UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA
CÂMPUS DE ILHA SOLTEIRA

Felipe Fontes Viçozo

Alocação de Subestações no Planejamento da Expansão dos Sistemas de Distribuição de Energia Elétrica

Ilha Solteira
2024

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

Felipe Fontes Viçozo

Alocação de Subestações no Planejamento da Expansão dos Sistemas de Distribuição de Energia Elétrica

Trabalho de conclusão de curso apresentado à Faculdade de Engenharia de Ilha Solteira – Unesp como parte dos requisitos para obtenção do título de bacharel em Engenharia Elétrica.

Nome do orientador
Prof. Dr. Jonatas Boas Leite

Ilha Solteira
2024

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Viçozo, Felipe Fontes.
V639a Alocação de subestações no planejamento da expansão dos sistemas de distribuição de energia elétrica / Felipe Fontes Viçozo. -- Ilha Solteira: [s.n.], 2024
63 f. : il.

Trabalho de conclusão de curso (Graduação em Engenharia Elétrica) - Universidade Estadual Paulista (UNESP), Faculdade de Engenharia, Ilha Solteira, 2024

Orientador: Jonatas Boas Leite

Inclui bibliografia

1. Subestações. 2. Fluxo de potência. 3. Clusterização.

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

FOLHA DE APROVAÇÃO

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

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO

Aos quatorze dias do mês de junho do ano de dois mil e vinte e quatro, o discente ***Felipe Fontes Viçozo***, matriculado sob o nº 172054362, tendo como banca examinadora o seu orientador, *o Prof. Dr. Jonatas Boas Leite, o Doutorando Brian Daniel Jaramillo León e o Doutorando Wilson Enrique Chumbi Quito*, apresentou o Trabalho de Graduação intitulado **"Alocação de Subestações no Planejamento da Expansão dos Sistemas de Distribuição de Energia Elétrica"**, obtendo a nota 10,0 (DEZ) e conceito APROVADO.

Prof. Dr. Jonatas Boas Leite
- Orientador-

Felipe Fontes Viçozo
- Discente-

Doutorando Brian Daniel Jaramillo León Blanco
- Membro da Banca -

Doutorando Wilson Enrique Chumbi Quito
- Membro da Banca -

AGRADECIMENTOS

Gostaria de agradecer a Deus, pela perseverança e sabedoria concedida durante os árduos anos de graduação. Aos meus pais, Ilton e Marli, pelo apoio incondicional, por me ensinarem a correr atrás de meus objetivos e nunca medirem esforços para possibilitar que eu tivesse um ensino de qualidade, estando presentes em meus momentos felizes e tristes e nunca largarem minha mão.

Aos meus amigos da República Santo Grau, com quem compartilhei os melhores momentos durante o período de faculdade e se tornaram família em Ilha Solteira. Em especial a meus amigos Lucas Vilela e Teodoro de Andrade por todo o companheirismo, suporte e motivação para enfrentar os desafios da graduação.

Ao meu orientador, Jonatas, por toda a disponibilidade, suporte e orientação durante a elaboração deste trabalho de graduação. A todos os docentes do curso de Engenharia Elétrica de FEIS – UNESP, por compartilharem seus conhecimentos, sempre com excelência, e me tornarem um jovem pronto a ingressar no mercado de trabalho e exercer minha função como profissional de engenharia.

E por fim a cidade de Ilha Solteira, onde vivi alguns dos melhores anos de minha vida, onde cresci como pessoa e me encontrei, e mesmo sendo uma cidade bem quente, é admirável.

RESUMO

Este trabalho tem como objetivo principal o desenvolvimento de uma metodologia para auxiliar no planejamento da alocação de novas subestações em redes de distribuição elétrica preexistentes, com o intuito de reduzir os custos operacionais do sistema e dos encargos associados a construção do novo centro de distribuição de energia. Para tal desígnio, foi elaborado uma heurística construtiva que após o cálculo do fluxo de potência em uma rede já existente, procede com a segmentação em *clusters* das barras do sistema por meio do algoritmo *K-means*, considerando como critério de clusterização a posição geográfica de cada barra, e determina a localização ideal para a construção da nova subestação, visando reduzir os custos relativos às perdas de potência ativa e otimizar a distribuição de energia. Utilizou-se uma rede de distribuição composta por 136 barras, com oito possíveis locais para a construção de um novo centro de distribuição de energia. A fim de identificar a solução mais vantajosa, foram ajustados parâmetros como a localização da nova subestação, o tipo de cabo a ser empregado na conexão do novo centro de distribuição e o sistema de distribuição principal, e o ponto na rede onde ocorre a mudança de *cluster*, com o propósito de determinar a solução com os menores custos associados.

Palavras-chave: Algoritmo K-means. Alocação de subestações. Clusterização. Fluxo de Potência. Sistemas de distribuição de energia elétrica.

ABSTRACT

This work's main objective is to develop a methodology to assist in planning the allocation of new substations in pre-existing electrical distribution networks, with the aim of reducing the operating costs of the system and the charges associated with the construction of the new energy distribution center. For this purpose, a constructive heuristic was developed which, after calculating the power flow in an existing network, proceeds with the segmentation into *clusters* of the network buses using the K-means algorithm, considering the geographic position as the clustering criterion of each bus, and determines the ideal location for the construction of the newsubstation, aiming to reduce costs related to active power losses and optimize energy distribution. A distribution network consisting of 136 buses was used, with eight possible locations for the construction of a new energy distribution center. In order to identify the most advantageous solution, parameters such as the location of the new substation, the type of cable to be used to connect the new distribution center and the main distribution system, and the point in the network where the cluster change occurs, with the purpose of determining the solution with the lowest associated costs.

Keywords: Electrical Distribution Systems. K-means Algorithm. Power Flow.Clustering. Substation Allocation.

LISTA DE FIGURAS

LISTA DE TABELAS

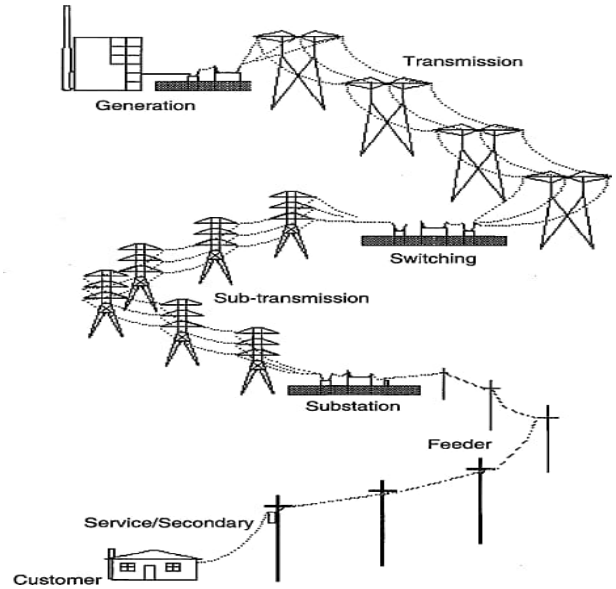
LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Um sistema de transmissão e distribuição de energia consiste em um conjunto de diversas linhas de transmissão, subestações e alimentadores (Willis, 2004) que são utilizadas para escoar a energia que sai das unidades geradoras (usinas hidrelétrica, eólicas, fotovoltaicas, entre outras) até os consumidores finais.

Figura 1: Sistema de transmissão e distribuição de energia.



Fonte: Willis (2004).

Ano após ano, a demanda por energia elétrica aumenta, aliada ao crescimento da população, expansão das indústrias e maior inserção da tecnologia nas atividades do dia a dia. Para atender essa demanda crescente por energia, no Brasil, o Ministério de Minas e Energia (MME) e a Agência Nacional de Energia Elétrica (ANEEL) planejam a expansão dos sistemas de distribuição e transmissão de energia elétrica. O planejamento é iniciado com os estudos técnicos fornecidos pelo Operador Nacional do Sistema Elétrico (ONS) e pela Empresa de Pesquisa Energética (EPE), que servem como base para a consolidação dos programas de expansão, tais como os leilões de transmissão.

Na prática, a expansão supracitada consiste na criação de novas unidades geradoras, na modernização e criação de novos trechos de linha de transmissão, inserção de equipamentos de compensação reativa e alocação de subestações em

pontos estratégicos da rede, para atender ao aumento da demanda por eletricidade, reduzir perdas totais e assegurar a confiabilidade sistema do elétrico.

No contexto da expansão de sistemas de distribuição, uma concessionária de energia elétrica, ao prever um aumento de demanda a ser atendida em longo prazo, elabora um plano diretor que deve definir o tipo, capacidade e localização da instalação de novos equipamentos (banco de capacitores, reguladores de tensão, novos alimentadores, entre outros) para uma dada área geográfica ou região com um conjunto de cargas previamente estimadas, considerando as restrições de capacidade em alimentadores, queda de tensão e previsões de demanda (Porkar et al., 2011) de forma a minimizar os custos totais de instalação e operação, respeitando as exigências técnicas para um bom funcionamento do sistema (Khodr et al., 2009), ou ainda considerando a avaliação dos índices de confiabilidade.

Dentre os objetivos principais que englobam os problemas de planejamento da expansão de sistemas elétricos de distribuição pode-se destacar a redução das perdas totais, o aumento da confiabilidade e a melhoria do perfil de tensão da rede (Shojaeian & Ghandehari, 2013) visando proporcionar economia monetária, redução de faltas, diminuição das interrupções do sistema e aumento da qualidade da energia oferecida ao consumidor. Por conta disso, estes temas são muito discutidos na literatura e vários métodos vêm sendo propostos.

A natureza combinatória desses tipos de problema é imposta pela restrição da radialidade e conectividade da rede, pela variedade de opções para alocação de novas subestações e topologia dos alimentadores, decisões de investimento, incerteza sobre a variação da demanda e da localização de faltas no sistema, disponibilidade de equipamentos, dentre outros (Singh, 2004). O desafio, portanto, é integrar o maior número de subproblemas, de forma a maximizar os benefícios.

As estratégias tradicionais de planejamento são baseadas na experiência e em regras pré-estabelecidas. Caso a demanda atinja um limite pré-determinado pela concessionária, uma nova instalação deve ser adicionada ao sistema, que pode ser a inclusão de novas subestações ou expansão da capacidade das mesmas (repotencialização), inserção de novos alimentadores ou ambos.

O planejamento envolve a seleção do número, localização e tamanho das subestações, além da configuração dos alimentadores de tal modo que o custo de instalação, juntamente com o custo da perda de energia, seja mínimo, mantendo a radialidade da rede e ao mesmo tempo não excedendo as limitações de capacidade e de queda de tensão em qualquer parte da rede.

1.1. OBJETIVOS GERAIS E ESPECÍFICOS

Neste trabalho de graduação, o principal objetivo é o desenvolvimento de um método computacional para análise da redução de custos atrelados a operação de uma rede de distribuição, utilizando uma topologia de rede, visando à otimização do planejamento da expansão dos sistemas de distribuição de energia elétrica, em um determinado período.

O método computacional tem como foco determinar, com base no resultado do fluxo de potência para a topologia de rede adotada, uma solução ótima que resulte na minimização dos custos das perdas de energia, através da determinação de local estratégico para a inserção de novas subestações, utilizando o algoritmo *K-means*, e determinação do melhor tipo de cabo para os alimentadores, minimizando também os custos atrelados a esses processos. Além disso, o método visa minimizar a queda de tensão ao longo da rede, a fim de melhorar a eficiência geral do sistema.

2 SUBESTAÇÕES

Segundo o Procedimento de Distribuição de Energia Elétrica no Sistema Elétrico Nacional – PRODIST “uma subestação consiste em um conjunto de instalações elétricas, em média ou alta tensão, que agrupam equipamentos, condutores e acessórios, destinados à proteção, medição, manobra e transformação de grandezas elétricas” (ANEEL, 2022). As subestações podem ser classificadas de acordo com sua função no sistema elétrico.

Quando conectadas em unidades geradoras, elas exercem o papel de elevar a tensão gerada a níveis de tensão utilizados na transmissão, portanto são chamadas de subestações elevadoras; quando conectam o sistema de transmissão com o de subtransmissão, são denominadas subestações de subtransmissão. Por fim, quando conectam o sistema de subtransmissão ao de distribuição (que leva energia aos consumidores finais), são denominadas estações distribuidoras (Frontin, 2013).

As subestações podem também ser classificadas quanto ao método de isolamento elétrico adotado:

a) **AIS (*Air isolated Substation*)**: São as subestações onde o isolamento entre as fases é feito através do ar. Trata-se das subestações convencionais. Devido ao isolamento elétrico ser realizado pelo ar, a distância entre as fases é grande, e devido a isso são necessários grandes espaços físicos para a construção desse tipo de subestação. Nessa, os equipamentos são dispostos em pátios, respeitando o distanciamento necessário entre eles. Na Figura 2 é apresentado um exemplo de AIS;

b) **GIS (*Gas Isolated Substation*)**: As subestações isoladas a gás, utilizam o gás SF_6 para o isolamento entre as fases. Esse gás é altamente dielétrico, e devido a isso, reduz o distanciamento necessário entre as fases. Nessa, os equipamentos são todos acoplados e inseridos dentro de compartimentos por onde o gás circula. Dessa forma, o espaço físico utilizado para a construção de uma subestação é reduzido, permitido a construção até mesmo dentro de centro urbanos, contudo, seu custo de construção é superior quando comparada uma subestação AIS. Na Figura 3 é apresentado um exemplo de GIS.

Figura 2: Subestação isolada a ar (AIS)

Fonte: Frontin (2013).

Figura 3: Subestação isolada a gás SF_6 (GIS)

Fonte: Siemens Energy (2023).

As subestações são compostas, de forma geral, dos seguintes equipamentos e sistemas:

a) **Transformador:** Equipamento com a função de transferir potência elétrica de um circuito para outro, transformando (Elevando/Diminuindo) tensões e correntes alternadas (AC), ou alterando os valores das impedâncias do circuito em que opera;
b) **Disjuntores:** Sua principal função é interromper correntes de curto-circuito em um pequeno intervalo de tempo, a fim de proteger os demais

equipamentos dos possíveis danos causados pela alta corrente supracitada. Ainda, devem ter a capacidade de estabelecer correntes de falta, como também de estabelecer/interromper correntes de menor magnitude, quando necessário, e isolar circuitos quando aberto;

c) **Seccionadoras:** Tem como objetivo a isolação segura dos equipamentos e das linhas de transmissão. Sua finalidade principal é garantir um distanciamento elétrico que assegure o isolamento do circuito após a abertura de um equipamento de bloqueio principal, tal como um disjuntor. Ainda, podem exercer funções como *by-pass* de disjuntores e bancos de capacitores, isolação de equipamentos para manutenção, transferência de barras de uma subestação e aterramento;
d) **Para-raios:** Seu papel é garantir que o sistema esteja protegido contra sobretensões. Eles operam como limitadores de tensão, impedindo que tensões acima das quais ele está preestabelecido para operar possam alcançar os equipamentos que ele visa proteger. Contribuem para o aumento de confiabilidade do circuito, economia (evitam danos aos equipamentos) e garantem continuidade de operação;
e) **Sistema de proteção e controle:** Parte essencial para o correto funcionamento de uma subestação e atuação dos equipamentos de proteção. Garante a automação da subestação, de forma a monitorar e controlar as grandezas elétricas intrínsecas ao processo de distribuição de energia, através do controle dos dispositivos de proteção e garantindo que eles atuem quando necessário. É composto por quatro níveis de automação (Negromonte, 2018):
    - Nível 0: Equipamentos de pátio tais como disjuntores, seccionadoras e religadores;
    - Nível 1: Composto pelos relés de proteção, que estão inseridos dentro dos painéis de proteção e controle;
    - Nível 2: Composto pelos computadores que fazem a supervisão da subestação e de seus equipamentos, constituído por uma unidade central e uma interface homem máquina (IHM);
    - Nível 3: Centro de operações do sistema, onde ocorre a operação a distância da subestação.

3 EXPANSÃO DO SISTEMA DE DISTRIBUIÇÃO

Primeiramente, apresenta-se a formulação matemática para o problema de expansão dos sistemas de distribuição com alocação das subestações. Devido à natureza combinatória deste tipo de problema, propõe-se a utilização de uma heurística construtiva que requer um algoritmo de cálculo do fluxo de potência e clusterização. Para isso a rede de distribuição é representada usando a teoria de grafos.

3.1 FORMULAÇÃO DA EXPANSÃO DO SISTEMA DE DISTRIBUIÇÃO

Com o objetivo de minimizar o custo total, considerando o custo da construção de novas subestações, C^{SE}, e alimentadores, C^{CCT}, e custo das perdas ativas na rede, C^{PERDAS}, no problema da expansão de sistemas de distribuição, tem-se o problema de otimização, cuja função objetivo é dada por (01).

$$\min C^{TOTAL} = C^{PERDAS} + C^{CCT} + C^{SE} \quad (01)$$

Sendo:

$$C^{PERDAS} = TC^{ENERGIA} \sum_{n=1}^{Na} \left((1 + J^{ENERGIA})^{-n} \sum_{i \in \Omega_L} R_i |I_i|^2\right) \quad (02)$$

$$C^{CCT} = \sum_{i=1}^{NCCT} C_i^{CABO} l_i \quad (03)$$

$$C^{SE} = \sum_{i=1}^{N_{SE}} C_i^{SE} \quad (04)$$

s.a.

$$\underline{V_i} \leq V_i \leq \overline{V_i}, \quad i \in \boldsymbol{\Omega}_B \quad (05)$$

$$0 \leq I_i \leq \overline{I_i}, \quad i \in \boldsymbol{\Omega}_L \quad (06)$$

$$\left(P_i^{SE}\right)^2 + \left(Q_i^{SE}\right)^2 \leq \left(\overline{S_i^{SE}},\right)^2, \quad i \in \boldsymbol{\Omega}_{SE} \quad (07)$$

O custo das perdas em (02) dependa do total de horas em um ano, T, da tarifa de energia $C^{ENERGIA}$ e da taxa de juros da energia, $J^{ENERGIA}$ ao longo de Na anos. Na determinação do custo de novos alimentadores (03), os custos dos cabos C^{CABO} e seus comprimentos l_i são considerados. Em (04), assume-se a construção e/ou repotencialização de N_{SE} subestações.

A restrição (05) se refere aos limites de tensão das barras na rede, $\boldsymbol{\Omega}_B$, fazendo com que elas permaneçam dentro de valores aceitáveis para um bom funcionamento do sistema. Já (06) se refere aos limites de corrente nominais dos ramos da rede, $\boldsymbol{\Omega}_L$, fazendo que com que os condutores funcionem dentro das margens previstas. Em (07) tem-se a capacidade máxima de potência permitida para as subestações.

3.2 FLUXO DE POTÊNCIA

O fluxo de potência representa a transferência de energia elétrica ao longo das barras, ramos e equipamentos que compõe uma rede de energia elétrica. O seu cálculo visa a determinar o estado operativo de uma rede, através da obtenção dos fasores de tensão nas barras (nós), fasores de corrente nos ramos (linhas), utilizando os valores das potências ativa (P) e reativa (Q) e as equações de balanço de potência:

$$P_i^G - P_i^D = V_i \sum_{ij \in c} V_j \left(G_{ij} cos\theta_{ij} + B_{ij} sin\theta_{ij}\right) \quad (07)$$

$$Q_i^G - Q_i^D = V_i \sum_{ij \in c} V_j \left(G_{ij}\, sin\theta_{ij} + B_{ij}\, cos\,\theta_{ij}\right) \quad (08)$$

Com esses valores, é possível determinar os perfis de tensão na rede, o cálculo das perdas de potência, e verificar a existência de problemas de tensão e sobrecargas, sendo dessa forma um estudo de extrema importância para as distribuidoras de energia, pois auxilia o planejamento da expansão e o incremento da confiabilidade no sistema (Martins, 2028).

3.2.1 Método de Varredura *Backward/Forward*

Uma das formas para o cálculo do fluxo de potência é a utilização de algoritmos que empregam o método de varredura backward/foward (B/F). Explorando a radialidade das redes de distribuição, o método de varredura B/F executa varreduras e iterações a fim de convergir em valores estáveis para as tensões e correntes que fluem no sistema:

a) *Varredura Backward:* Nessa etapa, o algoritmo adota o sentido reverso do fluxo de corrente, partido das barras de carga em direção a barra de origem. Essa fase da execução do algoritmo visa determinar os valores das correntes que fluem no sistema, como em (09).

$$\dot{J}_{i,p}^{k} = -\dot{I}_{i,p}^{k} + \sum_{m \in M_i} \dot{J}_{m,p}^{k} \tag{09}$$

Em que M_i é o conjunto de ramificações adjacentes a jusante ligada ao nó *i*, e $p \in (a, b, c)$ é a fase.

b) *Varredura Forward:* Nessa etapa, o algoritmo inicia a varredura partindo da barra de referência em direção às barras de carga, que se situam nas extremidades da rede. Assumindo um valor de tensão predeterminado para a barra de referência e da matriz de impedância de linha, são calculadas as tensões nos demais nós que compõe a rede, como em (10).

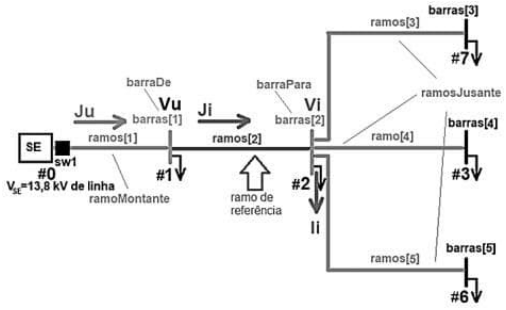
$$\dot{V}_{i,p}^{k} = \dot{V}_{i-1,p}^{k} - \sum_{q \in (a,b,c)} \dot{J}_{i,q}^{k} \bar{Z}_{i-1,p,q} \tag{10}$$

Em que $V_{i-1,p}^{k}$ representa a tensão a montante adjacente ao nó *i*, e $Z_{i-1,i}$ a impedância de linha. Por se tratar de um processo iterativo, essas varreduras são realizadas diversas vezes, de forma que os valores obtidos são atualizados após cada iteração, visando atingir o critério de convergência adotado (Marchesan, 2023).

3.3 TOPOLOGIA DA REDE

É comum que uma rede de distribuição seja representada através de uma lista de adjacências, a qual composta das conexões entre as barras e ramos e de seu sequenciamento, permitindo a identificação das componentes às jusantes e a montantede cada elemento, dessa forma armazenando as informações da topologia da rede real.

Figura 4 – Topologia de uma rede com seis barras



Fonte: Adaptado de Ferreira (2021)

3.3.1 GRAFOS

Um grafo é uma estrutura $G = (\boldsymbol{V}, \boldsymbol{E})$ que consiste em um agrupamento de $\boldsymbol{V}$ elementos, chamados de nós ou vértices, e um conjunto de arestas desordenadas, no formado $\boldsymbol{E}(i, j) ou\ \boldsymbol{E}(j, i)$, onde $i\ e\ j \in \boldsymbol{V}$. É notável a semelhança entre um grafo e a topologia de uma rede de distribuição, a qual é caracterizada pela conexão entre barras (Vértices) e ramos (arestas). Por conta disso, as informações de uma topologia de rede podem ser representadas através de um grafo, tornando possível o uso das funcionalidades dessa estrutura para análises de problemas relacionados a distribuição de energia elétrica. A utilização de algoritmos de navegação, que realizam buscas através de uma lista de adjacências pré-determinada, aliado a representação de uma rede utilizando grafos possibilita, por exemplo, o cálculo do fluxo de carga nessa rede (Tenesaca-Caldas, et al., 2022).

A propriedade de autossimilaridade dos grafos também favorece a utilização dessa estrutura para análises de redes de distribuição. Essa propriedade garante que recortes dos grafos se assemelham a estrutura original, devido aos grafos serem estruturas repetitivas, de forma que as mesmas características do grafo original são válidas nesses recortes (subgrafos). É uma propriedade típica dos fractais, devido a essa repetição da estrutura original ocorrer em diferentes escalas analisadas, possibilitando a clusterização.

3.3.2 BUSCA EM PROFUNDIDADE

Uma das formas de possibilitar que toda a topologia da rede seja percorrida durante a aplicação do método de varredura B/F é a utilização do algoritmo de busca

em profundidade (DFS), do Inglês, *Depth- first Search*. Esse algoritmo se aproveita da representação da topologia da rede em formato de grafo para garantir que cada vértice v do grafo seja visitado. Após visitar um vértice, cada outro vértice adjacente é visitado, e caso um vértice não tenha outros adjacentes, ou todos os seus vértices adjacentes tenham sido visitados, o algoritmo retorna ao predecessor de v. No caso de existirem vértices não visitados, a busca é reiniciada através deles. A busca é finalizada se o procedimento de visita e retrocesso entre os nós levar ao primeiro vértice, de início a busca (Drozdek, 2012). No quadro 1, tem-se o pseudocódigo do algoritmo DFS.

Quadro 1 – Pseudocódigo do algoritmo DFS.

```
1- DFS(v)
2-  num(v) = i+1;
3-  para todos vértices u adjacentes à v
4-   se num(u) é igual a 0
5-    Adicionar a aresta (uv) em arestas
6-    DFS(u)    >>>RECURSÃO
7-   fim se
8-  fim para
9- fim DFS()
```

```
10- BuscaEmProfundidade(v)
11-  para todos vértices v
12-   num(v) = 0
13-  fim para
14-  arestas = vazio
15-  i = 1
16-  enquanto existir v tal que num(v) é 0
17-   DFS(v)
18-  fim enquanto
19- fim BuscaEmProfundidade()
```

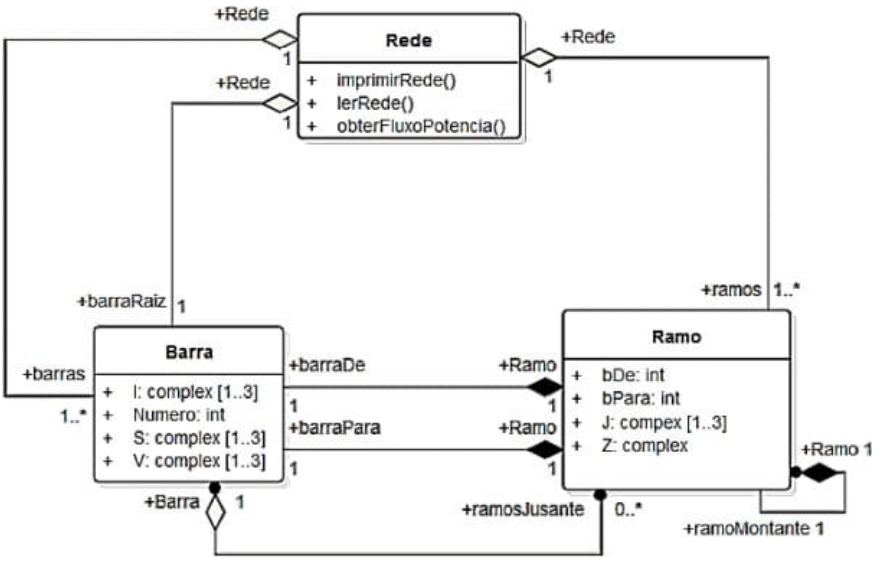
Fonte: (De Souza & Leite, 2023)

Quando aplicado para o cálculo do fluxo de potência, o algoritmo atualiza a tensão na barra no início dos cálculos, e quando a busca em profundidade é chamada recursivamente, o algoritmo faz a atualização do fluxo das correntes nos ramos.

Para facilitar a implementação desse algoritmo, é importante que os dados das redes sejam modelados usando um diagrama de classes e suas relações, seguindo os conceitos definidos pela Linguagem de Modelagem Unificada (UML), do Inglês *unified modeling language*.

Dada uma rede de distribuição real (*Rede*), composta por um número n, finito*, de* barras *(Barra)* e número *m, finito, de* ramos (*Ramo*). Cada ramo possui uma barra de origem (*barraDe*) e uma barra de destino (*barraPara*). Uma barra possui nenhum ou k ramos em sua jusante (*ramosJusante*). Na Figura 5, é apresentado o diagrama de classes em UML do modelo básico de uma rede de distribuição.

Figura 5: Diagrama de classes e relações proposto para uma rede de distribuição.



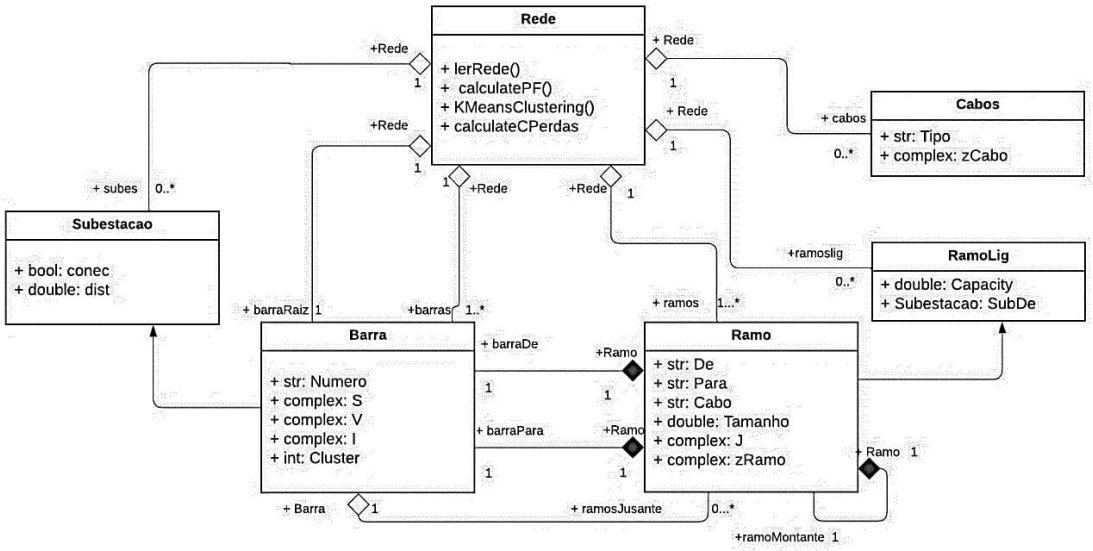
Fonte: Adaptado de Ferreira (2021)

A classe *Rede* agrega objetos da classe *Barra* e *Ramo.* Os objetos do tipo barra agregam objetos do tipo ramo, denominados de ramosJusante. Os objetos do tipo *ramo* possuem em sua composição objetos do tipo *Barra,* denominados b*arraDe* e *barraPara, e* também por um objeto do tipo *ramo,* denomidado *ramoMontante*.

Sendo assim, utilizando a UML, é possível que a radialidade da rede seja representada computacionalmente, de forma que um algoritmo recursivo, tal como o DFS, consiga percorrer todos os componentes desse modelo de dados da rede, assegurando que o fluxo de potência seja calculado em todas as barras e ramos, e consequentemente na rede completa.

Particularmente, para a solução do problema de expansão do sistema de distribuição com a alocação de novas subestações, adiciona-se ao modelo de rede básico mais três classes: *Subestação*; *Cabo*; e *RamoLig*. Dado um número finito de localidades para subestações candidatas, deve-se selecionar aquela que reduz os custos. Assim uma lista de cabos é necessária para a construção do novo circuito de conexão da subestação selecionada. No caso de divisão de um alimentador de distribuição em dois, ou mais, os ramos de ligação representam pontos de transferência de carga para alimentadores vizinhos. Na Figura 6, é apresentado o diagrama de classes atualizado para o problema de expansão.

Figura 6: Diagrama de classes e relações da rede implementada.



Fonte: Elaborado pelo autor

3.4 CLUSTERIZAÇÃO E K-MEANS

A clusterização consiste em uma técnica de análise de dados que tem o intuito de agrupar um conjunto de dados (ou objetos) de forma que objetos de um mesmo *cluster* tenham mais características similares entre si, quando comparados a objetos de outro *cluster* (Oliveira, 2008).

Partindo de um conjunto de dados $\boldsymbol{X} = \{x_i\}_{i=1}^{n}$ com n pontos num espaço de dimensão x, define-se um cluster $\boldsymbol{C} = \{\boldsymbol{C}_1, \boldsymbol{C}_2, \ldots . \boldsymbol{C}_k\}$ desse conjunto de dados. Para cada cluster $\boldsymbol{C_i}$ define-se um ponto z_i que representa esse *cluster*, e é chamado de centroide. Um método eficiente para realizar a clusterização de dados é o algoritmo *K-means*. Na Figura 7, há exemplos de clusterização.

Figura 7: Exemplo de clusterização

Fonte: Soares de Oliveira, Bitencourt, (2008)

Com base em um conjunto de dados $\boldsymbol{X}$, esse algoritmo inicia suas iterações distribuindo aleatoriamente os componentes desse conjunto em k *clusters*, definindo os centroides a partir da média dos pontos do cluster $\boldsymbol{C_i}$. Finalizada essa primeira iteração, o algoritmo passa a atribuir *clusters* aos dados e a atualizar os centroides. Durante a fase de atribuição de *clusters*, cada ponto $x_i \in \boldsymbol{X}$ é associado ao *cluster* cujo centroide z_i está mais próximo, definido em (11).

$$x_i \in \boldsymbol{C}_{j*} \; se \; j^* = argmin_{i=1,\ldots,k} \{||x_i - z_i||_2^2\} \tag{11}$$

Atribuídos os *clusters*, o algoritmo atualiza a posição do centroide através da média de todos os componentes do *cluster* $\boldsymbol{C_i}$. Essas etapas são realizadas iterativamente até o algoritmo alcançar um mínimo local, ou seja, a posição dos centroides não se atualizar após uma iteração (Fonseca & Beltrame, 2010).

3.5 HEURÍSTICA CONSTRUTIVA

Trazendo o foco para o tema do presente trabalho, a diversidade de fatores que influenciam a tomada de decisão para a construção de novas subestações torna essa tarefa difícil que requer análises criteriosas sobre diversos fatores, tais como custo, localização ótima, capacidade da subestação, perfil da rede, disponibilidade de alimentadores (Alves, et al.,2002). Desta forma, ferramentas computacionais baseadas em técnicas de otimização e metaheurísticas desempenham um papel importante nesse planejamento, fornecendo dados para embasar a tomada de decisão, reduzindo custos intrínsecos ao processo e tempo despendido na operação.

Neste trabalho o planejamento é realizado usando uma heurística construtiva, em que são considerados os custos atrelados a expansão dos sistemas de distribuição: os custos das perdas ativas na rede em (02), o custo da construção de novos alimentadores em (03) e o custo para construção de novas subestações em (04).

Assim, após a execução do algoritmo de fluxo de potência, são calculadas as perdas nos alimentadores como o somatório das perdas em cada linha de distribuição. No Quadro 1, tem-se o pseudocódigo para essa rotina da classe Rede.

Quadro 2: Pseudocódigo do cálculo do custo das perdas de potência na rede

```
1 – Procedimento CalculateCPerdas ()
2 -     PARA i < N° de ramos -1 FAÇA
3-          Perda de potência (i) ← 0
4 -         PARA l < 3  FAÇA
5 -             PARA c <3  FAÇA
6 -                 Perda de potência (i) ←Perda de potência(i) + |Impedância do ramo(i)| * |J(i)|²
7 -             FIM PARA
8 -         FIM PARA
9 -         Perdas totais ← Perda totais + Perda de potência (i) * 0,001
10 -    FIM PARA
11 -    PARA j < 20 FAÇA
12 -        Correção = Correção + 1.1² * Perdas totais
13 -    FIM PARA
14 -    Custo das perdas de potência ← 8760 * Correção
15 – FIM CalculateCPerdas
```

Fonte: Elaborado pelo autor

Considerando a existência das impedâncias dos cabos e demais componentes de uma rede de distribuição, é fato que a distância entre os alimentadores influência nas perdas de potência na rede elétrica. Sendo assim, um dos critérios adotados para determinar a localização ótima para a inserção de uma nova unidade geradora é a distância dessa localização para o centroide do cluster em que se pode construir a subestação candidata, nesse estudo nomeado como "*cluster zero*".

Assim é necessário, então, a criação de um procedimento para determinar qual, dentre as localizações das subestações candidatas, melhor atende a esse critério. NoQuadro 3, é apresentado o pseudocódigo de determinação da melhor localização dentre as subestações candidatas.

Para possibilitar a análise do impacto da inserção da nova subestação nas perdas ativas das barras pertencentes ao *cluster zero*, é necessário determinar o ramo em que ocorre a mudança de *clusters*, ramo que será denominado "*ramo de desconexão*". Este é o ramo de ligação entre os alimentadores vizinhos. No caso, o alimentador existente é divido em dois alimentadores fornecidos por subestações diferentes, logo é necessário identificar a fronteira entre esses alimentadores vizinhos de modo que a perda seja a menor possível em ambos os alimentadores. O pseudocódigo no Quadro 4 permite a determinação destes ramos de ligação, ou desconexão que separa o novo alimentador do existente.

Quadro 3: Pseudocódigo para determinar a localização da subestação.

1 – **Procedimento** ClusterSE()
2 - cordSEcon ← coordenadas da subestação conectada
3- distância 1 ← $\sqrt{(x_{cordSEcon} - x_{Centroide\ cluster\ 0})^2 + (y_{cordSEcon} - y_{Centroide\ cluster\ 0})^2}$
4 - distância 2 ← $\sqrt{(x_{cordSEcon} - x_{Centroide\ cluster\ 1})^2 + (y_{cordSEcon} - y_{Centroide\ cluster\ 1})^2}$
5 - **SE** distância 1 > distância 2
6 - Cluster da nova subestação = 0
7 - **SE NÃO**
8 - Cluster da nova subestação = 1
9 - **FIM SE**
10 - **PARA** i < N° de subestações não conectadas a rede – 1 **FAÇA**
11 - Subestação.distância (i) ← $\sqrt{(x_{subestação\ (i)} - x_{Centroide\ cluster\ 0})^2 + (y_{subestação(i)} - y_{Centroide\ cluster\ 0})^2}$
12 - **FIM PARA**
13 - Lista das distâncias das subestações ← Subestação.distância em ordem decrescente
14 - Menor distância ← Lista das distâncias das subestações (0)
15 - **PARA** i < N° de subestações não conectadas a rede – 1 **FAÇA**
16 - **SE** Subestação.distância (i) = Menor distância
17 - Subestação ótima = Subestação(i)
18 - **FIM SE**
19 - **FIM PARA**
20 – **FIM** ClusterSE()

Fonte: Elaborado pelo autor

O método é iniciado com a definição da variável local *cordSEcon* que recebe o valor das coordenadas X e Y da posição da subestação conectada a rede. Em seguida as variáveis locais *distância 1* e *distância 2* recebem os valores das distâncias entre a subestação conectada e o centroide do cluster 0 e centroide do cluster 1, respectivamente. Adiante, esses valores são comparados através de uma estrutura condicional. No caso do valor da *distância 1* for maior que o valor da *distância 2*, o cluster da nova subestação é definido como zero. Caso contrário, o cluster da nova subestação é definido como um. Essa condição visa atender o critério adotado para inserção da nova subestação, que é inseri-la no cluster oposto ao cluster da subestação existente, ou seja, no cluster cuja variável local distância é maior. Em seguida é iniciada uma estrutura de repetição que percorre todo o conjunto composto pelas possíveis novas subestações, calculando o valor da distância da posição dessas subestações com o centroide do cluster da nova subestação. Esses valores são armazenados no objeto *distância* da classe *Subestação*, que é uma lista.

Após o algoritmo percorrer todo o conjunto de possíveis novas subestações, a lista distância é ordenada em ordem crescente, de forma que a menor distância entre o centroide do cluster da nova subestação e as posições das candidatas a nova subestação ocupe a primeira posição da lista *distância*. A variável local *menor distância* recebe o valor da primeira posição da lista, e na sequência é iniciada a estrutura da repetição que percorre todo o conjunto de possíveis novas subestações. Dentro desta estrutura de repetição é incluída uma estrutura condicional que define a subestação ótima, utilizando como base a comparação entre o valor da variável local *menor distância* e o valor do objeto *distância* de cada subestação ao longo das iterações.

Quadro 4: Pseudocódigo para a determinação do ramo de desconexão

```
1 – Procedimento DescobreDesconec(int position)
2 -  ramoconec ← Ramo de conexão da nova subestação
3-   ramoauxiliar ← Ramo para do Ramo de conexão da nova subestação
4 - clusteratual ←Cluster da Barra De do ramo auxiliar
5 - ENQUANTO Ramo montante do ramo auxiliar != null e Cluster da Barra De do ramo auxiliar = clusteratual FAÇA
6 -      clusteratual ← Cluster da Barra De do ramo auxiliar atual
7 -      ramoauxiliar.visitado ←true // Parâmetro do objeto do tipo Ramo
8 -      ramoauxiliar ← Ramo montante do ramo auxiliar atual
9 -  FIM ENQUANTO
10 - SE position = 0 FAÇA // Ramo de desconexão é o ramo onde ocorre a mudança de cluster
11 -     ramodesconec ← ramoauxiliar
12 – FIM SE
13-  SE position = 1 FAÇA // Ramo de desconexão alterado para dois ramos a frente de onde ocorre a mudança de clusters
14 -     PARA i < 2 FAÇA
15 -          Cluster da Barra Para do ramo auxiliar ← Cluster da Barra De do ramo auxiliar
16 -          ramodesconec ← ramoauxiliar.Barra_Para.Ramos_Jusante.Where(r => r.visitado).First()
17 -          ramoauxiliar ← ramodesconec
18 -     FIM PARA
19 - FIM SE
20 -  SE position = 2 FAÇA // Ramo de desconexão alterado para dois ramos atrás de onde ocorre a mudança de cluster
21 -          ramodesconec ← ramoauxiliar.Barra_De.Ramo_Montante.Barra_De.Ramo_Montante
22 - FIM SE
23 – FIM DescobreDesconec(int position)
```

Fonte: Elaborado pelo autor

O método é iniciado com a definição da variável local ramoconec que recebe o ramo que faz a conexão da nova subestação com a topologia de rede. Após isso é definida a variável local ramo auxiliar que recebe o valor do ramo para do ramo de conexão, um ramo já pertencente a topologia de rede. A variável cluster atual recebe o cluster da barra De do ramo auxiliar. Em seguida é iniciada a estrutura de repetição While, com condição de ser executada até que ocorra a mudança do cluster da Barra De do ramo auxiliar. Dentro da estrutura de repetição a variável local *cluster atual* tem seu valor alterado para o cluster da barra De do ramo auxiliar da iteração, o ramo é marcado como visitado e o valor do variável *ramo auxiliar* é alterado para o ramo a montante do ramo auxiliar da iteração atual.

Esse processo garante que a rede seja percorrida em direção as Barras do cluster da subestação existente, de forma que quando a condição de mudança de cluster for satisfeita, a variável *ramo auxiliar* armazenará exatamente o ramo onde ocorre a mudança de cluster. O método possui um argumento *position* do tipo int, que altera a posição do ramo de desconexão conforme apresentado abaixo:

- *Position = 0 → ramodesconec* recebe o ramo onde ocorre a mudança de cluster;
- *Position = 1 → ramodesconec* recebe dois ramos a jusante do ramo onde ocorre a mudança de cluster;
- *Position = 2 → ramodesconec* recebe dois ramos a montante do ramo onde ocorre a mudança de cluster.

A posição desse ramo é utilizada para o seccionamento da topologia de rede, tornando possível a análise do impacto da inserção da nova subestação nas percas de potência da rede composta pelas barras posicionadas mais a extremidade da topologia de rede original.

Após a determinação do ramo de desconexão, uma nova estratégia efetua a atribuição de barras aos novos alimentadores em duas etapas:

3.5.1 Isolamento das barras para alteração dos *clusters*: Para isso, partindo da barra raiz, implementou-se a busca em profundidade (DFS) para percorrer as barras associadas ao *cluster 1*. Durante esse processo, o algoritmo efetua a alteração dos *clusters* dessas barras para 2, até queo ramo de desconexão seja identificado, conforme pseudocódigo no Quadro 5;

3.5.2 Ajuste dos *clusters* nas barras da topologia original: Em seguida, todasas barras da topologia são percorridas e aquelas atribuídas ao *cluster* um, tem seu cluster alterado para zero, conforme o pseudocódigo no Quadro 6.

Quadro 5: Pseudocódigo para isolar as barras que devem sofrer alteração de *cluster,*

```
1 – Procedimento ChangeCluster()
2 -    Procedimento DFS(ramo atual)
3-        SE Número da Barra Para do ramo atual = Número da Barra Para do ramo de desconexão FAÇA
4 -            Retorne
5 -        FIM SE
6 -        Cluster da Barra Para do ramo atual ← 2
7 -        PARA i < N° de Ramos Jusante da Barra Para do Ramo atual FAÇA
8 -            DFS(ramoatual.Barra_Para.Ramos_Jusante[i])
9 -        FIM PARA
10 - FIM DFS(ramo atual)
11 – cluster Barra Raiz ←2
12 - PARA i < N° de ramos jusante da Barra raiz FAÇA
13 -       DFS(Barra_Raiz.Ramos_Jusante[i])
14 - FIM PARA
15 – FIM ChangeCluster()
```

Fonte: Elaborado pelo autor

Quadro 6: Pseudocódigo para ajuste dos *clusters* na topologia original.

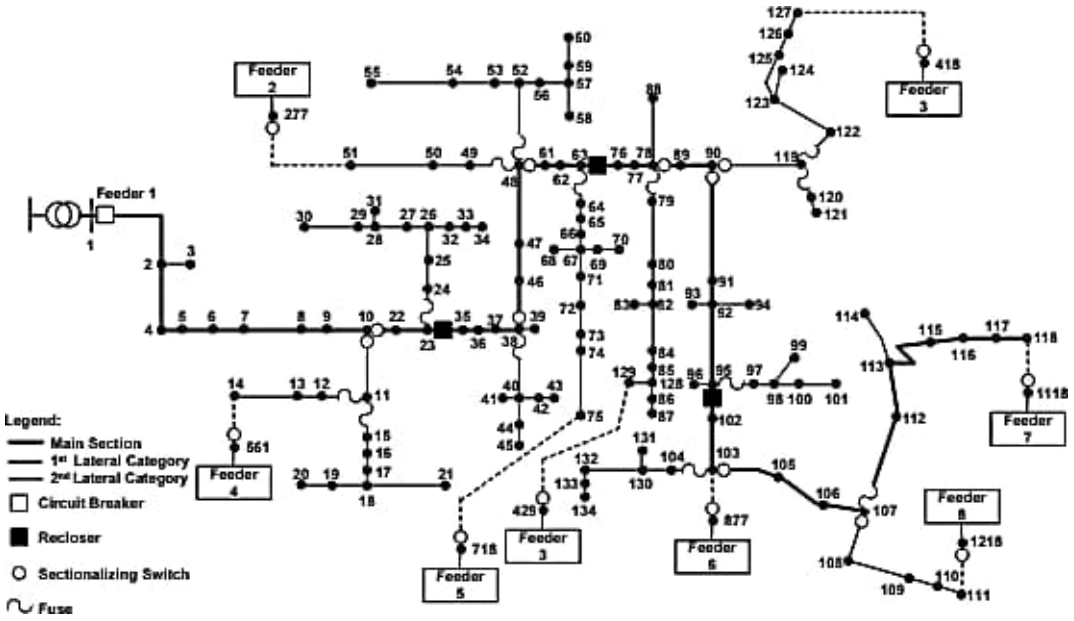
```
1 – Procedimento ChangeCorrection()
2 -        PARA i < N° de Barras FAÇA
3 -            SE cluster da  barra(i) = 1 FAÇA
4 -                cluster da barra(i) ← 0
5 -            FIM SE
6 -        FIM PARA
7- FIM ChangeCluster()
```

Fonte: Elaborado pelo autor

4 RESULTADOS

A fim de encontrar uma solução de boa qualidade, que minimize as várias parcelas dos custos atrelados a expansão de uma rede de distribuição, implementou-se a heurística construtiva que foi avaliada sob uma rede de distribuição com 136 barras, alimentada por uma subestação (barra raiz), que fornece energia em 13,8kV para a rede de distribuição com topologia apresentada na Figura 8.

Figura 8: Topologia da rede com 136 barras

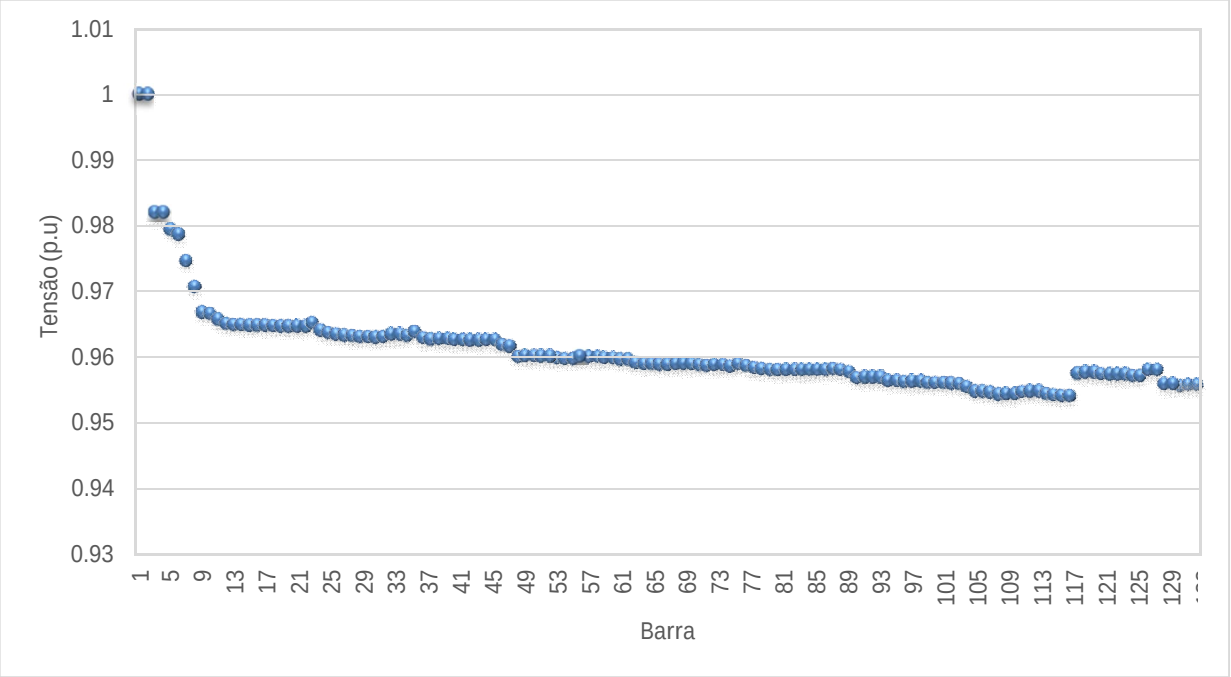


Fonte: LAPSEE (2017)

Para a implementação da heurística construtiva, foi utilizado o Microsoft Visual Studio 2022® e estruturada na linguagem C#. Neste ambiente, foi implementada o modelo da rede de distribuição conforme o diagrama de classes apresentado na Figura 6. A lógica completa pode ser observada no apêndice B.

Inicialmente, foi realizada a leitura dos dados da rede, para a criação das variáveis que são objetos desse estudo. Feito isso, executou-se o método *calculatePF()* para obter o resultado do fluxo de potência na rede, a fim de determinar as grandezas elétricas em cada uma das barras e ramos. Com os valores de tensão nas barras obtidos, foi traçado o perfil de tensão da rede. Na Figura 9, nota-se uma queda expressiva de tensão, à medida que a localização das barras se distancia da barra raiz, ou subestação.

Figura 9: Perfil de tensão de rede original



Fonte: Elaborado pelo autor

4.1 CUSTO DAS PERDAS DE POTÊNCIA E CLUSTERIZAÇÃO

No cálculo do custo das perdas, conforme (02), tem como valores base;

- $T \rightarrow Total\ de\ horas\ em\ um\ ano$ = 8760
- $J^{ENERGIA} \rightarrow Taxa\ de\ juros\ da\ energia$ = 0.1
- $n \rightarrow Número\ de\ anos$ = 20

A Tabela 1 apresenta os valores obtidos para a rede existente.

Tabela 1: Perdas de energia e seu custo para a rede

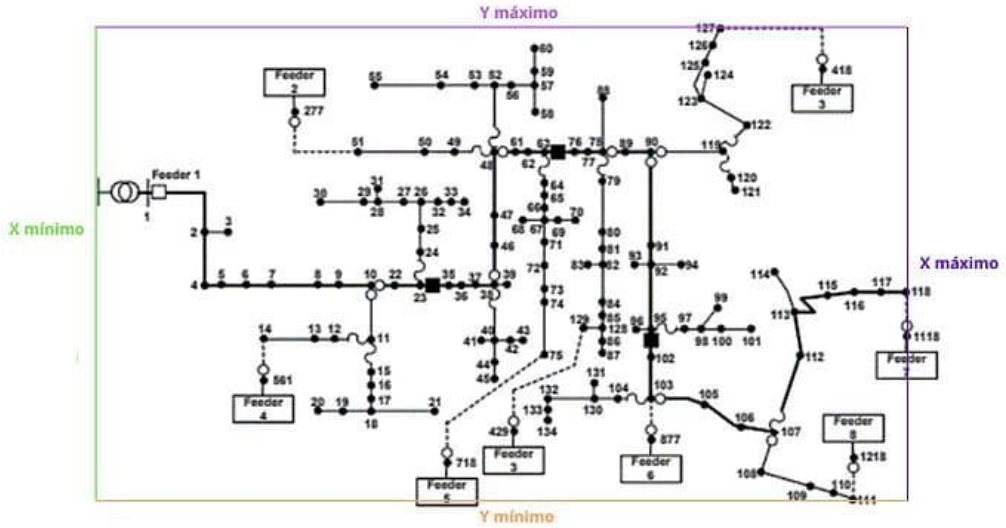
Perdas de energia	302,608 MWh
Custo das perdas de potência	R$ 6.206.233,52

Fonte: Elaborado pelo autor

Nota-se o alto valor de perdas nessa rede, e seu grande impacto financeiro. Visando atingir o objetivo desse estudo, a redução de custos atrelados a expansão da rede, fez-se necessário determinar a melhor posição para a inserção de uma nova subestação. Partindo das premissas que a rede está sendo analisada no espaço bidimensional, e cada barra possui uma posição x_i e uma posição y_i , foram

determinados os valores mínimos e máximos das posições das barras nos eixos coordenados, de forma que toda a rede fosse inserida dentro de um plano retangular, conforme representado na Figura 10.

Figura 10: Espaço cartesiano envolvendo a topologia da rede



Fonte: Adaptado pelo autor

Em seguida, a rede foi dividida em dois *clusters*, C_0 e C_1, utilizando o algoritmo K-means. Os centroides adotados para inicializar o algoritmo constam na Tabela 2.

Tabela 2: Posição inicial dos centroides para os *clusters* C_0 e C_1

Cluster	Posição Inicial do centroide
1	$0.25 * (X_{máximo} – X_{mínimo}); 0.5 * (Y_{máximo} – Y_{mínimo})$
0	$0.75 * (X_{máximo} – X_{mínimo}); 0.5 * (Y_{máximo} – Y_{mínimo})$

Fonte: Elaborado pelo autor

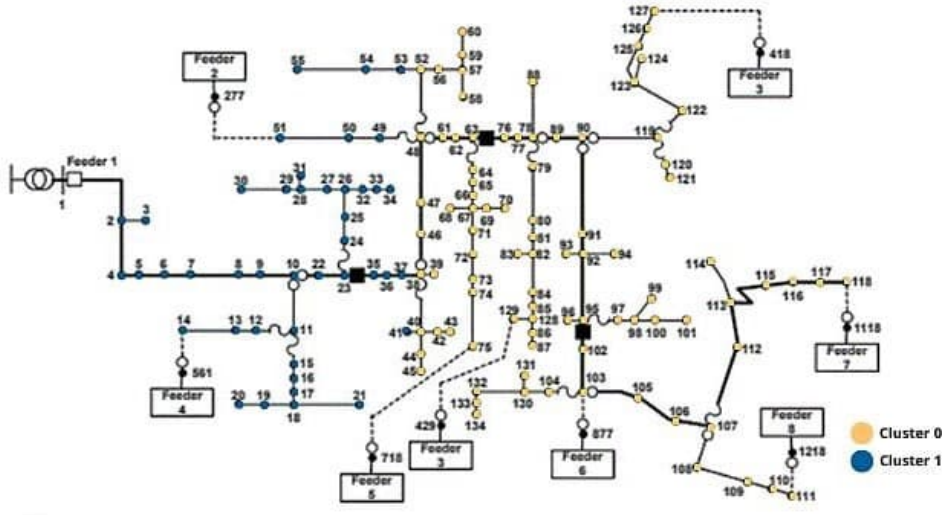
O processo iterativo do K-means é executado até a posição dos centroides parar se de alterar, atingindo um mínimo local e definindo os *clusters*, que estão representados na Figura 11.

4.2 ALOCAÇÃO DE SUBESTAÇÃO

Visando reduzir a perda de potência nesse sistema, é proposto que uma nova subestação seja incluída nessa rede. Nota-se na Figura 10 a indicação de oito

alimentadores vizinhos, as quais são as possíveis posições para a inserção dessa nova subestação, sendo essas posições elencados na Tabela 3.

Figura 11: Divisão da rede por *clusters*



Fonte: Adaptado pelo autor

Tabela 3: Novos alimentadores

Alimentador	Barra	Barra de conexão
2	277	51
3	418	127
4	561	14
5	718	75
6	877	103
7	1118	118
8	1218	111
9	429	129

Fonte: Elaborado pelo autor

O perfil de tensão na Figura 9 revela a existência de uma redução gradual da tensão à medida que as barras se afastam da subestação que alimenta a rede (barra raiz). A disposição dos *clusters*, organizados com base na distância entre as posições das barras em relação aos centroides, evidência que existe um grupo ótimo de barras localizadas no *cluster* oposto a subestação existente da rede (barra raiz), de forma que a inserção de uma nova subestação em uma das posições disponíveis nesse *cluster* é uma boa estratégia para equilibrar o perfil de tensão e garantir um fornecimento de energia eficiente às barras que compõe esse *cluster* e possivelmente reduzir as perdas do sistema.

Após a execução da rotina descrita no pseudocódigo do Quadro 2, foi identificada a posição ótima para a inserção da nova subestação. Essa posição é correspondente à localização do alimentador 6 - barra 877, na Tabela 3. Além disso, estabeleceu-se que o custo associado à construção da nova subestações, C^{SE}**,** é de R$ 1.000.000,00.

No pseudocódigo no Quadro 3, as estruturas condicionais entre as linhas 10 e 22 do algoritmo têm o propósito de modificar a posição do ramo de desconexão com base no argumento fornecido à função. A posição desse ramo é um fator que impacta diretamente a quantidade de perdas ativas na rede, e é considerado como um dos critérios para a escolha da solução ótima do problema em estudo. A princípio utiliza-se o ramo onde ocorre a mudança de *clusters* como ramo de desconexão.

Tabela 4: Ramo de desconexão determinado pela rotina no Quadro 3.

Ramo	Barra De	Barra Para
38	38	39

Fonte: Elaborado pelo autor

A teoria de grafos pressupõe que “subgrafos” tem as mesmas características do grafo original, de forma que a divisão da topologia de rede original em duas topologias, tendo como ponto de divisão entre elas o “ramo de desconexão” anteriormente determinado, possibilita verificar o impacto da inserção da nova subestação no sistema criado pelas barras que devem ser alimentadas pela nova subestação, sem que estas percam as suas características advindas da rede original. Contudo, ao observar a Figura 11, nota-se que existem barras pertencentes ao “*cluster* um” que após a desconexão do ramo supracitado passam a pertencer ao “*cluster* zero” devido à formação de subgrafos conexos.

Tabela 5: Barras que devem ter seu cluster alterado

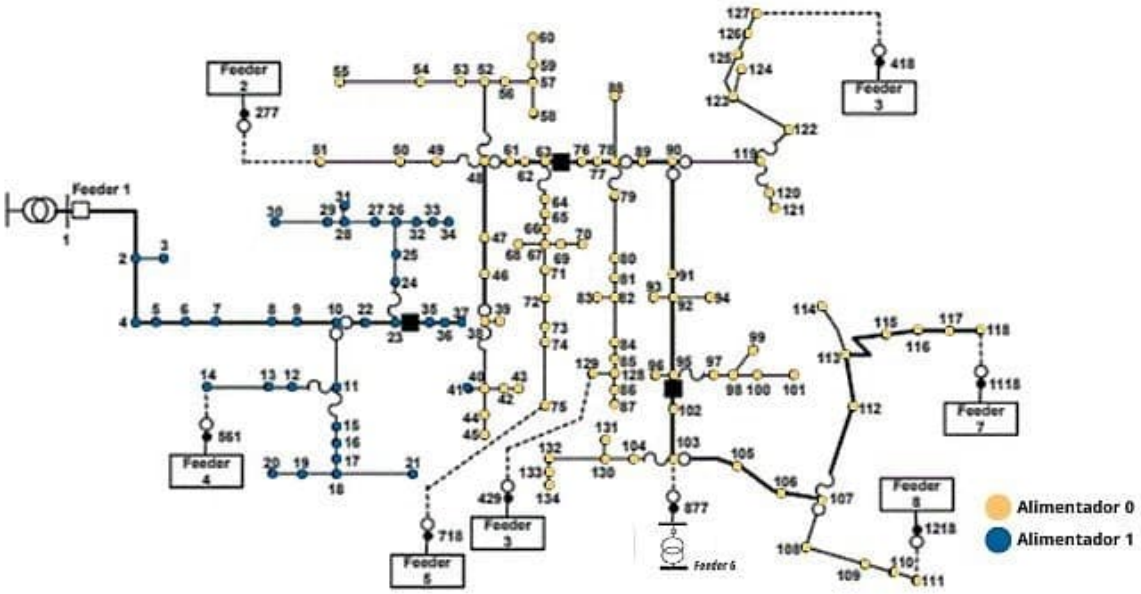
Barra
49
50
51
53
54
55

Fonte: Elaborado pelo autor

Após essa adequação dos *clusters*, com a execução das rotinas descritas nos pseudocódigos nos Quadros 4 e 5, a topologia da rede original é dividida em duas redes menores.

- Alimentador 1: Barras alimentadas pela subestação original;
- Alimentador 0: Barras alimentadas pela nova subestação.

Figura 12: Topologia após inserção da nova subestação



Fonte: Elaborado pelo autor

4.3 CABOS

Uma vez determinada a melhor posição para a inserção da nova subestação, e realizados os devidos arranjos na topologia da rede, é necessário determinar qual o tipo de cabo deve ser utilizado para compor o novo circuito (ramo de ligação da novasubestação). Na Tabela 6 são apresentados os tipos de cabos disponíveis.

Tabela 6: Tipos de cabos disponíveis e suas impedâncias

Tipo de cabo	Impedância própria
#0-1	0,29 + 1,92j
#2	1,084 +0,998j
#4	1,644 + 1,006j
#1/0	0,7567 + 1,006j
#4/0	0,4272 + 0,9609j

Fonte: Elaborado pelo autor

Além do aspecto financeiro, a seleção do tipo de cabo a ser utilizado no circuito de conexão é fundamentada no critério da impedância própria do cabo. Quanto menor a impedância, menores são as perdas de potência ativa no sistema causadas pela resistência elétrica encontrada durante a transmissão de energia ao longo desse cabo. Sendo assim, dois tipos de cabo se destacam como bons candidatos a serem aplicados nesse trecho, o #0-1 e o #4/0, devido a seus baixos valores de impedâncias. Considerando o comprimento do trecho que compõe o circuito de conexão igual a 0,0461 quilômetro, na Tabela 7 são apresentados os possíveis valores para o custo de construção de novos circuitos, C^{CCT}.

Tabela 7: Possíveis valores para o custo de construção de novos alimentadores

Tipo de cabo	Custo/km	C^{CCT}
#0-1	R$ 20.000,00	R$ 922,00
#4/0	R$40.000,00	R$ 1.844,00

Fonte: Elaborado pelo autor

4.4 SOLUÇÃO ÓTIMA PARA A REDUÇÃO DOS CUSTOS TOTAIS

Neste estágio, torna-se factível a determinação das três parcelas que compõem os custos da expansão da rede em análise, as quais são representadas por (02), (03) e (04). Desta maneira, viabiliza-se a determinação da solução ótima para o problema em estudo. A fim de alcançar este objetivo, foram conduzidos testes para avaliar o impacto da alteração da posição do ramo de desconexão e do tipo de cabo utilizado no circuito de conexão nas perdas ativas da rede. Os melhores resultados obtidos são apresentados nas próximas subseções e as demais soluções constam no Apêndice A.

4.4.1 Caso 1

Tabela 8: Configuração da rede no caso 1.

Posição da nova subestação	Tipo de cabo do circuito de conexão	N° do ramo de desconexão
877	#4/0	39

Fonte: Elaborado pelo autor

Tabela 9: Perdas de potência ativa na rede e seu custo para o caso 1.

Alimentador	0	1	Rede (Total)
Perdas de energia (kWh)	18,77	24,41	43,18
C^{PERDAS}	R$ 384.888,51	R$ 500.720,96	R$ 885.609,47

Fonte: Elaborado pelo autor

Sendo assim, o C^{TOTAL}, representado por (1), para essa configuração é de R$1.887.453,47, representado uma redução de 69,59% em relação ao valor do custo das perdas ativas na rede original.

4.4.2 Caso 2

Tabela 10: Configuração da rede no caso 2

Posição da nova subestação	Tipo de cabo do circuito de conexão	N° do ramo de desconexão
877	#0-1	39

Fonte: Elaborado pelo autor

Tabela 11: Perdas de potência ativa na rede e seu custo para o caso 2

Alimentador	0	1	Rede (Total)
Perdas de energia (kWr)	18,78	24,41	43,19
C^{PERDAS}	R$ 384.990,02	R$ 500.720,96	R$ 885.710,98

Fonte: Elaborado pelo autor

Nessa configuração, o valor do C^{TOTAL} é de R$ 1.886.632,98, refletindo uma redução de 69,60% em comparação ao valor do custo das perdas ativas na rede original.

Na comparação entre esses dois casos, onde a única distinção reside no tipo de cabo utilizado, fica evidente que optar pelo cabo #4/0 no circuito de conexão é a escolha mais eficiente para reduzir as perdas de potência no sistema, devido à sua menor impedância em comparação com o cabo #0-1. No entanto, é importante salientar que a análise deste estudo está centrada no aspecto financeiro da expansão dos sistemas de distribuição. Ao comparar os custos totais dos dois casos, observa-se que a utilização do cabo #0-1 é mais vantajosa devido ao seu menor custo de

aplicação, mesmo que seu impacto na redução das perdas ativas do sistema seja ligeiramente menor.

4.4.3 Caso 3

Tabela 12: Configuração da rede no caso 3.

Posição da nova subestação	Tipo de cabo do circuito de conexão	N° do ramo de desconexão
877	#0-1	48

Fonte: Elaborado pelo autor

Tabela 13: Perdas de potência ativa na rede e seu custo para o caso 3

Alimentador	0	1	Rede (Total)
Perdas de energia (kWh)	14,55	33,89	48,44
C^{PERDAS}	R$ 298.495,11	R$ 695.035,91	R$ 993.531,02

Fonte: Elaborado pelo autor

Observa-se que a modificação da posição do ramo de desconexão para dois ramos adiante da posição original (ramo 39) tem um impacto negativo nas perdas de potência ativa do sistema, resultando em um aumento de aproximadamente 5,2 kW em relação ao caso anterior. Ao analisar o C^{TOTAL} para esta configuração, que é de R$1.994.453,02, verifica-se uma redução de 67,86% no custo das perdas ativas em comparação com a rede original. Isso evidencia que esta disposição da rede não é a mais adequada para minimizar os custos relacionados à expansão do sistema, uma vez que os dois casos previamente apresentados demonstraram maior eficácia nessa redução.

4.4.4 Caso 4

Avaliando o valor das perdas de potência ativa desse caso, e somando seu custo as demais parcelas que compõe o C^{TOTAL} , obteve-se o valor de R$

1.862.921,51. Esse resultado reflete uma redução de 69,98% em comparação com os custos das perdas ativas do sistema original.

Tabela 14: Configuração da rede no caso 4.

Posição da nova subestação	Tipo de cabo do circuito de conexão	N° do ramo de desconexão
877	#0-1	37

Fonte: Elaborado pelo autor

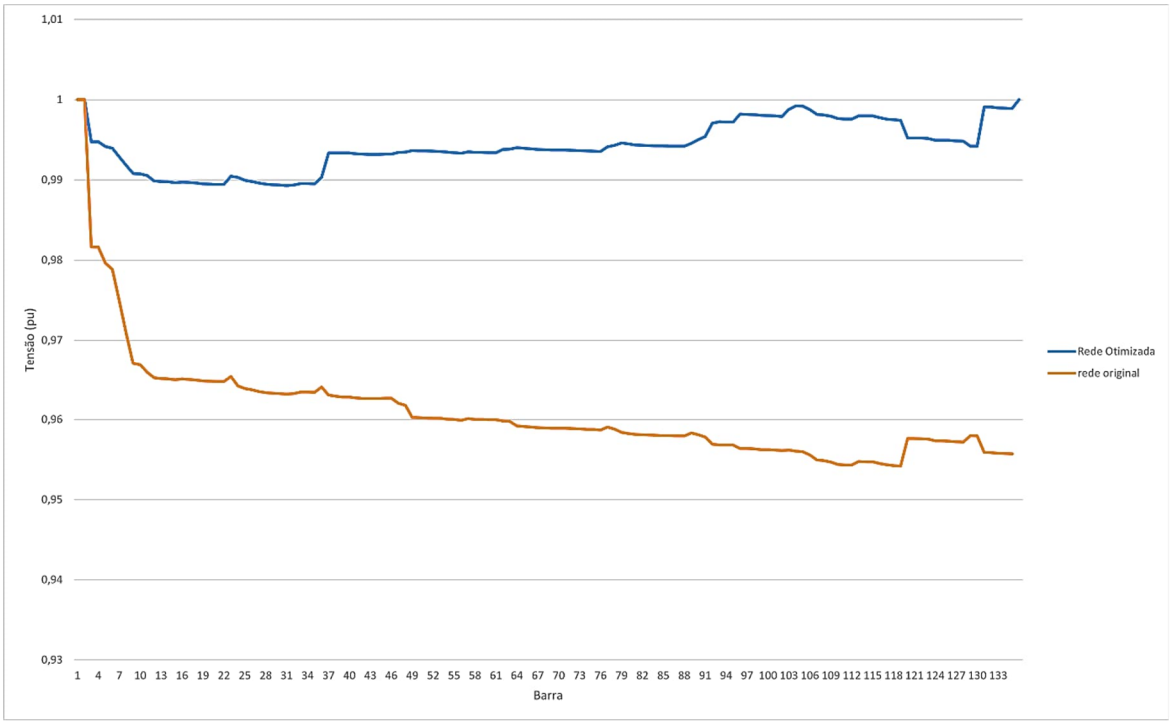
Tabela 15: Perdas de potência ativa na rede e seu custo para o caso 4

Alimentador	0	1	Rede (Total)
Perdas de energia (kWh)	20,29	21,74	42,03
C^{PERDAS}	R$ 416.052,80	R$ 445.946,71	R$ 861.999,51

Fonte: Elaborado pelo autor

Na Figura 13, é importante destacar uma melhora no perfil de tensão da rede.

Figura 13: Comparativo dos perfis de tensão (caso 4 x rede original)



Fonte: Elaborado pelo autor

Em relação ao perfil de tensão da rede original, conforme ilustrado na Figura 15, observa-se um impacto positivo desta configuração nos valores das tensões nas barras. Na rede otimizada, as tensões oscilam entre 0,989 pu e 1 pu, indicando um perfil de tensão equilibrado, o que resulta em uma distribuição de energia mais eficiente e se correlaciona diretamente com a substancial redução das perdas de potência ativa, evidenciada pelos valores apresentados na Tabela 15. Esta melhoria demonstra não apenas a estabilidade, mas também a otimização do sistema de distribuição, refletindo positivamente na operação e no desempenho global da topologia analisada. Esses resultados reforçam que essa configuração da rede representa uma solução satisfatória para a redução dos custos associados à expansão deste sistema de distribuição de energia.

5 CONCLUSÃO

A análise detalhada apresentada no decorrer da subseção 4.4 demonstra que a introdução de uma nova subestação em um sistema de distribuição existente pode ter um impacto significativamente positivo na redução dos custos operacionais associados à rede. No entanto, para maximizar esses benefícios é crucial que a decisão de construir novos alimentadores seja fundamentada em estudos e análises sólidas que evidenciem sua real necessidade e impacto. A aplicação de métodos computacionais para otimizar o planejamento financeiro dessa operação emerge como uma estratégia eficaz e essencial. Este estudo destaca a importância desses métodos, dada a complexidade dos fatores envolvidos na expansão dos sistemas de distribuição.

Os resultados revelam que a expansão do sistema acarretara uma redução de 69,98% nos custos totais associados à operação deste na configuração existente, validando a eficácia do método desenvolvido para atingir o objetivo deste estudo. Contudo, é importante ressaltar que existem diversas otimizações que podem ser implementadas para aprimorar o método desenvolvido, tais como a automação para a identificação da solução ótima, bem como a incorporação de outros critérios relevantes nas análises de expansão de redes de distribuição, a fim de permitir a aplicabilidade e eficiência deste método em sistemas reais.

REFERÊNCIAS

WILLIS; H. LEE.; ABB.Inc (org.). **Power Distribution Planning Reference Book**. 2.ed. Raleigh, North Carolina: Editora Marcel Dekker,Inc, 2004. 1212 p.

PORKAR, S., ABBASPOUR-TEHRANI-FARD, A., POURE, P., SAADATE, S., 2011, "*Distribution system planning considering integration of distributed generation and load curtailment options in a competitive electricity market*", **Electrical Engineering Journa**l, vol. 93, n. 1, pp. 23-32.

KHODR, H. M., VALE, Z., RAMOS, C., FARIA P., 2009, "*Optimization techniques for power distribution planning with uncertainties: A comparative study*", **IEEE Power & Energy Society General Meeting PES**. July.

SHOJAEIAN, S., GHANDEHARI, E., "*A heuristic multiobjective method for radial distribution networks reconfiguration*", 2013, **Chinese Journal of Engineering**, v. 2013, ID 654074. 4 pages.

SINGH, K. J. *Electricity distribution network expansion planning*. **Department of Engineering Science, The University of Auckland**, 2004

AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA (ANEEL), **"Procedimentos de Distribuição de Energia Elétrica no Sistema Elétrico Nacional (PRODIST) – Módulo 1: Glossário de termos técnicos do PRODIST"**.

Frotin, Sergio. O.; **Equipamentos de alta tensão: Prospecção e Hierarquização de Inovações Tecnológicas.** 1 ed. Brasília, Tiragem 2.000 livros, 2013. 934p

NEGROMONTE, VINICIUS A. M.; **Estudo do Sistema de Proteção de uma subestação.** 2018. 49p. Dissertação (Graduação em Engenharia Elétrica) – Universidade Federal de Campina Grande, Campina Grande, 2018.

MARTINS, Caio César Costa et al. **FLUXO DE CARGA EM REDES DE DISTRIBUIÇÃO OPERANDO EM EMERGÊNCIA.** 2018. 109 p. Dissertação (Pós-Graduação em Engenharia de Eletrecidade) – Centro de Ciências Exatas e Tecnológicas, Universidade Federal do Maranhão, São Luís, 2018.

MARCHESAN, Adriano Cavalheiro et al**. Análise de métodos de fluxo de potência por varredura para cálculo do desequilíbrio de tensão em sistemas de distribuição radiais.** In: 11th Seminar on Power Electronics and Control (SEPOC 2018). Brasil, 2023.

TENESACA-CALDAS, Marcelo et al. **Construção de modelos de redes de distribuição a partir de sistemas de informação geográfica utilizando grafos**. Simpósio Brasileiro de Sistemas Elétricos-SBSE, v. 2, n. 1, 2022.

DROZDEK, A. **Data Structures and Algorithms in C++,** Estados Unidos, Cengage Learning, 2012.

OLIVEIRA, Tatyana Bitencourt Soares de. **Clusterização de dados utilizando técnicas de redes complexas e computação bioinspirada**. 2008. Tese de Doutorado. Universidade de São Paulo, 2008.

FONSECA, Felipe Cesar Stanzani; BELTRAME, Walber Antônio Ramos. **Aplicações Práticas dos Algoritmos de Clusterização K-means e Bisecting K-means**. UFES, Vitória, 2010.

ALVES, Arileide Cristina et al. **Algoritmos genéticos aplicados ao planejamento da distribuiçao de energia elétrica em curitiba e regiao metropolitana**. 2002. Tese de mestrado Universidade Federal do Paraná, 2002.

APÊNDICE A – Demais soluções obtidas

Durante a exposição dos resultados no item 4.4, foram abordadas as soluções relacionadas a inserção da nova subestação na localização do alimentador 6, identificada como a alternativa mais eficaz para reduzir as perdas de potência na rede.

Esta seção apresentará as soluções considerando a inserção da nova subestação nas posições dos alimentadores três e cinco, posicionadas, respectivamente, como a segunda e terceira posições mais próximas ao centroide do cluster 0, critério que foi utilizado para determinação da posição ideal para a inclusão da nova subestação.

Tabela 16: Configuração da rede no caso 5

Posição da nova subestação	Tipo de cabo do ramo de conexão	N° do ramo de desconexão
429	4#0	39

Fonte: Elaborado pelo autor

Tabela 17: Perdas de potência ativa na rede e seu custo para o caso 5

Alimentador	Rede (Total)
Perdas de Potência ativa (kW)	57,45
C^{PERDAS}	R$ 1.178.188,55
C^{TOTAL}	R$ 2.180.032,85
% Redução dos custos	64,87

Fonte: Elaborado pelo autor

Tabela 18: Configuração da rede no caso 6

Posição da nova subestação	Tipo de cabo do ramo de conexão	N° do ramo de desconexão
718	4#0	39

Fonte: Elaborado pelo autor

Tabela 19: Perdas de potência ativa na rede e seu custo para o caso 6

Alimentador	Rede (Total)
Perdas de Potência ativa (kW)	68,09
C^{PERDAS}	R$ 1.396.534,00
C^{TOTAL}	R$ 2.398.378,00
% Redução dos custos	61,36

Fonte: Elaborado pelo autor

Os resultados acima evidenciam que a alteração da posição da nova subestação impacta negativamente na quantidade de perdas de potência da rede, corroborando com a escolha da posição do alimentador 6 como a mais apropriada.

APÊNDICE B – Lógica utilizada em C#

```
namespace modeloRD
{

    public class RamoLig: Ramo
    {
        public double Capacity; //Capacidade do ramo de ligação
        public Subestacao SubDE;
    }

    public class Subestacao: Barra
    {

        public bool conec; //e está conectada ou não a rede
        public double dist; //distancia do centroide do cluster que não tem SE

    }

    public class Cabos
   {
        public string Tipo;
        public Complex[][] zCabo;
   }

    public class Barra
    {
        public string Numero; //A identificação da barra (ID)
        public Complex[] S; // criação complex vetor de potencia
        public Complex[] Sg;//Potência fornecida pelo GD, quando houver
        public List<Ramo> Ramos_Jusante = new List<Ramo>(); //Lista dos ramos
jusantes a barra
        public Barra Barra_Montante;
        public Ramo Ramo_Montante;
        public Complex[] V; // Vetor de tensão nas barras
        public Complex[] I; // Vetor de corrente nas barras
        public double Vplus;//Auxiliar a armazenar a maior tensão
        public double Vminus;//Auxiliar a armazenar a maior tensão
        public double x; // utilizado na referência bidensional do k-means
        public double y; //utilizado na referência bidensional do k-means
        public int Cluster;
    }

    public class Ramo
    {
        //Dados de entrada
        public string De; //Barra de origem (From)
        public string Para; // Barra de destino (to)
        public Barra Barra_De; //Origem
        public Barra Barra_Para; //Destino
        public Ramo Ramo_Montante; //Ramo adjacente a montante
        public Complex[] J; // Vetor de corrnetes nos ramos.
```

```
        public Complex[][] zRamo; // Matriz de impedâncias
        public double Pt ; //Perda total no ramo
        public double Im; //Valor da maior corrente no ramo, entre as três fases
        public string Cabo;
        public double Tamanho;
        public bool visitado;
    }

    public class Rede
    {
        //Atributos
        public Barra[] barras; //Vetor com as barras da rede
        public Ramo[] ramos; //Vetor com os ramos da rede.
        public RamoLig[] ramoslig; //Ramos de ligação
        public Ramo ramodesconec; //Ramo de desconexão
        public Barra Barra_Raiz; //Barra de origem ou referência
        public Cabos[] cabos;
        public Subestacao[] subes; //Conjunto de possíveis subestações da REDE
        public Subestacao SEotima1; // Melhor Subestação para cluster 1
        public double Vse = 13800; // Módulo da tensão de referência
        public double erro_max = 1;
        public List<double> difS = new List<double>();
        public double PotPerdas = 0; //Potencia para o calculo do custo de Perdas na
rede
        public double Correcao = 0; // Auxilia no calculo do custo de Perdas na rede
        public double CPerdas; //Armazena o Custo de Perdas na Rede
        public double Vmax;
        public double Vmin;
        public double Imax; //maximo valor na rede
        public double PotSE;
        public double Ise = 0.0;
        public double Xmin;
        public double Xmax;
        public double Ymin;
        public double Ymax;
        public double[][] means;
        //Opção da impressão de dados
        public class outData {

            public bool _3V = true;
            public bool _3I = true;
            public bool difS = true;

        };

        public outData Dado_Saida = new outData();

        //Funções

        public void lerRede(XmlDocument dadoRD)
        {

            XmlNode ramosxml = dadoRD.DocumentElement.SelectSingleNode("Branch");
            XmlNode barrasxml = dadoRD.DocumentElement.SelectSingleNode("Bus");
```

```
            XmlNode caboxml = dadoRD.DocumentElement.SelectSingleNode("Cable");
            XmlNode subxml = dadoRD.DocumentElement.SelectSingleNode("Substation");
            XmlNode ramoligxml =
dadoRD.DocumentElement.SelectSingleNode("LinkBranch");


            //Leitura das subestações
            subes = new Subestacao[subxml.ChildNodes.Count];
            for (int i = 0; i < subes.Length; i++)
            {
                subes[i] = new Subestacao();

                subes[i].Numero = subxml.ChildNodes[i].Attributes["id"].Value;

                subes[i].x =
Convert.ToDouble(subxml.ChildNodes[i].Attributes["pX"].Value);
                subes[i].y =
Convert.ToDouble(subxml.ChildNodes[i].Attributes["pY"].Value);

                subes[i].conec =
bool.Parse(subxml.ChildNodes[i].Attributes["connected"].Value);
            }

            //Leitura dos dados dos cabos
            cabos = new Cabos[caboxml.ChildNodes.Count];
            for (int i = 0; i < cabos.Length; i++)
            {
                cabos[i] = new Cabos();

                cabos[i].Tipo = caboxml.ChildNodes[i].Attributes["size"].Value;

                cabos[i].zCabo = new Complex[3][];
                for (int j = 0; j < 3; j++) cabos[i].zCabo[j] = new Complex[3];

                cabos[i].zCabo[0][0] = new
Complex(Convert.ToDouble(caboxml.ChildNodes[i].Attributes["raa"].Value) ,
Convert.ToDouble(caboxml.ChildNodes[i].Attributes["xaa"].Value));
                cabos[i].zCabo[0][1] = new
Complex(Convert.ToDouble(caboxml.ChildNodes[i].Attributes["rab"].Value) ,
Convert.ToDouble(caboxml.ChildNodes[i].Attributes["xab"].Value)) ;
                cabos[i].zCabo[0][2] = new
Complex(Convert.ToDouble(caboxml.ChildNodes[i].Attributes["rac"].Value) ,
Convert.ToDouble(caboxml.ChildNodes[i].Attributes["xac"].Value));
                //linha fase b
                cabos[i].zCabo[1][0] = cabos[i].zCabo[0][1];
                cabos[i].zCabo[1][1] = new
Complex(Convert.ToDouble(caboxml.ChildNodes[i].Attributes["rbb"].Value) ,
Convert.ToDouble(caboxml.ChildNodes[i].Attributes["xbb"].Value));
                cabos[i].zCabo[1][2] = new
Complex(Convert.ToDouble(caboxml.ChildNodes[i].Attributes["rbc"].Value) ,
Convert.ToDouble(caboxml.ChildNodes[i].Attributes["xbc"].Value));
                // linha da fase C
                cabos[i].zCabo[2][0] = cabos[i].zCabo[0][2];
                cabos[i].zCabo[2][1] = cabos[i].zCabo[1][2];
```

```
                cabos[i].zCabo[2][2] = new
Complex(Convert.ToDouble(caboxml.ChildNodes[i].Attributes["rcc"].Value) ,
Convert.ToDouble(caboxml.ChildNodes[i].Attributes["xcc"].Value));

            }


            //Leitura dos dados das barras
            barras = new Barra[barrasxml.ChildNodes.Count];
            for (int i = 0; i < barras.Length; i++)
            {
                barras[i] = new Barra();
                barras[i].x =
Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["pX"].Value);
                barras[i].y =
Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["pY"].Value);
                barras[i].Numero = barrasxml.ChildNodes[i].Attributes["id"].Value;
                barras[i].S = new Complex[3];
                barras[i].S[0] = new
Complex(Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["pa"].Value) * 1e3,
Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["qa"].Value) * 1e3);
                barras[i].S[1] = new
Complex(Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["pb"].Value) * 1e3,
Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["qb"].Value) * 1e3);
                barras[i].S[2] = new
Complex(Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["pc"].Value) * 1e3,
Convert.ToDouble(barrasxml.ChildNodes[i].Attributes["qc"].Value) * 1e3);
            }

            //Leitura dos ramos de ligação
            ramoslig = new RamoLig[ramoligxml.ChildNodes.Count];
            for (int i = 0; i < ramoslig.Length; i++)
            {
                ramoslig[i] = new RamoLig();

                ramoslig[i].De = ramoligxml.ChildNodes[i].Attributes["from"].Value;
                ramoslig[i].Para = ramoligxml.ChildNodes[i].Attributes["to"].Value;
                ramoslig[i].Capacity =
Convert.ToDouble(ramoligxml.ChildNodes[i].Attributes["Capacity"].Value);

                //Associção do ramo com a barra de destino
                int counter = 0;
                while (barras[counter].Numero != ramoslig[i].Para) counter++;
                ramoslig[i].Barra_Para = barras[counter];

                //Associação com a SE de origem
                counter = 0;
                while (subes[counter].Numero != ramoslig[i].De) counter++;
                ramoslig[i].SubDE = subes[counter];
                ramoslig[i].Barra_De = subes[counter];

                //Definição do tamanho do ramo em KM
```

```
                ramoslig[i].Tamanho = Math.Sqrt(Math.Pow(ramoslig[i].Barra_Para.x -
ramoslig[i].SubDE.x, 2) + Math.Pow(ramoslig[i].Barra_Para.y - ramoslig[i].SubDE.y,
2)) * 1e-3; ;

                //Definição do tipo de cabo
                Cabos caboaux = cabos[0];
                int count = 0;
                //while(count < cabos.Length)
                //{
                //if (cabos[count].zCabo[0][0].Real < caboaux.zCabo[0][0].Real) //Ve
qual cabo tem a menor impedância com base na própria
                //{
                //    caboaux = cabos[count];
                //}
                //count++;
                //}
                //ramoslig[i].Cabo = caboaux.Tipo;
                ramoslig[i].Cabo = cabos[0].Tipo; //Permite alterar os tipos de cabo
pelo indice
                //Calculando a impedância dos ramos de ligação
                Complex[][] zAux = cabos.Where(c => c.Tipo ==
ramoslig[i].Cabo).FirstOrDefault().zCabo;

                ramoslig[i].zRamo = new Complex[3][];
                for (int j = 0; j < 3; j++)
                {
                    ramoslig[i].zRamo[j] = new Complex[3];
                    for (int k = 0; k < 3; k++)
                    {
                        ramoslig[i].zRamo[j][k] = zAux[j][k] * ramoslig[i].Tamanho;
                    }

                }

            }

            //Leitura dos dados dos ramos
            ramos = new Ramo[ramosxml.ChildNodes.Count];
            for (int i = 0; i < ramos.Length; i++)
            {
                ramos[i] = new Ramo();

                ramos[i].De = ramosxml.ChildNodes[i].Attributes["from"].Value;
                ramos[i].Para = ramosxml.ChildNodes[i].Attributes["to"].Value;

                ramos[i].Tamanho =
Convert.ToDouble(ramosxml.ChildNodes[i].Attributes["length"].Value)*1e-3; //Já pega
o tamanho pela base de dados 2

                ramos[i].Cabo = ramosxml.ChildNodes[i].Attributes["cable"].Value;

                Complex[][] zAux = cabos.Where(c => c.Tipo ==
ramos[i].Cabo).FirstOrDefault().zCabo;
```

```
            ramos[i].zRamo = new Complex[3][];
            for (int j = 0; j < 3; j++)
            {
                ramos[i].zRamo[j] = new Complex[3];
                for(int k =0; k<3; k++)
                {
                    ramos[i].zRamo[j][k] = zAux[j][k] * ramos[i].Tamanho;
                }

            }


            //Associção entre o ramo e a barra de origem
            int counter = 0;
            while (barras[counter].Numero != ramos[i].De) counter++;
            ramos[i].Barra_De = barras[counter];

            //Associção do ramo com a barra de destino
            counter = 0;
            while (barras[counter].Numero != ramos[i].Para) counter++;
            ramos[i].Barra_Para = barras[counter];

            //ramos[i].Barra_Para.Barra_Montante = ramos[i].Barra_De;

            ramos[i].Barra_Para.Ramo_Montante = ramos[i];

        }

        //Definição do seu ramo com seu ramo montante
        for (int i = 0; i < ramos.Length; i++)
        {

            for (int j = 0; j < ramos.Length; j++)
            {
                if (ramos[i].De == ramos[j].Para)
                {

                    ramos[i].Ramo_Montante = ramos[j];
                    break;
                }

            }
        }

        //Associação da barra com ramos jusantes
        for (int i = 0; i < barras.Length; i++)
        {

            for (int j = 0; j < ramos.Length; j++)
            {
                if (barras[i].Numero == ramos[j].De)
                {

                    barras[i].Ramos_Jusante.Add(ramos[j]);
```

```
                }

            }
        }

        //Busca da barra raiz da rede
        for (int j = 0; j < ramos.Length; j++)
        {
            if (ramos[j].Ramo_Montante == null)
            {

                Barra_Raiz = ramos[j].Barra_De;
                break;
            }

        }
    } //Leitura dos dados de rede

    public void calculatePF()
    {
        void DFS(Ramo linha)
        {
            //Adicionar as operações de ida.
            for (int i = 0; i < 3; i++)
            {

                linha.Barra_Para.V[i] = linha.Barra_De.V[i]; //Vi = Vu

                for (int j = 0; j < 3; j++)
                {
                    linha.Barra_Para.V[i] -= linha.zRamo[i][j] * linha.J[j];
                }

                //Atualização da diferença de potência

                Complex delta_S = linha.Barra_Para.V[i] *
Complex.Conjugate(linha.Barra_Para.I[i]) - (linha.Barra_Para.S[i] +
linha.Barra_Para.Sg[i]);
                erro_max = Math.Max(erro_max, Math.Max(Math.Abs(delta_S.Real),
Math.Abs(delta_S.Imaginary)));
            }

            for (int i = 0; i < 3; i++) {

                linha.Barra_Para.I[i] = Complex.Conjugate((linha.Barra_Para.S[i]
+ linha.Barra_Para.Sg[i]) / linha.Barra_Para.V[i]);
                linha.J[i] = linha.Barra_Para.I[i];


            }
            for (int i = 0; i < linha.Barra_Para.Ramos_Jusante.Count(); i++)
            {
                if (linha.Barra_Para.Ramos_Jusante[i].Barra_Para.Cluster ==
linha.Barra_Para.Cluster)
                {
```

```
                    DFS(linha.Barra_Para.Ramos_Jusante[i]);
                    //Adicionar as operações de retorno

                    for (int j = 0; j < 3; j++)
                    {
                        linha.J[j] += linha.Barra_Para.Ramos_Jusante[i].J[j];

                    }
                }

            }

        } // Função recursiva de percurso na rede

        //Algoritmo PF
        int count = 0;

        double tolerancia = 1e-4;

        erro_max = 1;
        count = 0;

        for (int i = 0; i < barras.Length; i++)
        {
            barras[i].V = new Complex[3];
            barras[i].I = new Complex[3];
            barras[i].Sg = new Complex[3];
        }

        for (int i = 0; i < ramos.Length; i++)
        {
            ramos[i].J = new Complex[3];
        }

        for (int i = 0; i < 3; i++)
        {
            Barra_Raiz.V[i] = new Complex(Math.Cos(-2 * Math.PI * i / 3) * Vse /
Math.Sqrt(3), Math.Sin(-2 * Math.PI * i / 3) * Vse / Math.Sqrt(3));
        }

        //if (comGD == true)
        //{
        //    for (int i = 0; i < geradores.Length; i++)
        //    {
        //        for (int j = 0; j < barras.Length; j++)
        //        {
        //            if (geradores[i].id_Bus == barras[j].Numero)
        //            {
        //                barras[j].Sg = geradores[i].S;

        //                break;
        //            }
        //        }
        //    }
```

```
            //}

            //Cálculo do FP
            while (count < 100 && erro_max > tolerancia)
            {
                count++;
                erro_max = 0;

                for (int i = 0; i < Barra_Raiz.Ramos_Jusante.Count(); i++)
                {
                    DFS(Barra_Raiz.Ramos_Jusante[i]);
                }
                difS.Add(erro_max);
            }

        } //Função para Fluxo de Potência

        public void imprimirValor(string _path)
        {
            //Create a file to write to.
            using (StreamWriter sw = new StreamWriter(_path))
            {
                int count = 1; //Contador dos tipos de dados de saída
                double Vse = 13800 / Math.Sqrt(3);//Tensão de base

                //Diferença de potência mínima
                //if (outData.difSMin)
                //{
                //sw.WriteLine("<<<" + count.ToString() + ". DIFERENÇA DE POTÊNCIA
MÍNIMA>>>----------------------------------------------------------------------------");
                //sw.WriteLine("Smin = {0}", difSMin.ToString());
                //count++;
                //}

                //Quantidade de iterações
                //if (outData.qtdIte)
                //{
                //sw.WriteLine("<<<" + count.ToString() + ". QUANTIDADE DE
ITERAÇÕES>>>-------------------------------------------------------------------------------
----");

                //sw.WriteLine("Nint = {0}", qtdIte.ToString());


                //count++;
                //}

                //Custo computacional
                //if (outData.CusCom)
                //{
                //sw.WriteLine("<<<" + count.ToString() + ". TEMPO DE
PROCESSAMENTO>>>---------------------------------------------------------------------------
---------");
                //sw.WriteLine("Tmédio = {0} ms", (comT.Elapsed.TotalMilliseconds /
Npontos).ToString());
```

```
                //count++;
                //}

                //Diferença de potência por iteração
                if (Dado_Saida.difS)
                {
                    sw.WriteLine("<<<" + count.ToString() + ". DIFERENÇA DE POTÊNCIA
POR ITERAÇÃO>>>----------------------------------------------------------------------");
                    for (int i = 0; i < difS.Count; i++)
                        sw.WriteLine("{0}: {1}", i, difS[i].ToString());
                    count++;
                }

                //Valores de tensão
                if (Dado_Saida._3V)
                {
                    sw.WriteLine("<<<" + count.ToString() + ". TENSÃO(pu)>>>--------
--------------------------------------------------------------------------------");
                    for (int i = 0; i < barras.Length; i++)
                        sw.WriteLine("{0}: {1}[{2}°] {3}[{4}°] {5}[{6}°]",
                            barras[i].Numero,
                            (barras[i].V[0].Magnitude / Vse).ToString(),
                            (barras[i].V[0].Phase * 180 / Math.PI).ToString(),
                            (barras[i].V[1].Magnitude / Vse).ToString(),
                            (barras[i].V[1].Phase * 180 / Math.PI).ToString(),
                            (barras[i].V[2].Magnitude / Vse).ToString(),
                            (barras[i].V[2].Phase * 180 / Math.PI).ToString());
                    count++;
                }

                //Valores de Corrente
                if (Dado_Saida._3I)
                {
                    sw.WriteLine("<<<" + count.ToString() + ". CORRENTE(A)>>>-------
--------------------------------------------------------------------------------");
                    for (int i = 0; i < ramos.Length ; i++)
                        sw.WriteLine("{0}: {1}[{2}°] {3}[{4}°] {5}[{6}°]",
                            ramos[i].Barra_Para.Numero.ToString(),
                            ramos[i].J[0].Magnitude.ToString(),
                            (ramos[i].J[0].Phase * 180 / Math.PI).ToString(),
                            ramos[i].J[1].Magnitude.ToString(),
                            (ramos[i].J[1].Phase * 180 / Math.PI).ToString(),
                            ramos[i].J[2].Magnitude.ToString(),
                            (ramos[i].J[2].Phase * 180 / Math.PI).ToString());
                    count++;
                }
            }
        } //Gera arquivvo de texto com resultados do Fluxo de Potência

        public void calculateCPerdas()
        {
            for (int i = 0; i < ramos.Length - 1; i++)
            {
                ramos[i].Pt = 0;
```

```
                for (int l = 0; l < 3; l++)
                {
                    for (int c = 0; c < 3; c++)
                    {
                        ramos[i].Pt = ramos[i].Pt + ramos[i].zRamo[l][c].Real *
Math.Pow(ramos[i].J[c].Magnitude, 2);

                    }
                }

                PotPerdas = PotPerdas + (ramos[i].Pt * 1e-3); // Atributo  do Objeto
RD, PotPerdasRD armazenou o valor do somatório *1 (Preenchido em amarelo)
            }

            for (int j = 0; j < 20; j++)
            {
                Correcao = Correcao + (Math.Pow(1.1, -j) * PotPerdas); // Realiza o
somátorio *2 (Preenchido em Vermelho) e armazena o resultado no atributo RD.Correcao
            }

            CPerdas = 8760.0 * 0.25 * Correcao; //Calcula o custo das perdas para a
rede, armazenando-o como um atributo do objeto RD.

        }

        public void calculateMinMax()
        {
            Vmin = barras.Min(bus => bus.V.Min(v => v.Magnitude)) / (Vse /
Math.Sqrt(3));

            Vmax = barras.Max(bus => bus.V.Max(v => v.Magnitude)) / (Vse /
Math.Sqrt(3));

            Imax = ramos.Max(branch => branch.J.Max(J => J.Magnitude));

            Xmax = barras.Max(bus => bus.x);

            Xmin = barras.Min(bus => bus.x);

            Ymax = barras.Max(bus => bus.y);

            Ymin = barras.Min(bus => bus.y);
        }

        public void calculatePotSE()
        {
            for (int i = 0; i < Barra_Raiz.Ramos_Jusante.Count; i++)
            {
                for (int j = 0; j<3; j++)
                {
                    Ise = Barra_Raiz.Ramos_Jusante[i].J[j].Magnitude + Ise;

                }
```

```
            }
            PotSE = (Barra_Raiz.V[0].Magnitude) * Ise;

         }

        public void KMeansClustering(int nbrK)
        {
            // K-Menas Algorithm

            //Function to assign the cluster to a data point
            Func<double[][], int[], double[], int> getCluster = null;
            getCluster = (_means, sumN, posDP) =>
            {
                int cluster = 0;

                double[] squaredED = new double[_means.Length];
                double minSquaredED = 0.0;
                for (int i = 0; i < _means.Length; i++)
                {
                    for (int j = 0; j < posDP.Length; j++) squaredED[i] +=
Math.Pow(posDP[j] - _means[i][j], 2.0);
                    if (i == 0 || squaredED[i] < minSquaredED)
                    {
                        cluster = i;
                        minSquaredED = squaredED[i];
                    }
                }
                sumN[cluster]++;

                return cluster;
            };
            // Definir da posição inicial do centróide


            means = new double[nbrK][];
            for (int i = 0; i < nbrK; i++)
            {
                means[i] = new double[nbrK];
                for (int j = 0; j < means[i].Length; j++)
                {
                  if(j % 2 != 0)
                  {
                        double pos = Ymax + Ymin;
                        double div = 0.5;
                        means[i][j] = pos * div;
                  }
                  else
                  {
                     double pos = Xmin + Xmax;
                        if (i % 2 == 0)
                        {
                             double div = 0.25;
                            means[i][j] = pos * div;
                        }
                        else
```

```
                    {
                         double div = 0.75;
                        means[i][j] = pos * div;
                    }
                }

            }
        }


        int[] clusterN = new int[nbrK];
        foreach (Barra bus in barras)
        {
            //Assign the cluster using squared Euclidian distance vector
            bus.Cluster = getCluster(means, clusterN, new double[] { bus.x,
bus.y });
        }

        bool hasChangedCluster = true;
        int iterations = 1;
        while (hasChangedCluster && iterations <= 100)
        {
            hasChangedCluster = false;
            for (int i = 0; i < means.Length; i++) means[i] = new double[nbrK];

            foreach (Barra bus in barras) //Recalcula o centroide
            {
                //Get new centroid
                means[bus.Cluster][0] += bus.x / clusterN[bus.Cluster];
                means[bus.Cluster][1] += bus.y / clusterN[bus.Cluster];
                //means[(int)line.Attributes["Cluster"]][2] +=
(double)line.Attributes["eP"] / clusterN[(int)line.Attributes["Cluster"]]; Estamos
utilizando espaço Bidimensional
            }

            for (int i = 0; i < clusterN.Length; i++) clusterN[i] = 0;

            foreach (Barra bus in barras)
            {
                //Assign the cluster using squared Euclidian distance vector
                int newCluster = getCluster(means, clusterN, new double[] {
bus.x, bus.y });
                if (newCluster != bus.Cluster)
                {
                    hasChangedCluster = true;
                    bus.Cluster = newCluster;
                }
            }

            iterations++;
        }


    }
```

```
        public void ClusterSE()
        {
            //Associa a SE conecta a um centroide
            double[] cordSEcon = new double[]
            {
                subes.Where(se => se.conec == true).First().x, subes.Where(se =>
se.conec == true).First().y
            };

            double dist1 = Math.Sqrt(Math.Pow(cordSEcon[0] - means[0][0], 2) +
Math.Pow(cordSEcon[1] - means[0][1], 2));
            double dist2 = Math.Sqrt(Math.Pow(cordSEcon[0] - means[1][0], 2) +
Math.Pow(cordSEcon[1] - means[1][1], 2));
            int clusterDisc;

            if (dist1 > dist2)
            {
                subes.Where(se => se.conec == true).First().Cluster = 1;
                clusterDisc= 0;
            }
            else
            {
                subes.Where(se => se.conec == true).First().Cluster = 0;
                clusterDisc = 1;
            }

            for (int i = 0; i < subes.Length; i++)
            {
                if (subes[i].conec == false ) //pega as distâncias dos centróides
das SE não conectadas a rede
                {
                    subes[i].dist = Math.Sqrt(Math.Pow(subes[i].x -
means[clusterDisc][0], 2) + Math.Pow(subes[i].y - means[clusterDisc][1], 2));
                }

            }


            //Variavel local que vai armazenar a menor distancia de cada centroide
            //           double MenorDist = subes.Where(se => se.conec ==
false).Min(substation => substation.dist);
            var listmenordist = subes.Where(se => se.conec ==
false).OrderByDescending(substation => substation.dist).Reverse().ToArray();
            double MenorDist =  listmenordist[0].dist;


            for (int i = 0; i < subes.Length; i++) //Determinação das SE otimas
            {
                if (subes[i].dist == MenorDist) //Determina qual a SE proxima a C1
                {
                  SEotima1 = subes[i];
                }

            }
```

```
        }

        public void DescobreDesconec(int position)
        {

            Ramo ramoconec = ramoslig.Where(rl => rl.De ==
SEotima1.Numero).FirstOrDefault();
            Ramo ramoaux = ramos.Where(r => r.Para ==
ramoconec.Para).FirstOrDefault();
            double clusteratual = ramoaux.Barra_De.Cluster;
            while (ramoaux.Ramo_Montante != null && ramoaux.Barra_De.Cluster ==
clusteratual)
            {
                clusteratual = ramoaux.Barra_De.Cluster;
                ramoaux.visitado = true;
                ramoaux = ramoaux.Ramo_Montante;


            }


            if (position == 1)
            {
                // Para alterar o ramo de desconexão uma posição a frente
                for (int i = 0; i < 2; i++)
                {
                    ramoaux.Barra_Para.Cluster = ramoaux.Barra_De.Cluster;
                    ramodesconec = ramoaux.Barra_Para.Ramos_Jusante.Where(r =>
r.visitado).First();  //1 ramos a frente.}
                    ramoaux = ramodesconec;
                }
            }

            if (position == 2)
            {
                ramodesconec =
ramoaux.Barra_De.Ramo_Montante.Barra_De.Ramo_Montante; // 2 ramos atrás. Está
funcionando.
            }

            if (position == 0)
            {
                ramodesconec = ramoaux;

            }


        }

        public void ChangeCluster()
        {
            void DFS(Ramo linha)
            {

                if (linha.Barra_Para.Numero == ramodesconec.Barra_Para.Numero)
```

```
                {
                    return;
                }

                 linha.Barra_Para.Cluster = 2;

                for (int i = 0; i < linha.Barra_Para.Ramos_Jusante.Count(); i++)
                {
                    DFS(linha.Barra_Para.Ramos_Jusante[i]);
                    //Adicionar as operações de retorno

                }
            }
                Barra_Raiz.Cluster = 2;

            for (int i = 0; i < Barra_Raiz.Ramos_Jusante.Count(); i++)
            {

                DFS(Barra_Raiz.Ramos_Jusante[i]);
            }




        }

        public void ClusterCorrection()
        {
            for(int i = 0; i< barras.Length; i++)
            {
                if (barras[i].Cluster == 1)
                {
                    barras[i].Cluster = 0;
                }

            }

        }

        public void DirectionCorretion(Ramo liga, Ramo desconect)
        {
            Ramo RamoAux1 = liga;
            Ramo Ramojusante = liga.Barra_Para.Ramo_Montante;
            Ramo RamoAux2= desconect;

            while (RamoAux1 != desconect )
            {
                Ramo Swap = RamoAux1 != liga ? RamoAux1.Barra_De.Ramo_Montante :
RamoAux1;
                RamoAux1.Barra_De.Ramos_Jusante.Remove(RamoAux1); //Remove o ramo
atual da lista de ramos jusantes da barra DE
                RamoAux1.Barra_De.Ramos_Jusante.Add(Swap); //Define o ramo montante
da barra de como ramo jusante
```

```
                RamoAux1.Barra_De.Ramo_Montante = RamoAux1 != liga ?
RamoAux1:null;  //Ramo atual vira ramo montante da barra DE
                RamoAux2 = RamoAux1;
                RamoAux1 = RamoAux1 != liga ? Swap :
RamoAux1.Barra_Para.Ramo_Montante;
            }
                Ramo Swap_ = RamoAux1.Barra_De.Ramo_Montante;
                RamoAux1.Barra_De.Ramos_Jusante.Remove(RamoAux1); //Remove o ramo
atual da lista de ramos jusantes da barra DE
               // RamoAux1.Barra_De.Ramos_Jusante.Add(Swap_); //Define o ramo
montante da barra de como ramo jusante
                RamoAux1.Barra_De.Ramo_Montante = RamoAux1 != liga ? RamoAux1 :
null;  //Ramo atual vira ramo montante da barra DE
                RamoAux1 = desconect;

            while (RamoAux1.Barra_De != liga.Barra_Para)
            {
                String Deaux = RamoAux1.De;
                String Paraux1 = RamoAux1.Para;
                Barra BDaux = RamoAux1.Barra_De;
                Barra BPaux = RamoAux1.Barra_Para;
                RamoAux1.Para = Deaux;
                RamoAux1.De = Paraux1;
                RamoAux1.Barra_De = BPaux;
                RamoAux1.Barra_Para = BDaux;
                RamoAux1 = RamoAux1.Barra_De.Ramo_Montante;
            }
            liga.Barra_Para.Ramos_Jusante.Add(Ramojusante);

        }



    }

} //Modelo da rede de distribuição

namespace ConsoleApp
{
    class Program
    {
        static void Main(string[] args)
        {

            XmlDocument Xdoc = new XmlDocument();


            //Objeto do arquivo de dados

            Xdoc.Load(System.IO.Directory.GetParent(@"../../").FullName +
"\\tDN_135b_V4.xml");


            //Objeto do modelo da rede de distribuição
            modeloRD.Rede RD = new modeloRD.Rede();
```

```
            //Leitura dos dados da rede de distribuição
            RD.lerRede(Xdoc);

            //Calcular PF
            RD.calculatePF();


            //Executa o Kmeans
            RD.KMeansClustering(2);

            //Imprimir resultado do PF no pré falta
            RD.imprimirValor(System.IO.Directory.GetParent(@"../../").FullName +
"\\resultadoPF_pre.txt");

            //Calculo do Custo de perdas na Rede
            RD.calculateCPerdas();

            //Calculo do maior e menor valor de tensão e maior valor de corrente
            RD.calculateMinMax();

            // Calculo da Potência ativa fornecida pela SE
            RD.calculatePotSE();

            //Determina qual SE está mais proxima dos centroides
            RD.ClusterSE();

            //Verifica em qual barra há a mudança de cluster e qual o ramo que deve
ser utilizado para desconexão
            RD.DescobreDesconec(2);

            RD.ChangeCluster();

            RD.ClusterCorrection();

            Rede[] alimentadores = new Rede[2];
            //Rede nova -> novo alimentador
            alimentadores[0] = new Rede();
            List<Barra> lbarras = RD.barras.Where(b => b.Cluster == 0).ToList();
            lbarras.Add(RD.SEotima1);
            alimentadores[0].barras = lbarras.ToArray();
            alimentadores[0].Barra_Raiz = RD.SEotima1;
            alimentadores[0].Barra_Raiz.Ramos_Jusante.Add(RD.ramoslig.Where(rl =>
rl.De == RD.SEotima1.Numero).FirstOrDefault());
            List <Ramo> lramos = RD.ramos.Where(r => r.Barra_Para.Cluster
==0).ToList();
            lramos.Add(RD.ramoslig.Where(rl => rl.De ==
RD.SEotima1.Numero).FirstOrDefault());
            alimentadores[0].ramos = lramos.ToArray();
            alimentadores[0].DirectionCorretion(RD.SEotima1.Ramos_Jusante[0],
RD.ramodesconec);

            //Alimentador original
            alimentadores[1] = new Rede();
```

```
            alimentadores[1].barras = RD.barras.Where(b => b.Cluster ==
2).ToArray();
            alimentadores[1].Barra_Raiz = RD.Barra_Raiz;
            alimentadores[1].ramos = RD.ramos.Where(r => r.Barra_Para.Cluster ==
2).ToArray();

            for(int i=0; i<alimentadores.Length; i++)
            {
                alimentadores[i].calculatePF();
                alimentadores[i].calculateCPerdas();
                alimentadores[i].calculateMinMax();
                alimentadores[i].calculatePotSE();
                alimentadores[i].imprimirValor(System.IO.Directory.GetParent(@"../..
/").FullName + "\\resultadoall" + i.ToString() + ".txt");
            }



        }
    }
}
```