genótipo", ou cromossomo).

O problema de alocação de chaves possui uma restrição que torna essa escolha crítica. Dentro de um conjunto de N ramais candidatos, deve-se escolher exatamente K locais para instalar as chaves (onde K é o número fixo de chaves de manobra). A forma como o cromossomo representa esta alocação de K chaves impacta diretamente todo o algoritmo.

1. A estrutura do espaço de busca: A representação define o "mapa" que o algoritmo irá explorar.

2. A lógica dos operadores genéticos: A forma do cromossomo descreve como a recombinação e a mutação devem funcionar. Uma codificação inadequada pode exigir operadores complexos e computacionalmente caros para garantir que os "filhos" (novas soluções) sejam válidos, tornando a resolução ineficiente.

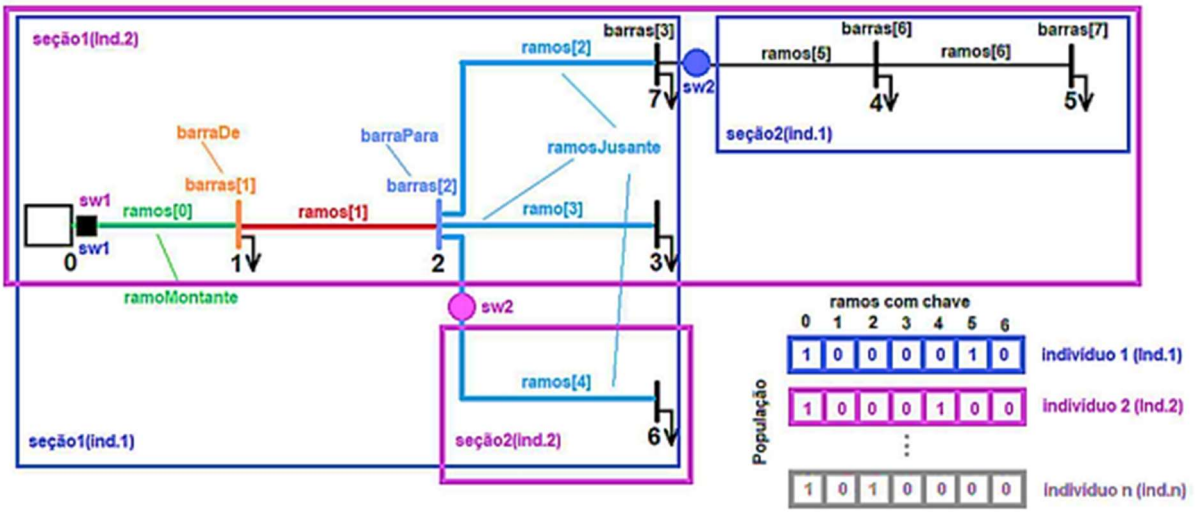
Este trabalho implementa e analisa criticamente duas metodologias de codificação para este problema. Uma abordagem binária e uma abordagem decimal (inteira), comparando seus méritos e dificuldades.

3.4 METODOLOGIA 1: ARQUITETURA DE CODIFICAÇÃO BINÁRIA

A primeira arquitetura para problemas de seleção e alocação é a representação binária.

1. Cromossomo (Codificação): Cada indivíduo é um vetor de bits. A dimensão (comprimento) deste vetor é N, onde N é o número total de ramos candidatos na rede. Se a rede possui 136 ramos candidatos, o cromossomo terá 136 posições;
2. Gene: Cada gene no cromossomo corresponde a um ramo candidato específico. Um valor '1' na posição i significa que uma chave está alocada no ramo i; um valor '0' significa que não está. A figura 3 ilustra essa representação.

Figura 3 – Representação dos indivíduos/codificação cromossômica do AG.



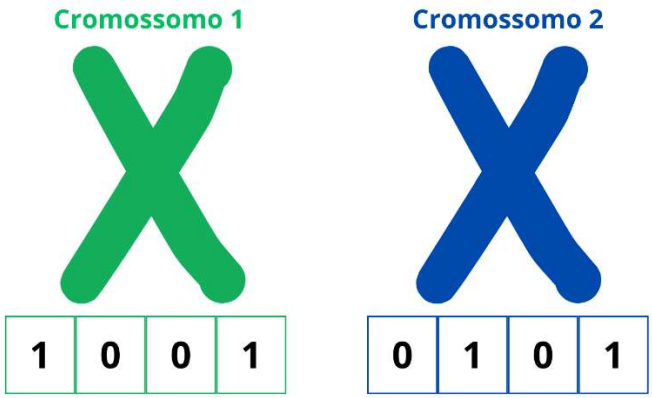
Fonte: Kheirkha (2024).

Um dos desafios desta codificação é a restrição de K chaves (o número fixo de chaves). O espaço de busca teórico é de 2^N, mas apenas uma porção dessas soluções é válida (aquelas com exatamente K bits '1'). A população inicial não pode ser gerada de forma totalmente aleatória; ela deve ser construída de forma que cada indivíduo (cromossomo) contenha exatamente K bits de valor '1'.

3. Operadores Adaptados: Os operadores genéticos padrão (como *crossover* de um ponto ou mutação de *bit flip*) não podem ser usados diretamente, pois gerariam indivíduos inviáveis (com $K-1$ ou $K+1$ chaves). Portanto, os operadores devem ser adaptados:

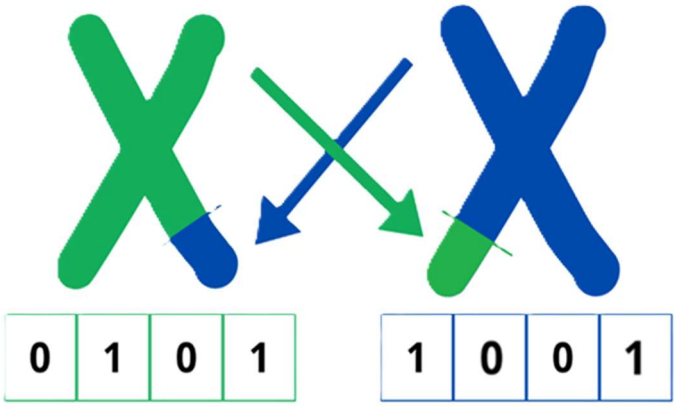
4. Recombinação: A recombinação é adaptada para preservar o número de chaves. Conforme ilustrado nas Figuras 4 e 5, o processo seleciona um "ponto de 1" (uma chave) aleatório em cada pai e troca os segmentos de genes a partir daquela chave.

Figura 4 – Processo de Recombinação (antes)



Fonte: Próprio autor.

Figura 5 – Processo de Recombinação (depois)

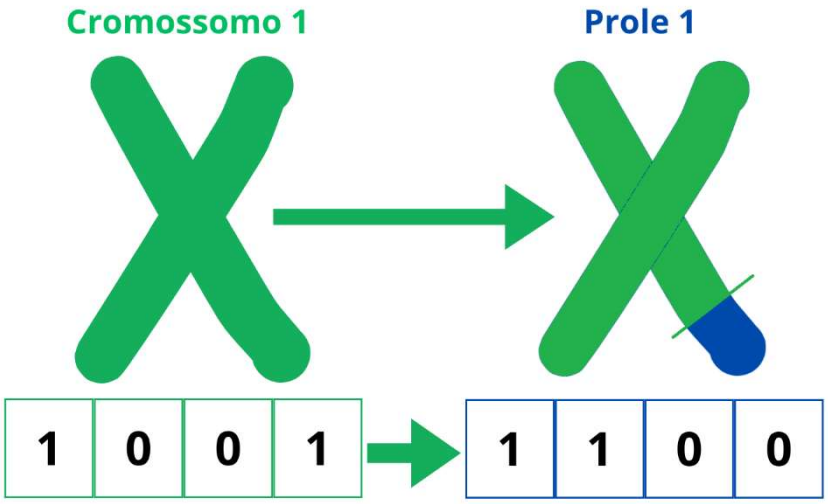


Fonte: Próprio autor.

A figura 5 mostra os "Filhos" (Prole) resultantes da troca. Note que o número de '1's pode não ser preservado, exigindo um mecanismo de seleção que escolha o filho com o fitness melhor e que tenha o número correto de chaves.

5. Mutação: A mutação não pode ser um simples "bit flip". Ela deve ser implementada como uma "troca". O algoritmo seleciona aleatoriamente um gene '1' e o altera para '0', e, simultaneamente, seleciona aleatoriamente um gene '0' e o altera para '1'. Este processo preserva a quantidade de chaves no indivíduo, mantendo sua validade.

Figura 6 – Processo de mutação



Fonte: Próprio autor.

Nesta metodologia, o cromossomo é de tamanho N, a maioria dos genes é '0', e seus operadores exigem uma lógica de preservação da contagem para manter a validade das soluções.

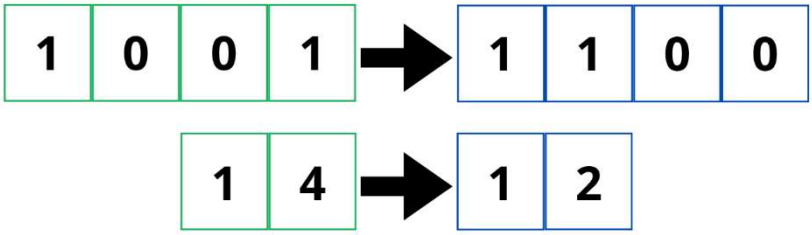
3.5 METODOLOGIA 2: ARQUITETURA DE CODIFICAÇÃO INTEIRA

A segunda arquitetura, proposta para análise comparativa neste trabalho, utiliza uma representação decimal (ou inteira) que aborda o problema de forma mais direta e natural.

- Cromossomo (Codificação): Cada indivíduo é um vetor de números inteiros. A dimensão (comprimento) deste vetor é K, onde K é o número fixo de chaves a serem alocadas. Se $K = 2$ chaves, o cromossomo deve ter 2 posições.

- Gene: O valor de cada gene no cromossomo representa o número decimal (inteiro) do ramo candidato onde a chave está alocada. Por exemplo, em uma rede de distribuição de 4 posições candidatas, se $K = 2$, um cromossomo seria um vetor de 2 posições, como [1, 4], ilustrado na Figura 7.

Figura 7 – Posicionamento de chaves para binário e inteiro.



Fonte: Próprio autor.

- A restrição de K chaves é satisfeita pela própria estrutura do cromossomo (que tem tamanho K). O algoritmo não precisa de lógica extra para contar chaves. O espaço de busca é diretamente o número de combinações de N, que é exatamente o espaço do problema.

- O novo desafio desta codificação é a unicidade. Os operadores não podem gerar um indivíduo com o identificador (ID) de ramo repetido (ex: [5, 12, 30, **41**, 55, **41**, ...]), pois não é possível alocar duas chaves no mesmo local.

Operadores Adaptados:

1. População Inicial: A geração da população inicial é simples: para cada indivíduo, o algoritmo sorteia K IDs de ramos únicos do conjunto de N ramos candidatos.

2. Recombinação: Operadores de *crossover* padrão podem gerar filhos com genes duplicados. Para solucionar isso, deve-se implementar uma função de reparo. Após o *crossover*, esta função verifica se há IDs duplicados no cromossomo filho e se houver, o gene duplicado é substituído por um novo ID de ramo aleatório que ainda não esteja presente no cromossomo, repetindo o processo até que a unicidade seja garantida.

3. Mutação: A mutação consiste em selecionar aleatoriamente um gene (uma posição no vetor de tamanho K) e substituir seu valor (o ID do ramo) por um novo

ID de ramo aleatório, garantindo apenas que este novo ID não exista nas outras posições.

Nesta metodologia, o cromossomo é reduzido (tamanho K), e seus operadores exigem uma lógica para preservar a unicidade.

3.6 COMPONENTES UNIFICADOS PARA COMPARAÇÃO

Para que a análise comparativa entre as duas metodologias de codificação (binária e decimal) seja confiável e válida, todos os outros componentes do AG devem ser mantidos idênticos para ambas as implementações.

3.6.1 Função de Avaliação (*Fitness*)

Ambas as abordagens compartilharão a mesma função objetivo: o CIC, conforme definido por (2). Como o AG é um algoritmo de maximização e o objetivo do problema é *minimizar* o CIC, o valor do CIC deve ser convertido em uma função de adequação (*fitness*). Um valor de *fitness* maior corresponde a um CIC menor, ou seja, uma solução melhor. Isso é feito através de uma inversão, como em (3).

$$Fitness = 100000/CIC \tag{3}$$

3.6.2 Seleção

O mecanismo de seleção, que escolhe quais indivíduos da população atual são mais aptos a gerar a próxima, será o mesmo para ambas as arquiteturas. Como o objetivo é selecionar os indivíduos com menor CIC (maior *fitness*), uma estratégia robusta como a seleção por roleta será utilizada, onde a probabilidade de um indivíduo ser selecionado é proporcional ao seu fitness.

3.6.3 Critério de Parada

O processo do AG termina quando uma condição de parada é atingida. Este critério será o mesmo para ambos os testes. Múltiplas condições podem ser usadas,

mas a mais comum, que será usada como base, é o número máximo de gerações. Atingido este número de iterações, o algoritmo para e retorna a melhor solução encontrada até aquele momento.

3.7 DETERMINAÇÃO DOS PARÂMETROS DE CONTROLE

Além da arquitetura de codificação, o desempenho do AG é fortemente influenciado pela calibração de seus parâmetros de controle: Tamanho da População (TP), Quantidade de Gerações (NG), Taxa de Recombinação (TR) e Taxa de Mutação (TM).

Para determinar esses parâmetros, pode-se utilizar um método heurístico, cujo fluxograma está ilustrado na Figura 8. Este método ajusta os parâmetros (TM, TR, NG, TP) com base em critérios de convergência e limites pré-estabelecidos.

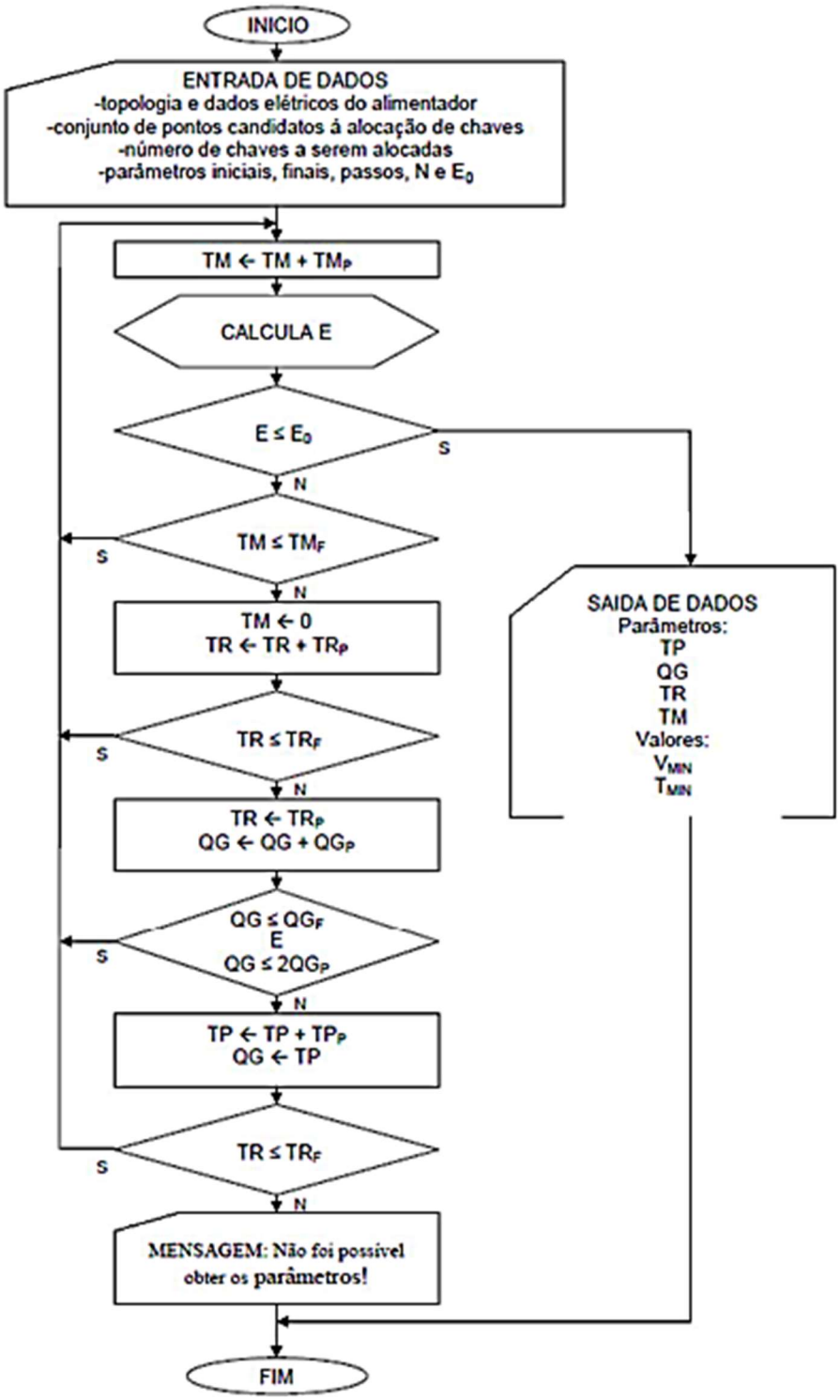
- Tamanho da População (TP): Define a diversidade da busca. Um TP pequeno converge rápido, mas pode ficar preso em ótimos locais. Um TP grande explora mais o espaço, mas é computacionalmente mais lento.

- Quantidade de Gerações (NG): Define a quantidade de gerações.

- Taxa de Recombinação (TR): Define a probabilidade de dois indivíduos selecionados "pais" gerarem "filhos", promovendo a combinação de boas características.

- Taxa de Mutação (TM): Define a probabilidade de uma mudança aleatória em um gene, injetando diversidade e promovendo a *exploração* de novas áreas do espaço de busca.

Figura 8 – Fluxograma do programa utilizado para obter os parâmetros



Fonte: Leite & Mantovani (2012).

A Tabela 1 ilustra um exemplo de calibração de parâmetros, onde os valores ideais podem variar dependendo da complexidade do problema (representada por C_i, o número de chaves).

Tabela 1 - Exemplo de valores de parâmetros do AG por quantidade de chaves.

Execução (i)	C_i	TP	NG	TR	TM
0	2	50	50	10	45
1	6	100	150	80	20
2	10	100	200	90	15
3	15	200	300	90	30
4	20	350	400	90	5

Fonte: Leite & Mantovani, 2012

Em que:

C_i*: Número de chaves;*

TP: Tamanho da População;

QG: Quantidade de Gerações;

TR: Taxa de Recombinação;

TM: Taxa de Mutação.

4 IMPLEMENTAÇÃO E METODOLOGIA COMPUTACIONAL

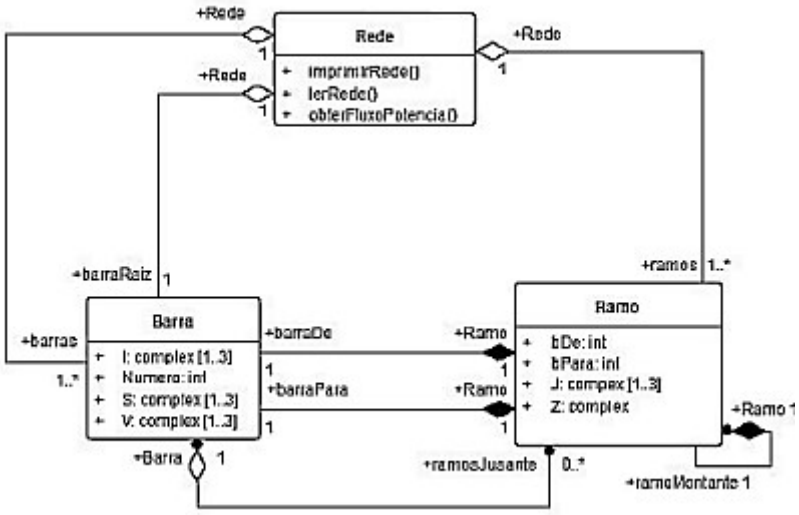
Este capítulo detalha a arquitetura computacional desenvolvida para implementar e comparar as duas metodologias de AG, binária e decimal (inteira), para o problema de alocação de chaves. Este trabalho expande a implementação do algoritmo binário realizado em outros trabalhos, desenvolvendo um segundo AG com codificação decimal, permitindo uma análise comparativa direta.

Ambas as metodologias foram desenvolvidas na linguagem de programação C# (C Sharp) no ambiente MS Visual Studio®, com o ambiente utilizado para a execução do código possuindo as seguintes configurações: Intel Core i5-9300H 2.40GHz (6 CPUs) e 8192MB RAM. Compartilhando uma base comum para a modelagem da rede e o cálculo da função objetivo.

4.1 ESTRUTURA DE DADOS E CONSTRUÇÃO DA REDE

O primeiro passo é a modelagem da rede em memória. A Figura 9 ilustra o diagrama de classes que define a topologia básica, mostrando como a classe Rede se relaciona com Barra e Ramo.

Figura 9 - Diagrama de classe de rede de distribuição



Fonte: Ferreira, 2021.

O Algoritmo 1 apresenta o pseudocódigo para o método *lerRede*(.). Esta função é responsável por ler um arquivo XML, determinar os objetos Barra e Ramo e conectar a topologia (definindo ramosJusante, ramoMontante, etc.).

Algoritmo 1 - Pseudocódigo do método *lerRede*(.)

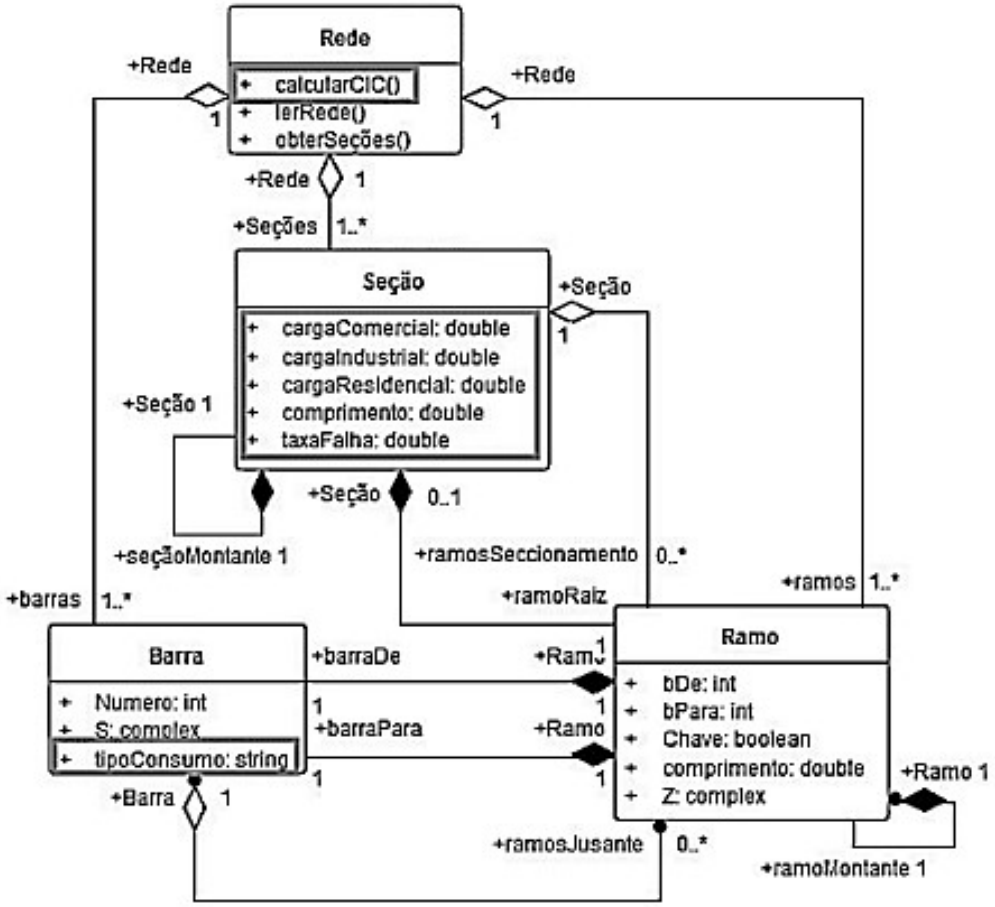
```
FUNÇÃO lerRede(arquivoXML):
    // 1. Carregar nós XML (Branch, Bus, TieSwitch)
    Nós_Barra = arquivoXML.SelecionarNós("Bus")
    Nós_Ramo = arquivoXML.SelecionarNós("Branch")

    // 2. Criar Barras
    PARA CADA nó_barra em Nós_Barra:
        b = NOVA Barra
        b.Numero = nó_barra.Atributo("id")
        b.Carga = nó_barra.Atributos("pa", "qa", etc.)
        b.TipoCarga = nó_barra.Atributo("type")
        Adicionar b na lista Rede.barras
    FIM-PARA

    // 3. Criar Ramos
    PARA CADA nó_ramo em Nós_Ramo:
        r = NOVO Ramo
        r.De = nó_ramo.Atributo("from")
        r.Para = nó_ramo.Atributo("to")
        r.Chave = nó_ramo.Atributo("switch") // (0 ou 1)
        r.Comprimento = nó_ramo.Atributo("length")
        Adicionar r na lista Rede.ramos
    FIM-PARA

    // 4. Conectar Topologia
    // (Loops para associar r.deBarra, r.paraBarra,
    // r.ramoMontante e barra.ramosJusante)
FIM-FUNÇÃO
```

Fonte: Próprio autor.

4.2 MAPEAMENTO TOPOLÓGICO E CÁLCULO DO CIC

Com a rede em memória, a base para o AG é a capacidade de avaliar qualquer configuração de chaves. A Figura 10 mostra o diagrama de classes para esta etapa, introduzindo a classe `Secao`.

Figura 10 - Diagrama de classe para obtenção das seções da rede



Fonte: Fernandes, 2023.

4.2.1 Obtenção das Seções

A cada reconfiguração proposta pelo AG, a divisão das seções muda. O Algoritmo 2 (*`ObterSecoes(.)`*) implementa a lógica de busca em profundidade (BEP) para mapear a rede.

Algoritmo 2 – Pseudocódigo do Método *ObterSecoes*(.) (dentro da classe Rede)

```
FUNÇÃO ObterSecoes():
    // 1. Identificar inícios de seção
    Rede.Secoes = NOVA Lista
    PARA CADA ramo EM Rede.ramos:
        SE ramo.Chave == 1 ENTÃO:
            s = NOVA Secao(ramoRaiz = ramo)
            Adicionar s em Rede.Secoes
        FIM-SE
    FIM-PARA

    // 2. Mapear cada seção
    PARA CADA secao EM Rede.Secoes:
        // 3. Chamar Busca em Profundidade (BEP)
        BEP(secao.ramoRaiz.paraBarra, secao)
    FIM-PARA

    // 4. Conectar seções (definir montante/jusante)
FIM-FUNÇÃO
// Função interna de Busca em Profundidade
FUNÇÃO BEP(barraAtual, secaoAtual):
    // 1. Acumular Cargas da Barra
    secaoAtual.cargaResidential += barraAtual.CargaResidencial
    // ... etc ...

    // 2. Percorrer ramos a jusante
    PARA CADA ramo EM barraAtual.ramosJusante:
        secaoAtual.comprimento += ramo.Comprimento
        SE ramo.Chave == 0 ENTÃO: // Ramo normal
            BEP(ramo.paraBarra, secaoAtual)
        SENÃO: // Encontrou outra chave
            secaoAtual.ramoSeccionamento.Adicionar(ramo)
        FIM-SE
    FIM-PARA
FIM-FUNÇÃO
```

Fonte: Próprio autor.

4.2.2 Cálculo do CIC

O Algoritmo 3 (*CalcularCIC*(.)) implementa a função objetivo (Equação (2) do Capítulo 2). Ele executa um loop duplo sobre todas as seções i e j, aplicando o custo de restauração (se $i == j$) ou o custo de chaveamento (se $i \neq j$) e ponderando pela carga, tipo de carga e comprimento da seção.

Algoritmo 3 - Pseudocódigo do método *CalcularCIC*(.)

```
FUNÇÃO CalcularCIC():
    cicTotal = 0
    gama = 0.025 // (taxa de falha)
    n = Rede.Secoes.Contagem

    PARA i = 0 ATÉ n-1: // (Seção i onde a falta ocorre)
        li = Rede.Secoes[i].comprimento / 1000.0 // (em km)
        PARA j = 0 ATÉ n-1: // (Seção j que sofre a interrupção)
            secaoJ = Rede.Secoes[j]

            // Define custos baseado na contingência             SE i == j
ENTÃO: // Custo de restauração
                custo_r = 7.0; custo_c = 200.0; custo_i = 30.0
            SENÃO: // Custo de chaveamento
                custo_r = 0.55; custo_c = 40.0; custo_i = 5.0
            FIM-SE

            Cij = (secaoJ.percResidencial * custo_r) + ...
            Lj = secaoJ.CargaTotal

            cicTotal += gama * li * Cij * Lj
        FIM-PARA
    FIM-PARA

    RETORNAR cicTotal
FIM-FUNÇÃO
```

Fonte: Próprio autor.

Algoritmo 4 - Pseudocódigo do cálculo de *Fitness* (Método decodifica)

```
FUNÇÃO decodifica(individuo):
    // 1. Aplicar cromossomo do indivíduo à rede
    // (Este passo varia entre binário e inteiro)
    AplicarCromossomoNaRede(individuo.cromossomo)

    // 2. Mapear a rede resultante
    Rede.ObterSecoes() // (Chama Algoritmo 2)

    // 3. Calcular CIC
    cic = Rede.CalcularCIC() // (Chama Algoritmo 3)

   4. Converter CIC em Fitness
    SE cic > 0 ENTÃO:
        individuo.fitness = 100000.0 / cic
    SENÃO:
        individuo.fitness = Infinito // (Valor máximo)
    FIM-SE
FIM-FUNÇÃO
```

Fonte: Próprio autor.

4.3 IMPLEMENTAÇÃO (METODOLOGIA 1: BINÁRIA)

Esta seção detalha a implementação do AG com codificação binária, que serve como comparativo.

4.3.1 Indivíduo Binário

O cromossomo é um vetor de booleanos de tamanho N(número total de ramos). A Figura 3 ilustra esta representação, onde cada bit "1" indica uma chave alocada. O Algoritmo 5 mostra como o cromossomo é aplicado na rede.

Algoritmo 5 - Pseudocódigo *AplicarCromossomoNaRede*(.) (Binário)

```
FUNÇÃO AplicarCromossomoNaRede(cromossomo_binario):
    PARA i = 0 ATÉ Rede.ramos.Comprimento-1:
        SE cromossomo_binario[i] == true ENTÃO:
            Rede.ramos[i].Chave = 1
        SENÃO:
            Rede.ramos[i].Chave = 0
        FIM-SE
    FIM-PARA
FIM-FUNÇÃO
```

Fonte: Próprio autor.

4.3.2 Operadores Genéticos (Binário)

A classe `AG_binario` contém os operadores adaptados para a codificação binária, que deve preservar o número de chaves (K).

O Algoritmo 6 mostra o trecho de código da geração inicial. Ele cria um `bool[tamanho_cromossomo]` e sorteia `nChaves` posições aleatórias (de 1 em diante) para receber `true`.

Algoritmo 6 - Pseudocódigo da População Inicial

```
FUNÇÃO GerarPopulaçãoInicial_Binaria(nP, N, K):
    // N = total de ramos, K = num. chaves
    PARA i = 1 ATÉ nP:
        individuo = NOVO Individuo(tamanho = N)
        individuo.cromossomo[0] = true // (Disjuntor é fixo)
        // Sorteia K-1 posições (além da 0)
        PARA g = 1 ATÉ K-1:
            PosChave = SortearAleatorio(1, N-1)
            // Lógica para evitar duplicatas (não mostrada no C#)
            individuo.cromossomo[PosChave] = true
        FIM-PARA
        Adicionar individuo na População
    FIM-PARA
FIM-FUNÇÃO
```

Fonte: Próprio autor.

O Algoritmo 7 detalha o operador *recombina*(.) binário, que é complexo pois não cruza o array `bool[]`, mas sim os índices das chaves ativas para preservar a contagem K.

Algoritmo 7 - Pseudocódigo do Operador de Recombinação (Classe AG_binario)

```
FUNÇÃO Recombinar_Binario(pai1, pai2):
    filho1, filho2 = NOVO Individuo
    pontoCorte = SortearAleatorio(1, K-1) // (K = num. chaves)
    indices_pai1 = EncontrarIndices(pai1.cromossomo, '1')
    indices_pai2 = EncontrarIndices(pai2.cromossomo, '1')

Criar filho 1
    Copiar genes (de 0 até pontoCorte) de indices_pai1 para filho1
    Copiar genes (de pontoCorte+1 até K) de indices_pai2 para filho1

    //Criar filho 2
    Copiar genes (de 0 até pontoCorte) de indices_pai2 para filho2
    Copiar genes (de pontoCorte+1 até K) de indices_pai1 para filho2

    Avaliar fitness(filho1) e fitness(filho2)
    RETORNAR (filho com maior fitness)

FIM-FUNÇÃO
```

Fonte: Próprio autor.

O Algoritmo 8 mostra o operador `muta`. Ele inverte um bit aleatório (podendo adicionar ou remover uma chave) e, em seguida, remove uma chave aleatória (definindo um k-ésimo '1' como `false`) para tentar manter a contagem de chaves.

Algoritmo 8 - Pseudocódigo do Operador de Mutação (Classe AG_binario)

```
FUNÇÃO Mutar_Binario(individuo):
    // 1. Inverte um bit aleatório (pode quebrar a restrição K)
    ponto1 = SortearAleatorio(1, N-1)
    individuo.cromossomo[ponto1] = NÃO individuo.cromossomo[ponto1]
```

```
    // 2. Remove um '1' aleatório (tenta corrigir a restrição K)
    pontoK = SortearAleatorio(1, K-1)
    indice_do_pontoK = Encontrar_Kesima_Chave(individuo.cromossomo, pontoK)

    SE indice_do_pontoK for válido ENTÃO:
        individuo.cromossomo[indice_do_pontoK] = false
    FIM-SE
FIM-FUNÇÃO
```

Fonte: Próprio autor.

4.4 IMPLEMENTAÇÃO (METODOLOGIA 2: DECIMAL/INTEIRA)

Esta seção detalha a nova implementação, foco deste trabalho, que usa a codificação decimal (inteira).

4.4.1 Indivíduo Inteiro

A classe `individuo_inteiro` (Algoritmo 9) define o cromossomo como um array de inteiros (`int[]`) de tamanho K (o `nChaves`). Cada gene armazena o identificador (ID) do ramo que possui a chave.

O Algoritmo 9 mostra como o método decodifica (Algoritmo 4) é implementado para esta codificação.

Algoritmo 9 - Pseudocódigo AplicarCromossomoNaRede (Inteiro)

```
FUNÇÃO AplicarCromossomoNaRede(cromossomo_inteiro):
    // 1. Resetar todas as chaves da rede
    PARA CADA ramo EM Rede.ramos:
        ramo.Chave = 0
    FIM-PARA

    // 2. Ativar disjuntor (ramo 0)
    Rede.ramos[0].Chave = 1
```

```
        // 3. Ativar chaves do cromossomo
        PARA CADA indiceRamo EM cromossomo_inteiro:
            SE (indiceRamo > 0 E indiceRamo < Rede.ramos.Comprimento)
            ENTÃO:
                Rede.ramos[indiceRamo].Chave = 1
            FIM-SE
        FIM-PARA
    FIM-FUNÇÃO
```

Fonte: Próprio autor.

4.4.2 Operadores Genéticos (Inteiro)

A geração de população (Algoritmo 10) é necessária para garantir a individualidade dos genes (sem duas chaves no mesmo ramo), um `HashSet` é usado para verificar se o índice de ramo sorteado (`PosChave`) já foi utilizado.

Algoritmo 10 - Pseudocódigo da População Inicial (Inteiro)

```
    FUNÇÃO GerarPopulaçãoInicial_Inteira(nP, N, K):
        // N = total de ramos, K = num. chaves
        PARA i = 1 ATÉ nP:
            individuo = NOVO Individuo(tamanho = K)
            genesUsados = NOVO HashSet
            PARA g = 0 ATÉ K-1:
                REPETIR
                    PosChave = SortearAleatorio(1, N-1)
                ATÉ QUE genesUsados NÃO CONTÉM PosChave

                individuo.cromossomo[g] = PosChave
                genesUsados.Adicionar(PosChave)
            FIM-PARA
            Adicionar individuo na População
        FIM-PARA
    FIM-FUNÇÃO
```

Fonte: Próprio autor.

O Algoritmo 11 mostra o operador recombina inteiro. Ele utiliza um *Crossover* de 1 Ponto padrão (Array.Copy), que é muito rápido, mas pode gerar filhos com genes duplicados.

Para corrigir isso, a função auxiliar *repararCromossomo*(.) é chamada. Esta função (também no Algoritmo 11) itera pelo cromossomo do filho e usa um HashSet para encontrar duplicatas. Se uma duplicata é encontrada, ela é substituída por um novo gene aleatório que ainda não esteja em uso.

Algoritmo 11 – Pseudocódigo da Recombinação e Reparo (Classe AG_inteiro)

```
FUNÇÃO Recombinar_Inteiro(pai1, pai2):
    // 1. Crossover de 1 Ponto
    pontoCorte = SortearAleatorio(1, K-1)
    filho1 = (Início de pai1) + (Fim de pai2)
    filho2 = (Início de pai2) + (Fim de pai1)

    // 2. Reparar filhos
    RepararCromossomo(filho1)
    RepararCromossomo(filho2)

    Avaliar fitness(filho1) e fitness(filho2)
    RETORNAR (filho com maior fitness)
FIM-FUNÇÃO

FUNÇÃO RepararCromossomo(individuo):
    genesUsados = NOVO HashSet
    PARA i = 0 ATÉ K-1:
        gene = individuo.cromossomo[i]

        // Se gene é duplicado OU inválido (ex: 0)
        SE (gene == 0 OU genesUsados CONTÉM gene) ENTÃO:
            REPETIR
                novoGene = SortearAleatorio(1, N-1)
            ATÉ QUE genesUsados NÃO CONTÉM novoGene
```

```
                individuo.cromossomo[i] = novoGene
                genesUsados.Adicionar(novoGene)
            SENÃO:
                genesUsados.Adicionar(gene)
            FIM-SE
        FIM-PARA
    FIM-FUNÇÃO
```

Fonte: Próprio autor.

O Algoritmo 12 implementa a mutação. Ele escolhe uma posição aleatória no cromossomo (`pontoMutacao`) e a substitui por um novo índice de ramo aleatório (`novoGene`), garantindo que o novo gene já não exista no cromossomo.

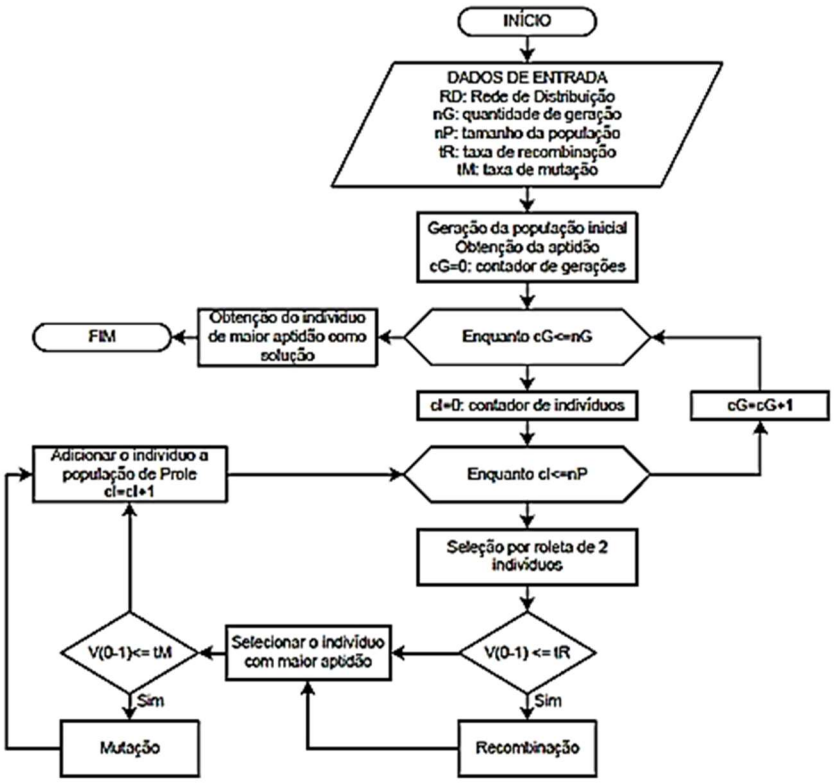
Algoritmo 12 - Pseudocódigo Operador de Mutação (Classe AG_inteiro)

```
    FUNÇÃO Mutar_Inteiro(individuo):
        // 1. Escolher posição a mutar
        pontoMutacao = SortearAleatorio(0, K-1)
        genesAtuais = NOVO HashSet(individuo.cromossomo)

        // 2. Encontrar novo gene válido
        REPETIR
            novoGene = SortearAleatorio(1, N-1)
        ATÉ QUE genesAtuais NÃO CONTÉM novoGene

        // 3. Aplicar mutação
        individuo.cromossomo[pontoMutacao] = novoGene
    FIM-FUNÇÃO
```

Fonte: Próprio autor.

O percurso principal de execução do AG, para ambas as metodologias, segue o fluxograma apresentado na Figura 11.

Figura 11 - Fluxograma para desenvolvimento do programa do AG.



Fonte: Leite & Mantovani (2012).

A implementação desse fluxo é descrita no Algoritmo 13, que representa o *metodoAg*(.). A única diferença entre as duas implementações (Binária e Inteira) são as funções específicas de *GerarPopulação*(.), *Recombinar*(.) e *Mutar*(.) que são chamadas, conforme detalhado nas seções 4.3 e 4.4.

Algoritmo 13 - Pseudocódigo do Loop Principal do AG (*metodoAg*(.))

```
// 1. Inicialização
SE Metodologia == BINARIA ENTÃO:
    População = GerarPopulaçãoInicial_Binaria(nP, N, K) // (Alg. 6)
SENÃO:
    População = GerarPopulaçãoInicial_Inteira(nP, N, K) // (Alg. 10)
FIM-SE

para cada individuo i em População
    i.fitness = CalcularFitness(i) // (Alg. 4)
melhorIndividuo = EncontrarMelhor(População)
```

```
      // 2. Loop de Gerações
      para g = 1 até nG
          NovaPopulação = []
          para j = 1 até nP
              // 3. Seleção (por Roleta, conforme código)
              pai1 = SelecionarPorRoleta(População)
              pai2 = SelecionarPorRoleta(População)

              // 4. Recombinação (Crossover)
              se (sortear() < tR)
                  se (Metodologia == BINARIA)
                      filho = Recombinar_Binario(pai1, pai2) // (Alg. 7)
                  senão // Metodologia == INTEIRA
                      filho = Recombinar_Inteiro(pai1, pai2) // (Alg. 11)
                  fim-se
              senão
                  filho = MelhorDe(pai1, pai2) // Elitismo
              fim-se

              // 5. Mutação
              se (sortear() < tM)
                  se (Metodologia == BINARIA)
                      Mutar_Binario(filho) // (Alg. 8)
                  senão // Metodologia == INTEIRA
                      Mutar_Inteiro(filho) // (Alg. 12)
                  fim-se

                  filho.fitness = CalcularFitness(filho) // Recalcula se
mutou
              fim-se

              // 6. Atualização
              NovaPopulação.adicionar(filho)
              se (filho.fitness > melhorIndividuo.fitness)
                  melhorIndividuo = filho
              fim-se
```

```
        fim-para

        População = NovaPopulação
        AtualizarFitnessTotal(População) // (para próxima roleta)
    fim-para

    Retornar melhorIndividuo
```

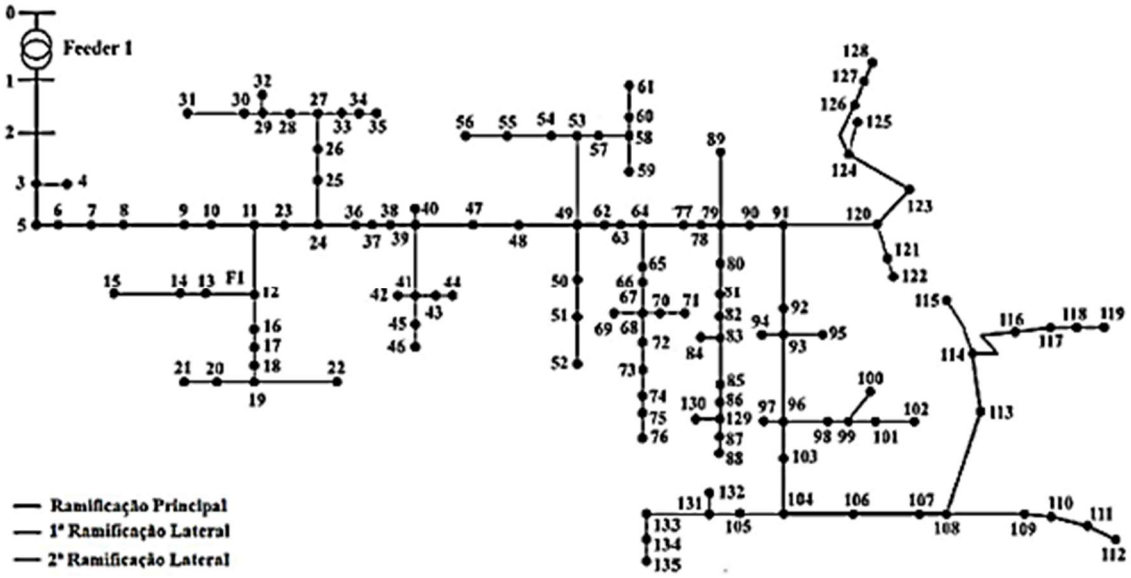
Fonte: Próprio autor.

5 RESULTADOS E DISCUSSÕES

Para validar e comparar as duas metodologias de codificação (binária e inteira) descritas no Capítulo 4, utiliza-se um estudo de caso baseado em uma rede de distribuição real. A rede elétrica em análise está descrita em um arquivo *.xml seguindo a estrutura padrão da Figura 9.

Na Figura 12, representa-se a topologia da rede de distribuição sob análise, que é composta por 136 barras (pontos enumerados de 0 a 135). O problema consiste em encontrar a melhor alocação para 10 chaves seccionadoras (incluindo o disjuntor da subestação).

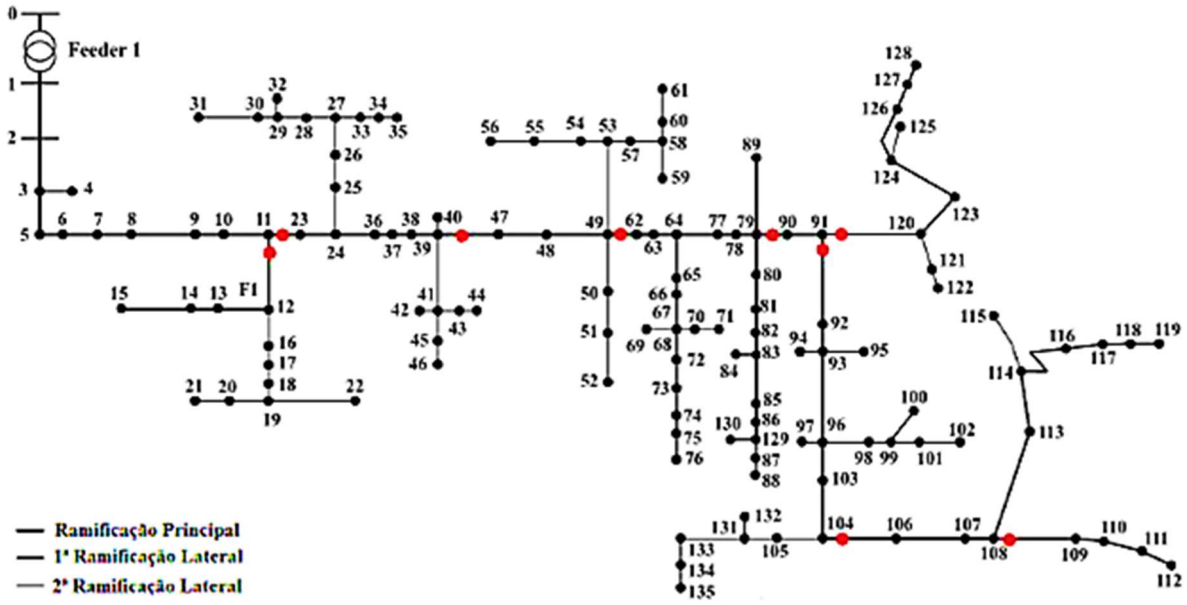
Figura 12 - Rede de distribuição com 136 barras.



Fonte: LAPSEE (2020).

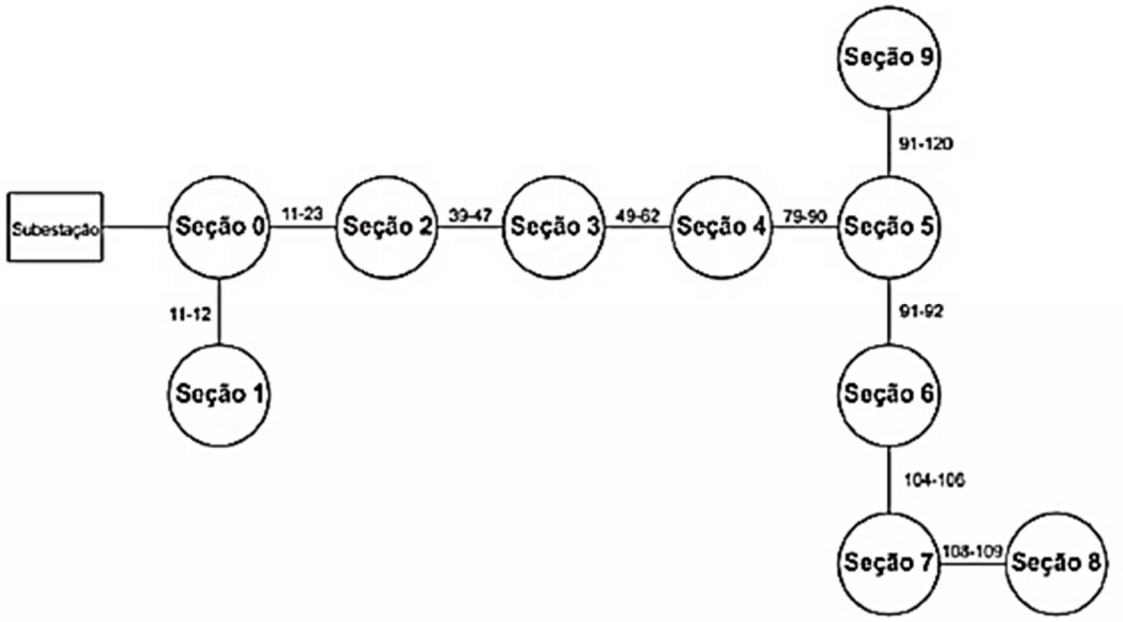
As Figuras 13 e 14 ilustram a configuração inicial (não otimizada) das chaves, conforme lida do arquivo rede.xml.

Figura 13 – Alocação das chaves na rede de distribuição (Configuração Inicial).



Fonte: Adaptado do LAPSEE (2025).

Figura 14 – Diagrama de seções da rede de distribuição (Configuração Inicial).



Fonte: Jun 2025.

Para o cálculo do CIC, os parâmetros de custo e taxa de falha foram utilizados conforme descrito a seguir:

- A taxa de falhas permanentes em toda a rede é de γ = 0,025 falhas/km-ano;

- As funções custo de restauração para manutenção na rede são fr = 7.0 US\$, fc = 200.00 US\$ e fi = 30.00 US\$;
- As funções custo de chaveamento na rede são fr = 0.55 US\$, fc = 40.00 US\$ e fi = 5.0 US\$

O cálculo do CIC para esta configuração inicial (Figuras 13 e 14) resultou em US$ 30.046,92, correspondente a um *fitness* de 3,3278. Este valor serve como linha de base para medir a eficácia das otimizações.

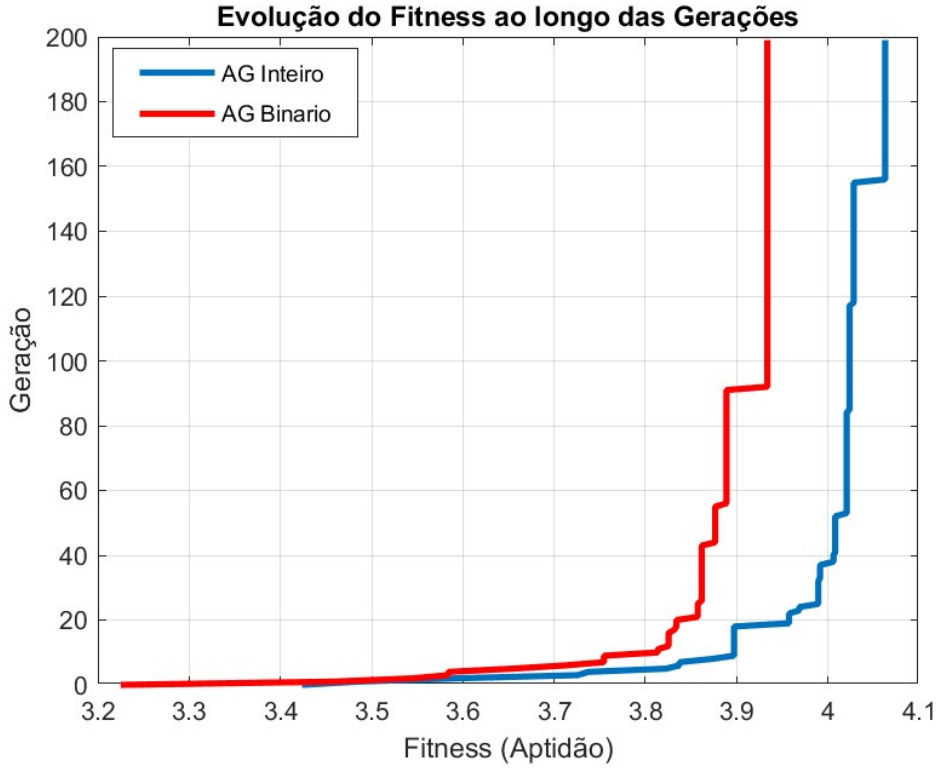
5.1 Análise Comparativa de Convergência

Na primeira etapa da análise, ambos os algoritmos (AG binário e AG inteiro) foram executados em uma simulação única para avaliar sua velocidade de convergência. Os parâmetros utilizados, baseados na Tabela 1 para 10 chaves, foram:

- Tamanho da População (nP): 100
- Quantidade de Gerações (nG): 200
- Taxa de Recombinação (tR): 0,9
- Taxa de Mutação (Tm): 0,15

A Figura 15 apresenta a evolução do melhor *fitness* encontrado por cada metodologia ao longo das 200 gerações.

Figura 15 – Gráfico da evolução do Fitness ao longo das gerações (Comparativo).



Fonte: Própria do autor.

Pela análise da Figura 15, observa-se que ambas as metodologias conseguiram melhorar significativamente a solução inicial (*fitness* de 3,3278), com o AG inteiro atingindo valores de *fitness* superiores a 4,0 (correspondente a um CIC inferior a US$ 25.000,00).

No entanto, a velocidade de convergência apresentou uma diferença notável:

- O AG Binário (linha vermelha) demonstrou uma convergência mais rápida, atingindo o patamar de *fitness* 3,9 por volta da geração 90.

- O AG Inteiro (linha azul) apresentou uma convergência mais eficiente, levando menos gerações para atingir os mesmos patamares de *fitness*, alcançando um resultado de alta qualidade ao final das 200 gerações.

Em termos de tempo de execução para esta simulação, os valores foram muito próximos:

- AG binário: 2574 ms (2,57 segundos)
- AG inteiro: 2774 ms (2,77 segundos)

Embora o AG binário tenha sido levemente mais rápido em tempo de processamento, o AG inteiro apresentou maior eficiência em alcançar soluções com *fitness* maior.

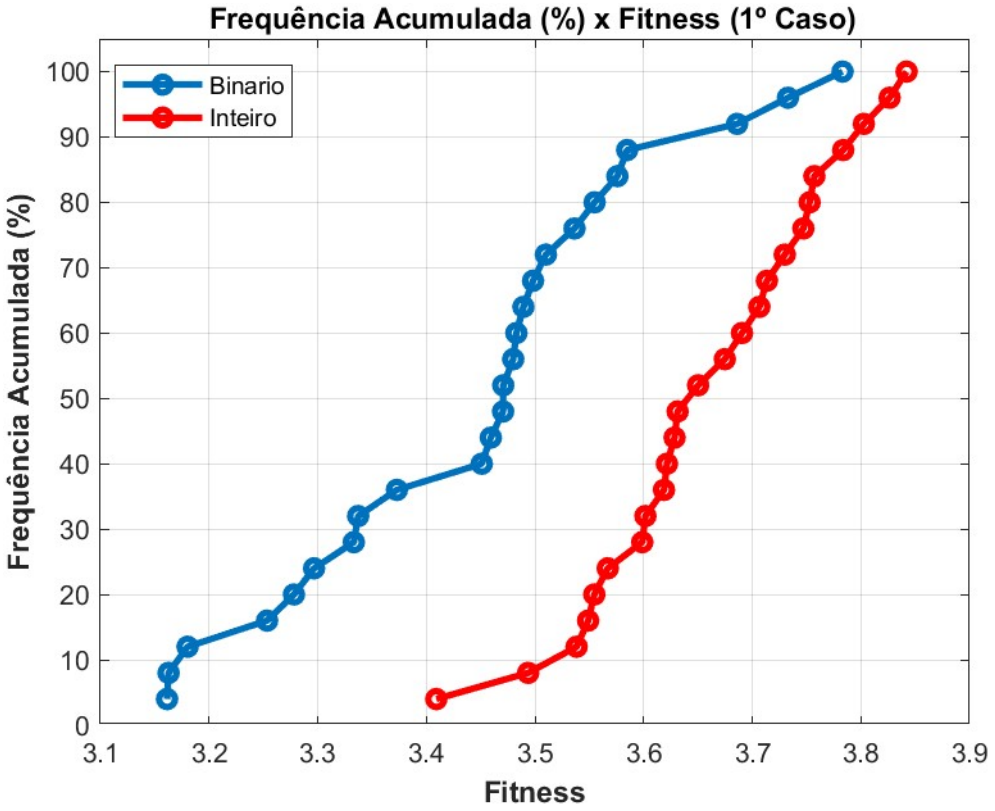
5.2 Análise Comparativa de Robustez e Estabilidade

Para entender como a alteração dos parâmetros afeta o valor do *fitness* e a estabilidade de cada metodologia, foi realizada uma análise de robustez. Esta análise utilizou parâmetros reduzidos para simular um cenário com recursos computacionais limitados:

- Tamanho da População (nP): 10
- Quantidade de Gerações (nG): 30
- Taxa de Recombinação (tR): 0,9
- Taxa de Mutação (Tm): 0,15
- Número de Repetições: 25 execuções

A Figura 16 apresenta a curva de frequência acumulada do melhor *fitness* obtido nas 25 execuções para cada metodologia.

Figura 16 – Gráfico de Frequência Acumulada (%) x Fitness (1º Caso).



Fonte: Própria do autor.

A análise da Figura 16 continua a demonstrar a superioridade do método inteiro no desempenho em comparação com a Figura 15. Com parâmetros reduzidos, o AG Inteiro (linha vermelha) foi consistentemente melhor:

- A curva vermelha (AG Inteiro) está visivelmente deslocada para a direita, indicando que, na maioria das 25 execuções, ela encontrou soluções com *fitness* superior ao do AG Binário.

- O AG Binário (linha azul) apresentou uma dispersão muito maior, encontrando soluções de *fitness* mais baixo em muitas de suas execuções.

Os dados estatísticos coletados durante este experimento (Tabela 2) confirmam a análise visual:

Tabela 2 - Dados estatísticos (nG=30, nP=10, 25 repetições)

Metodologia	Média (Tempo)	Desvio Padrão (Fitness)	Variância (Fitness)
AG Binário	167,16 ms	0,1673	0,0280
AG Inteiro	167,04 ms	0,1086	0,0118

Fonte: Própria do autor.

A análise da Tabela 2 demonstra que:

- O AG Inteiro foi significativamente mais estável e robusto. Seu desvio padrão (0,1086) e variância (0,0118) foram menores que os do AG Binário (0,1673 e 0,0280, respectivamente). Isso significa que o AG Inteiro retorna resultados de qualidade de forma mais consistente.
- A menor dispersão e o deslocamento da curva (Figura 16) indicam que a média de *fitness* do AG Inteiro foi superior à do AG Binário neste cenário.
- O tempo médio de execução para este experimento foi muito próximo para as duas metodologias.

6 CONCLUSÃO

A confiabilidade das redes de distribuição de energia é essencial para o planejamento energético, e a alocação otimizada de chaves seccionadoras é uma das principais estratégias para aliviar os impactos de falhas. Este trabalho focou em otimizar essa alocação visando a minimização do CIC.

A natureza combinatória e não linear (PNLIM) deste problema de otimização torna o uso de métodos clássicos inviável. Embora o AG seja uma técnica eficaz, seu desempenho é diretamente ligado às suas escolhas de implementação. Este Trabalho de Graduação investigou um método central: o impacto da codificação do cromossomo na eficiência da solução. O objetivo principal foi realizar uma análise crítica comparativa entre a codificação binária e a codificação decimal (inteira).

Ambas as metodologias foram implementadas e validadas em um estudo de caso de uma rede de 136 barras, demonstrando a capacidade de otimizar a configuração inicial, que possuía um CIC de US$ 30.046,92 (fitness 3,3278). Ambas as abordagens foram capazes de encontrar configurações superiores.

Contudo, a análise comparativa de desempenho, apresentada no Capítulo 5, revelou uma vantagem clara e decisiva da metodologia decimal (inteira) em ambos os cenários de teste:

Na análise de convergência (Figura 15): A abordagem inteira demonstrou uma velocidade de exploração muito superior. Ela atingiu a região de soluções de alta qualidade (*fitness* de 3,9) em aproximadamente 20 gerações, enquanto a abordagem binária necessitou de 90 gerações para alcançar o mesmo patamar.

Na análise de robustez (Figura 16), a abordagem inteira novamente se mostrou superior, apresentando resultados médios consistentemente melhores e com variabilidade significativamente menor (variância de 0,0118 contra 0,0280 da binária), provando ser uma metodologia mais estável e confiável.

Notavelmente, essa superioridade em robustez foi alcançada sem muita diferença computacional. Os dados de tempo de execução (Tabela 2) mostraram que o tempo médio de processamento de ambas as metodologias foi bem próximo.

Este trabalho conclui, portanto, que a codificação decimal (inteira) representa uma metodologia mais eficiente e robusta para o problema de alocação de chaves seccionadoras. Ao tratar a restrição do número de chaves (K) de forma nativa, o algoritmo demonstrou uma capacidade superior de navegar pelo espaço de busca, entregando soluções melhores de forma mais rápida e confiável do que a abordagem binária.

REFERÊNCIAS

BÄCK, T. **Evolutionary Algorithms in Theory and Practice**. New York: Oxford University Press, 1996.

COELLO COELLO, C. A.; VAN VELDHUIZEN, D. A.; LAMONT, G. B. **Evolutionary Algorithms for Solving Multi-Objective Problems**. 2. ed. New York: Springer, 2002.

FERNANDES, M. N. **Análise de desempenho de algoritmos de busca em grafos para processamento de topologia de redes de distribuição de energia**. 2023. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) – Universidade Estadual Paulista (UNESP), Ilha Solteira, 2023.

JIN, L. J. **Análise da técnica de solução para o problema de alocação ótima das chaves seccionadoras em redes de distribuição de energia elétrica**. 2025. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) – Universidade Estadual Paulista (UNESP), Ilha Solteira, 2025.

LEITE, J. B.; MANTOVANI, J. R. S. **Análise da implementação do algoritmo para o cálculo do custo de interrupção (CIC), em redes de distribuição de energia, e da análise do algoritmo genético na solução do problema de alocação de chaves em alimentadores radiais de distribuição de energia elétrica**. 2021. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) – Universidade Estadual Paulista (UNESP), Ilha Solteira, 2021.

MICHALEWICZ, Z. **Genetic Algorithms + Data Structures = Evolution Programs**. 3. ed. Berlin: Springer, 1996.

MORELATO, A. L.; MONTICELLI, A. Heuristic search approach to distribution system restoration. **IEEE Transactions on Power Delivery**, [S. l.], v. 4, n. 4, p. 2235-2241, out. 1989.

REIZ, C. et al. A Multiobjective Approach for the Optimal Placement of Protection and Control Devices in Distribution Networks With Microgrids. **IEEE Access**, [S. l.], v. 10, p. 41776-41788, 2022. DOI: 10.1109/ACCESS.2022.3166918.

TENG, J.-H.; LIU, Y.-H. A novel ACS-Based optimum switch relocation method. **IEEE Transactions on Power Systems**, [S. l.], v. 18, n. 1, p. 113-120, fev. 2003.