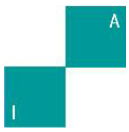
Inteligencia Artificial, 24(68), 123-137
doi: 10.4114/intartif.vol24iss68pp123-137



INTELIGENCIA ARTIFICIAL

http://journal.iberamia.org/

Algoritmo Híbrido Genético Guloso para o Problema de Alocação de Agregadores em Redes Elétricas Inteligentes

Greedy Genetic Hybrid Algorithm for the Data Aggregator Allocation Problem in Smart Grids

Sami Nasser Lauar[1,A], Mário Mestria[1,2,B,C]

[1]Coordenadoria De Engenharia Elétrica - Instituto Federal Do Espírito Santo (Ifes), Av. Vitória, N° 1729 - Bairro Jucutuquara, 29040-780, Vitória, Es Brasil;
[2]Programa De Pós-Graduação Em Tecnologias Sustentáveis Campus Vitória, Ifes
[A]samilauar10@gmail.com, [B]mmestria@ifes.edu.br, [C]mmestria@uol.com.br

Abstract In this work, we present a metaheuristic based on the genetic and greedy algorithms to solve an application of the set covering problem (SCP), the data aggregator positioning in smart grids. The GGH (Greedy Genetic Hybrid) is structured as a genetic algorithm, but it has many modifications compared to the classic version. At the mutation step, only columns included in the solution can suffer mutation and be removed. At the recombination step, only columns from the parent's solutions are available to generate the offspring. Moreover, the greedy algorithm generates the initial population, reconstructs solutions after mutation, and generates new solutions from the recombination step. Computational results using OR-Library problems showed that the GGH reached optimal solutions for 40 instances in a total of 75 and, in the other instances, obtained good and promising values, presenting a medium gap of 1,761%.

Resumo Neste trabalho é proposta uma meta-heurística baseada nos algoritmos genéticos e gulosos para resolver uma aplicação do problema de cobertura de conjuntos (PCC), a alocação de agregadores em redes elétricas inteligentes. O HGG (Híbrido Genético Guloso) é estruturado como um algoritmo genético, mas apresenta diversas modificações em relação a sua versão clássica. Na etapa de mutação apenas colunas inclusas na solução podem sofrer mutação e serem removidas. Na etapa de recombinação apenas colunas das soluções pais podem gerar filhos. Além disso, o algoritmo guloso é usado para gerar a população inicial, reconstruir soluções após a mutação e construir as soluções geradas pela recombinação. Os resultados computacionais usando problemas da OR-Library mostraram que o HGG alcançou soluções ótimas em 40 instâncias num total de 75 e, nas outras instâncias, obteve valores bons e promissores, apresentando um gap médio de 1,761%.

Keywords: Optimization, Metaheuristics, Smart Grids, Set Covering Problem, Genetic Algorithms, Greedy Algorithms.

Palavras Chave: Otimização, Meta-heurística, Redes Inteligentes, Problema de Cobertura de Conjuntos, Algoritmos Genéticos, Algoritmos Gulosos.

ISSN: 1137-3601 (print), 1988-3064 (on-line)

1 Introdução

O conceito de Rede Elétrica Inteligente (REI), ou *smart grid* (em inglês), abrange uma ampla gama de questões de pesquisa, como: controle distribuído, detecção de falhas, previsão, estabilidade da rede, comunicação de dados e resposta à demanda. Assim, *smart grid* é uma área multidisciplinar que apresenta muitos desafios [17].

Sendo assim, é importante entender o funcionamento de uma REI que, apesar de ter vários conceitos, nesse trabalho é adotada e visualizada como composta de medidores inteligentes, agregadores de dados e uma central gestora [22] [13] [1].

O faturamento não é mais a única função dos medidores inteligentes, que podem coletar mais de 20 parâmetros de dados elétricos [16] a uma taxa que varia de uma coleta a cada cinco minutos, a uma coleta a cada hora. Por outro lado, a grande popularidade dos medidores inteligentes faz com que uma grande quantidade de dados de consumo de eletricidade sejam coletados. Isso significa que as concessionárias de distribuição de energia elétrica precisam lidar com uma quantidade considerável de dados [2]. É fornecida a coleta de dados com maior eficácia do serviço, integralidade e acesso universal a partir de medidores inteligentes, produzindo informações valiosas sobre o consumo de eletricidade, comportamentos e estilos de vida do consumidor [24].

Além disso, numa infraestrutura de medição avançada é possível ter sistemas de gerenciamento de dados de medição, sistemas de monitoramento e sistemas de informação e controle [15]. Ainda, nessa infraestrutura de medição podem ser coletados os dados que representam uma fonte de informação em tempo real, não apenas sobre o consumo de eletricidade, mas também como um indicador de outras dinâmicas sociais, demográficas e econômicas dentro de uma cidade [11].

Portanto, em uma REI, o medidor inteligente é responsável por medir dados de energia para cada consumidor e muitas vezes é idealizado como capaz de se comunicar com os eletrodomésticos da residência, esta última atribuição faz parte do fenômeno "internet das coisas". Além disso ele também pode realizar a interrupção e religamento da energia, isso tudo de forma remota por meio de uma comunicação bidirecional com os agregadores de dados. A comunicação entre medidores, agregadores e a central pode ser realizada por tecnologias sem fio (rede de celulares, GPRS, WiMax, WiFi, ZigBee, Bluetooth) ou cabos (fibra óptica, cabo coaxial e cabos metálicos) [13].

Os agregadores de dados são dispositivos posicionados de forma difusa pela rede e têm o papel de realizar medições, detectar falhas e agir como ponte entre a central e os medidores inteligentes, podendo até mesmo armazenar dados dos medidores por um determinado período de tempo. A comunicação entre agregadores e medidores é bidirecional, ou seja, os agregadores tanto recebem dados dos medidores quanto enviam comandos para os medidores, o mesmo ocorre entre agregadores e a central. Por último, a central gestora é onde a concessionária armazena os dados dos consumidores e faz o monitoramento e gerenciamento da rede.

Com tudo isso em vista, muitas são as vantagens da implementação de uma REI, dentre elas está reduzir custos de operação e gestão da rede ao permitir o controle remoto, além do aumento da eficiência energética e confiabilidade do sistema elétrico ao possibilitar a detecção e correção automática de problemas. Além disso, as REIs permitiriam ao distribuidor e consumidor monitorar o consumo de energia em tempo real, viabilizando a implementação de tarifas flutuantes, em que o preço do kWh varia ao longo do dia. Isso possibilitaria economia aos consumidores que evitarem de usar energia em horários de pico, que a tarifa é mais cara, e, assim, evita-se sobrecarregar a rede e diminui-se a demanda energética e necessidade de expansão.

Sob outro ponto de vista, as *smart grids* são um avanço natural quando se aborda o tema cidades inteligentes, dado que fazem parte do fenômeno IoT (Internet das Coisas). Como pode ser observado na Figura 1, em uma REI ocorre a integração do sistema de distribuição de energia à rede mundial de computadores, residências, indústrias e sistemas de geração e armazenamento de energia elétrica (bidirecionalidade de energia elétrica). Salienta-se ainda que as REIs apresentam vantagens do ponto de vista ambiental, pois elas permitem integrar fontes de energia renováveis, como eólicas e fotovoltaicas, à rede elétrica.

Escolher eficientemente as melhores posições para os agregadores é uma tarefa difícil, principalmente em grandes cidades que podem conter milhares de medidores em um único bairro [22]. As tecnologias atuais para comunicar agregadores e medidores limitam o posicionamento de agregadores de dados ao

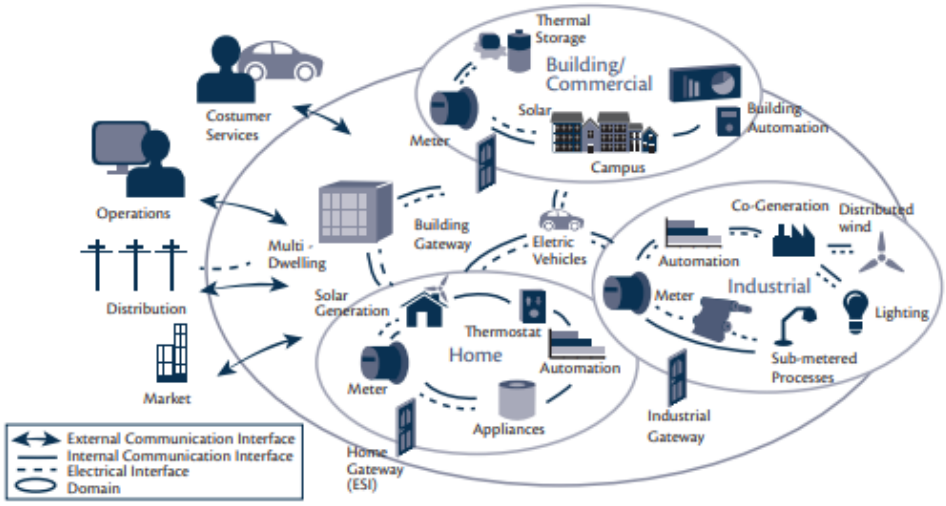


Figura 1: Representação de uma Rede Elétrica Inteligente. Fonte: [13].

longo da rede de distribuição de energia elétrica devido à diminuição da propagação do sinal [18]. Sendo assim, é necessário um modelo matemático que simplifique este problema para assim resolvê-lo, um modelo que se encaixa perfeitamente é o Problema de Cobertura de Conjuntos (PCC) ou *Set Covering Problem* (SCP). Este trabalho propõe uma meta-heurística para resolver o SCP e, consequentemente, o problema de alocação de agregadores em uma *smart grid*.

Trabalhos relacionados já adaptaram esse problema como um SCP [22] ou *vertex-covering problem* [8] e até mesmo modelos de otimização adaptados [1], todos visando a redução dos custos de implementação e expansão, assim como este trabalho. Por outro lado, há trabalhos na literatura que visam a melhoria do sinal entre os componentes da REI [25] ou melhor estabilidade da rede [23].

Este artigo está organizado da seguinte forma: na Seção 2 é descrito o modelo matemático utilizado. Na Seção 3 apresenta-se os algoritmos heurísticos que compõem a meta-heurística desenvolvida (HGG) para resolver o SCP enquanto na Seção 4 apresenta-se o HGG detalhadamente. Na Seção 5 são apresentados os resultados dos testes computacionais, na Seção 6 os resultados do HGG são comparados com outros algoritmos modernos e na Seção 7 são feitas as considerações finais.

2 Modelo Matemático

O SCP é um modelo aplicado em uma vasta gama de problemas, dentre eles, podemos citar como relevantes a montagem da escala de trabalho de uma equipe (*crew schedulling*) [5] e posicionamento de ambulâncias [20].

No SCP, busca-se escolher o menor número de conjuntos para cobrir todos os nós. A Figura 2 mostra como o SCP é aplicado no problema de posicionamento de agregadores, nela os medidores são os nós, os postes são as possíveis localização dos agregadores e, consequentemente, são o ponto central dos conjuntos. Feita a representação geográfica, é montada a matriz de cobertura que analisará quais nós cada conjunto abrange, sendo que ainda é possível associar um custo individual para cada conjunto, representado pelo vetor de custos.

Construindo o modelo do SCP para um problema de n postes e m medidores, ele pode ser descrito pela função objetivo, que é a equação (1), e as restrições, que são as equações (2) e (3). Logo, a formulação matemática é feita como:

Minimizar:

$$C = \sum_{j=1}^{n} c_j \cdot x_j \tag{1}$$

Sujeito à:

$$\sum_{j=1}^{n} a_{ij} \cdot x_j \geq 1 \qquad i = 1, 2, 3...m \tag{2}$$

$$x_j \in \{0, 1\} \qquad j = 1, 2, 3...n \tag{3}$$

Sendo que X é o vetor de solução e nele, $x_j = 1$ se o poste j pertence a solução e $x_j = 0$ caso contrário. A matriz $A_{m \times n}$ é chamada matriz de cobertura e é preenchida com $a_{ij} = 1$ se o medidor i está na área de cobertura do agregador posicionado no poste j, caso contrário, $a_{ij} = 0$. Por fim, c é chamado de vetor de custos e c_j representa o custo de ter um agregador no poste j.

3 Métodos Heurísticos

Há muitas formas de resolver um SCP, entre elas estão métodos exatos como os métodos *simplex*, *branch and bound* e *branch and cut*. Entretanto, o problema de otimização SCP abordado é do tipo NP-difícil [10] e nem sempre métodos exatos encontram soluções ótimas devido a recursos computacionais de memória RAM (*Random Access Memory*) exigidos para instâncias de grande porte. Nesse sentido, é necessário recorrer a métodos heurísticos, que, por sua vez, não garantem encontrar a solução ótima, mas conseguem soluções de alta qualidade em tempo computacional baixo.

Trabalhos encontrados na literatura apresentam várias heurísticas para resolver o SCP, como algoritmos *greedy* [14], genéticos [4], de busca em vizinhança [6], de rede neural [12] entre outros métodos heurísticos e variações dos clássicos *greedy* [21] e genético [7]. Neste trabalho foi desenvolvido um algoritmo genético, que é auxiliado pelo método *greedy*, para solucionar o SCP, essa decisão foi tomada considerando o potencial do método genético em resolver problemas de larga escala, que é o esperado para o planejamento de *smart grids*, enquanto o método *greedy* fornece os indivíduos para gerar a população inicial e as novas gerações.

3.1 Algoritmos Genéticos

Os Algoritmos Genéticos (AG) são métodos de busca probabilísticos baseados na seleção natural e genética, nele a população em análise é o conjunto de indivíduos, interpretados como possíveis soluções para o problema. Já o indivíduo é formado por um conjunto de cromossomos e a informação de cada cromossomo são os genes. Os dois são interpretados de acordo com o problema em questão, no caso do caixeiro viajante, por exemplo, cromossomos seriam rotas e os genes as cidades, mas no caso do SCP cromossomos são as possíveis localizações de agregadores e o gene a escolha de ter ou não um agregador nessa posição.

O AG clássico é dividido em etapas, sendo elas: A inicialização, em que é criada a população inicial. Logo em seguida, a seleção consiste na identificação e escolha dos indivíduos mais adaptados, limitando o tamanho da população. A recombinação é uma etapa de troca de genes entre os indivíduos. Por último

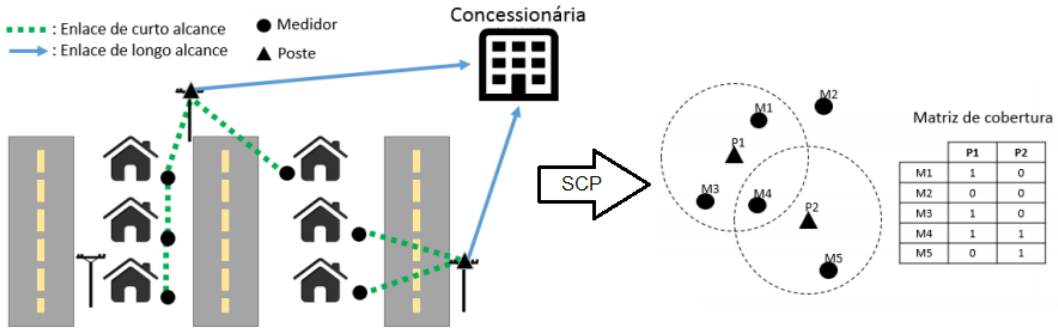


Figura 2: Planejamento de uma Smart Grid modelado como um SCP. Fonte: [22]

ocorre a mutação, que é a mudança aleatória nos genes de um ou mais indivíduos. As etapas se repetem da seleção até à mutação até serem encontrados indivíduos adaptados o suficiente ou o tempo de testes se encerrar.

3.2 Algoritmos Gulosos

O algoritmo guloso é simples, objetivo e efetivo. Em contraste com o algoritmo genético, que é probabilístico, o algoritmo guloso é essencialmente determinístico (apesar de que variações desses aplicados, por exemplo ao GRASP [21] podem torná-lo probabilístico), o que algumas vezes se torna um obstáculo na busca da solução ótima, pois como afirma Reyes [21], algoritmos gulosos raramente encontram soluções próximas da ótima quando tratam de problemas grandes, ainda que encontrem soluções boas. No algoritmo guloso são usadas funções de aptidão para avaliar as colunas e guiar a tomada de decisões, essas funções dependem de fatores como o custo da coluna e quantas linhas ela abrange.

4 O Algoritmo Híbrido Genético Guloso (HGG)

O HGG é um algoritmo genético com diversas modificações, ele emprega o método guloso na inicialização, mutação e recombinação. O pseudocódigo dele é mostrado no Algoritmo 1.

Algoritmo 1 Híbrido Genético Guloso

```
1: função HGG(A, c, timeLimit = −1, niterLimit = −1, objLimit = −1)
2:     niter ← 0
3:     {pop, tampop, t1} ← PopulaçãoInicial(A, c)
4:     se (timeLimit == −1) & (niter == −1) então
5:         timeLimit ← tempoLimite(t1)
6:     fim se
7:     {group, tamgroup, bestSol, bestObj} ← Avaliação(pop, t1)
8:     enquanto (tempo() ≤ timelimit) & (niter ≠ niterLimit) & (bestObj ≥ objLimit) faça
9:         niter ← niter + 1
10:        pop ← Recombinação(group)
11:        Pm ← ProbabilidadeMutação(tempo(),timelimit)
12:        pop ← Mutação(pop, Pm)
13:        {pop, bestSol, bestObj} ← Seleção(tampop, pop, bestSol, bestObj)
14:    fim enquanto
15:    retorna bestSol, bestObj
16: fim função
```

Para retornar a solução do SCP o HGG recebe a matriz de cobertura $A_{m\times n}$, o vetor de custos c e pode receber parâmetros que são condições de paradas, sendo eles o tempo limite de execução $timeLimit$, o número limite de iterações $niterLimit$ ou um valor desejado para o custo da solução $objLimit$. Em seguida é gerada a população inicial pela função PopulaçãoInicial(A, c) (seção 4.1) e se não for fornecido um tempo limite o HGG calcula um a partir da função tempoLimite$(t1)$ (seção 4.2) para então passar pela etapa de avaliação da população inicial (seção 4.3) antes de entrar no loop genético de Recombinação (seção 4.4), Mutação (seção 4.5) e Seleção (seção 4.6), respectivamente, para gerar novas e melhores soluções até ser satisfeito um dos critérios de parada.

4.1 População Inicial

Na construção da população inicial emprega-se o método *greedy*. Na construção de cada solução é feita a avaliação das colunas por meio de uma dentre 8 funções de *fitness* , sendo que algumas foram baseadas por funções propostas por Lan [14], as funções de fit_j são: $fit_j = 1/c_j \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/g_i$, $fit_j = 1/\sqrt{c_j} \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/g_i$, $fit_j = 1/c_j \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/\sqrt{g_i}$, $fit_j = 1/c_j^2 \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/g_i$, $fit_j = 1/c_j \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/g_i^2$, $fit_j = 1/c_j \cdot \sum_{i=p_j[0]}^{p_j[-1]} (1+g_i)/g_i$, $fit_j = 1/c_j \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/log(g_i+1)$ e $fit_j = 1/c_j \cdot \sum_{i=p_j[0]}^{p_j[-1]} 1/(g_i \cdot log(g_i+1))$.

Sendo que p_j é o vetor que contém todas as linhas que a coluna j abrange, c_j é o custo da coluna j e g_i é a quantidade de colunas que cobrem a i-ésima linha. As funções fit_j divergem das funções de Lan pois elas usam um somatório que inclui no cálculo da *fitness* a quantidade de colunas que também abrangem cada linha que a coluna avaliada abrange, tornando a avaliação mais precisa.

Salienta-se que usar diversas funções de avaliação é importante para o algoritmo genético pois implica em maior diversidade genética e ao mesmo tempo diminui a chance de serem criados clones na população inicial, com isso aumenta-se a chance do algoritmo encontrar soluções de boa qualidade e próximas da solução ótima.

A calibração do parâmetro *tampop* (tamanho da população) resultou em $tampop = 10$ após testes computacionais envolvendo diversas instâncias da base OR-Libray. A influência desse parâmetro se traduz no fato de que uma população pequena resulta em uma análise rápida, mas que pode levar a mínimos locais (se existirem), o que pode ser entendido como uma baixa variabilidade genética da população conforme forem se passando as gerações. Por outro lado, uma população muito grande resultaria em uma análise demorada, mas com mais chances de encontrar uma melhor solução. Mais um fator a ser apontado é o caráter determinístico do método *greedy* empregado para gerar a população inicial, que para uma população muito grande acarretaria em muitos indivíduos clones que não contribuiriam de forma eficiente para o algoritmo.

Dessa forma, foram testados tamanhos de população fixos de 6 até 100 e até mesmo desenvolvida uma função para determinar o tamanho da população, fazendo com que o tamanho da população variasse de 100 para problemas pequenos até 10 para problemas grandes, de forma a equilibrar o tempo de cada iteração. Isso se tornou vantajoso para alguns problemas pequenos como o CYC.06 mas apenas prolongou significativamente o tempo para achar a solução na maioria dos problemas. Tendo isso em vista e a partir de testes feitos foi adotada uma população fixa de 10 indivíduos para o HGG.

Ainda no quesito do tamanho da população, foi testado um mecanismo baseado na imigração, em que são adicionadas novas soluções na população caso o algoritmo fique por um determinado tempo sem achar uma solução melhor, a fim de sair de um possível mínimo local, entretanto os testes não apresentaram bons resultados e essa adição foi descartada.

4.2 Tempo Limite

A função tempoLimite($t1$) calcula o tempo limite de execução do HGG com base no tamanho do problema, este que é identificado pelo parâmetro $t1$ (tempo para gerar a primeira solução da população), o cálculo do tempo limite é feito usando uma função exponencial se $t1$ é menor que um valor t_x ou uma função logarítmica se $t1$ é maior que esse valor, sendo que t_x é um valor determinado a partir de testes para ser aproximadamente o tempo de gerar uma única solução para um problema de médio porte. Dessa forma, o tempo limite para problemas pequenos é cerca de 1 minuto, para problemas médios é cerca de 5 minutos e para problemas grandes pode chegar a cerca de 30 minutos. Constata-se que mesmo que o usuário entre com um objetivo limite, o tempo limite ainda será determinado, isso ocorre pois é possível que o objetivo determinado pelo usuário não seja possível de se alcançar e o algoritmo execute indefinidamente.

4.3 Avaliação

Na etapa de avaliação é identificada a melhor solução (com menor custo), o resultado a função objetivo para esta solução e por fim é definido um grupo de tamanho *tamgroup* com os melhores indivíduos da população, apenas soluções desse grupo realizam a recombinação. A avaliação de cada individuo é feita usando a função objetivo do SCP presente da equação (1).

4.4 Recombinação

A etapa de recombinação no AG clássico consiste no uso de *crossover operators*, que são pontos do material genético que delimitam as partes do material genético será trocada entre dois indivíduos. A recombinação com o *crossover operator* gera 2 soluções que podem ou não ser factíveis, e por ser uma troca aleatória de colunas entre duas soluções, dificilmente contribui para se obter uma solução melhor que os pais.

Dessa forma, a recombinação desenvolvida consiste em selecionar aos pares soluções "pais" do *group* (grupo com *tamgroup* melhores soluções da população) para criar uma solução "filha" usando o método guloso. Para isso, somente as colunas das soluções pais são disponíveis para serem escolhidas pelo método guloso, garantindo que a solução gerada pela recombinação seja factível, boa e possivelmente melhor que as soluções pais. Uma desvantagem desse procedimento é que a solução criada pode ser igual a uma das soluções pais, principalmente se eles tiverem muitas das colunas escolhidas em comum.

Os testes para calibrar o parâmetro *tamgroup* (tamanho do grupo) foram feitos comparando o quanto a mutação e a recombinação diminuíram o custo da melhor solução de uma iteração para outra, atribuindo um número de pontos igual a diferença do custo total entre a nova melhor solução e a anterior. Logo, para $tamgroup = 6$ foi observado que a mutação obteve uma pontuação de até 19,4 vezes melhor (resultado para problemas da classe A da OR-Library) em testes de problemas pequenos e cerca de 2 vezes melhor em problemas médios. Por outro lado, a recombinação obteve um desempenho melhor em problemas de larga escala, chegando a obter uma pontuação 5,8 vezes maior que a mutação no maior problema testado, que é o CYC.11. Sendo assim, foi feito que o parâmetro *tamgroup* possa variar de 4 a 10 dependendo do tamanho do problema, que é medido pelo tempo do algoritmo guloso gerar a primeira solução ($t1$). O cálculo de *tamgroup* é feito na etapa de Avaliação (seção 4.3).

Também foram testadas outras formas de realizar a recombinação, são elas: gerar uma solução a partir dos genes de 3 indivíduos pais; usar todos as colunas selecionadas pelos pais para então retirar as redundantes, como sugere Constantino [7], e adicionar de forma aleatória colunas que não são dos pais para poderem fazer parte da solução filha. Todavia, todas as 3 modificações foram descartadas nos testes por não produzirem resultados melhores que o método já empregado ou por prolongarem o tempo de execução.

4.5 Mutação

No método clássico adicionar mais colunas do que remover não contribui para o objetivo de minimizar a função de custos, só aumenta o custo, ainda por cima essas adições e remoções sendo feitas de forma aleatória não é algo muito eficiente, devido ao aumento da redundância (uma linha coberta por mais de uma coluna) nas soluções quando colunas aleatórias entram na solução. Logo, a etapa de mutação do HGG foi repensada do algoritmo clássico, nesta, somente onde $x_j = 1$ que pode ocorrer uma mutação e, caso ocorra, x_j se torna 0 e a coluna j não faz mais parte dessa solução. Ao final da mutação, soluções factíveis podem tornar-se inválidas, logo, cada solução passa pelo mesmo método guloso que gera a população inicial para reconstruí-la escolhendo-se as melhores colunas possíveis para isso. Deve-se ressaltar que todos os indivíduos da população participam da mutação.

Outra adição feita a etapa de mutação, mas já proposta por Beasley [4], é a de variar a probabilidade de ocorrer uma mutação (Pm), fazendo-a aumentar conforme o algoritmo converge em uma solução. Contudo, para determinar a variável, o tipo de função que determinará a probabilidade de mutação e os valores mínimos e máximos dela foram feitos vários testes. Foram testados os crescimentos logarítmico, exponencial e de reta e os intervalos que fariam Pm começar entre 4 a 8% e terminasse de 12 a 50%. Ao final, foi adotada uma função logarítmica que varia de 6 até 35% tendo como a variável independente a razão do tempo de execução e o tempo limite.

4.6 Seleção

A Seleção é feita no final de cada iteração, nela, o tamanho da população é limitado por *tampop* e são selecionadas apenas as melhores soluções para passar para a próxima geração (ou próxima iteração), ao mesmo tempo, é identificada a melhor solução e atualizada a variável *bestObj* com o resultado da função objetivo para esta solução.

5 Resultados Computacionais

Os testes foram conduzidos em um notebook com uma CPU Intel Core i5-8250U 1,6 GHz e 8GB de memória RAM. O algoritmo foi implementado na linguagem de programação Python 3.

Os testes foram feitos em 75 problemas da OR-Library [3], que reúne problemas de otimização de vários tipos, dentre eles está o SCP. Os problemas do tipo SCP da OR-Library são divididos em classes de acordo com algumas de suas características. Para os testes foram usadas as classes 4, 5, 6, A, B, C, D, E, NRE, NRF, NRG, NRH, CYC e CLR. Entre eles há problemas de pequena, média e larga escala, em que se variam a quantidade de linhas, colunas, a densidade ($d_{\%} = 100.q/(m.n)$, sendo q o número de 1s na matriz de cobertura $A_{m \times n}$) e o custo de cada coluna, todos esses parâmetros são expostos na Tabela 1. Desse modo, quanto menor o tamanho do problema e quanto mais linhas cada coluna abrange menos tempo e processamento é requerido para se atingir a solução ótima, o que pode ser explicado também pelo fato de que se os conjuntos abrangem mais nós, haverá menos colunas na solução e consequentemente menos iterações serão necessárias para gerá-la.

Tabela 1: Dados dos problemas do tipo SCP da OR-Library.

Instância	Linhas (m)	Colunas (n)	m x n	Densidade (%)	Intervalo de Custos
4	200	1.000	200.000	2	1-100
5	200	2.000	400.000	2	1-100
NRE	500	5.000	2.500.000	10	1-100
NRF	500	5.000	2.500.000	20	1-100
NRG	1.000	10.000	10.000.000	2	1-100
NRH	1.000	10.000	10.000.000	5	1-100
A	300	3.000	900.000	2	1-100
B	300	3.000	900.000	5	1-100
C	400	4.000	1.600.000	2	1-100
D	400	4.000	1.600.000	5	1-100
E	50	500	25.000	10	1-100
CLR.10	511	210	107.310	12,33	1
CLR.11	10223	330	337.590	12,41	1
CLR.12	2047	495	1.013.265	12,46	1
CLR.13	4095	715	2.927.925	12,48	1
CYC.06	240	192	46.080	2,08	1
CYC.07	672	448	301.056	0,89	1
CYC.08	1.792	1.024	1.835.008	0,39	1
CYC.09	4.608	2.304	10.616.832	0,17	1
CYC.10	11.520	5.120	58.982.400	0,078	1
CYC.11	28.160	11.264	317.194.240	0,0355	1

Para fazer uma análise da qualidade da solução atribuída pelo programa foram necessárias no mínimo 10 execuções para cada problema e foi calculado a média aritmética, o desvio padrão ($\sigma = \sqrt{\sum(x_i - \bar{x})}$) e o GAP ($GAP_{\%} = 100.(\bar{Obj} - BKS)/BKS$), que relaciona o objetivo encontrado pela heurística com o melhor objetivo na literatura (BKS ou *Best Known Solution*) [14], em porcentagem. Foi calculado tanto o GAP do melhor objetivo encontrado quanto do objetivo médio dos testes de cada problema. Na Tabela 2 encontram-se os resultados dos testes, sendo que o tempo, presente na última coluna, é a média do tempo que o HGG levou para encontrar a solução final retornada pelo algoritmo.

A fim de medir a eficácia do algoritmo genético em melhorar as soluções encontradas pelo método *greedy* ao gerar a população inicial também foi medido o GAP médio das soluções da população inicial. Dessa forma, o gráfico da Figura 3 reúne o GAP da população inicial (*Greedy Gap*) e da solução final do HGG para as 75 instâncias, comprovando o bom desempenho da parte genética do HGG já que ele melhorou as soluções em todos os problemas com exceção do scp5.3 e dos problemas da classe E, em que a solução ótima já havia sido encontrada na população inicial, logo, não havia possibilidade de melhora.

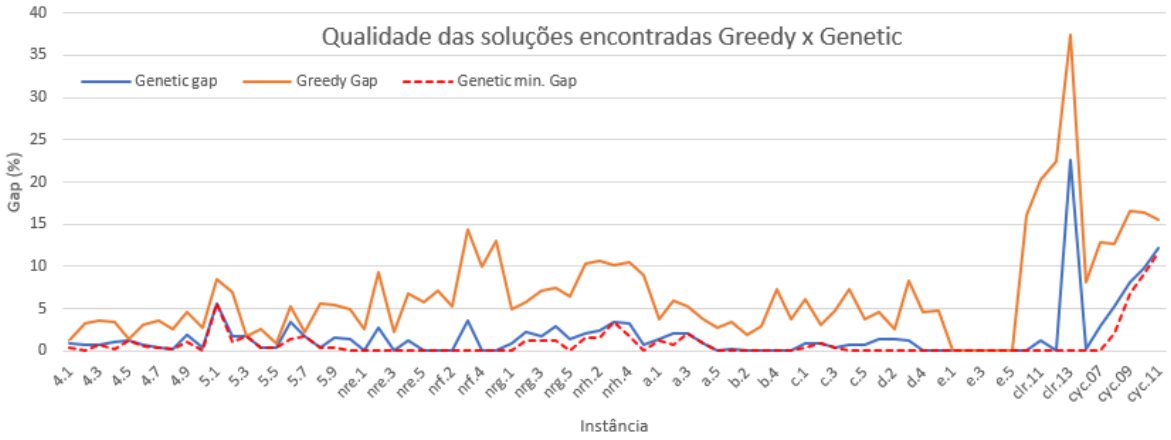


Figura 3: Gráfico de comparação das soluções geradas pelo método guloso e genético.

Tabela 2: Resultados dos testes.

Instância	BKS	Obj Min	Gap Min (%)	Obj Med	Gap Med (%)	σ	Tempo (s)
4.1	429	431	0,466	432,8	0,886	0,63	3,59
4.2	512	**512**	**0,000**	515,9	0,762	4,90	27,20
4.3	516	520	0,775	520,0	0,775	0,00	34,40
4.4	494	495	0,202	499,5	1,113	1,58	19,00
4.5	512	518	1,172	518,0	1,172	0,00	2,09
4.6	560	563	0,536	563,9	0,696	0,57	70,50
4.7	430	432	0,465	432,0	0,465	0,00	18,30
4.8	492	493	0,203	493,0	0,203	0,00	7,13
4.9	641	648	1,092	653,0	1,872	2,36	90,30
4.10	514	**514**	**0,000**	516,0	0,389	1,15	31,27
5.1	253	267	5,534	267,3	5,652	0,48	78,80
5.2	302	305	0,993	307,5	1,821	2,51	79,00
5.3	226	230	1,770	230,0	1,770	0,00	0,99
5.4	242	243	0,413	243,0	0,413	0,00	41,07
5.5	211	212	0,474	212,0	0,474	0,00	3,50
5.6	213	216	1,408	220,4	3,474	3,06	81,50
5.7	293	298	1,706	298,0	1,706	0,00	3,04
5.8	288	289	0,347	289,0	0,347	0,00	105,00
5.9	279	280	0,358	283,2	1,505	2,30	86,50
5.10	265	**265**	**0,000**	268,6	1,358	1,27	43,40
nre.1	29	**29**	**0,000**	**29,0**	**0,000**	0,00	26,10
nre.2	30	**30**	**0,000**	30,8	2,667	0,41	138,00
nre.3	27	**27**	**0,000**	**27,0**	**0,000**	0,00	36,00
nre.4	28	**28**	**0,000**	28,4	1,250	0,49	234,00
nre.5	28	**28**	**0,000**	**28,0**	**0,000**	0,00	32,30
nrf.1	14	**14**	**0,000**	**14,0**	**0,000**	0,00	103,00
nrf.2	15	**15**	**0,000**	**15,0**	**0,000**	0,00	18,20
nrf.3	14	**14**	**0,000**	14,5	3,571	0,53	155,00

Instância	BKS	Obj Min	Gap Min (%)	Obj Med	Gap Med (%)	σ	Tempo (s)
nrf.4	14	**14**	**0,000**	**14,0**	**0,000**	0,00	65,70
nrf.5	13	**13**	**0,000**	**13,0**	**0,000**	0,00	160,00
nrg.1	176	**176**	**0,000**	177,6	0,909	0,70	364,00
nrg.2	154	156	1,299	157,5	2,273	0,85	600,00
nrg.3	166	168	1,205	168,8	1,687	0,63	569,00
nrg.4	168	170	1,190	172,9	2,917	1,79	494,00
nrg.5	168	**168**	**0,000**	170,4	1,429	1,35	588,00
nrh.1	63	64	1,587	64,3	2,063	0,53	481,00
nrh.2	63	64	1,587	64,5	2,381	0,00	472,00
nrh.3	59	61	3,390	61,0	3,390	0,00	413,00
nrh.4	58	59	1,724	59,9	3,276	0,74	552,00
nrh.5	55	**55**	**0,000**	55,4	0,727	0,52	637,00
a.1	253	256	1,186	256,5	1,383	0,53	52,04
a.2	252	254	0,794	257,2	2,063	0,00	118,00
a.3	232	237	2,155	237,0	2,155	0,00	24,60
a.4	234	236	0,855	236,0	0,855	0,00	33,70
a.5	236	**236**	**0,000**	236,1	0,042	0,30	93,10
b.1	69	**69**	**0,000**	**69,2**	**0,290**	0,40	59,60
b.2	76	**76**	**0,000**	**76,0**	**0,000**	0,00	13,00
b.3	80	**80**	**0,000**	**80,0**	**0,000**	0,00	124,00
b.4	79	**79**	**0,000**	**79,0**	**0,000**	0,00	92,90
b.5	72	**72**	**0,000**	**72,0**	**0,000**	0,00	17,80
c.1	227	228	0,441	229,1	0,925	1,00	374,00
c.2	219	221	0,913	221,0	0,913	0,00	26,70
c.3	243	244	0,412	244,0	0,412	0,00	87,50
c.4	219	**219**	**0,000**	220,6	0,731	1,50	305,00
c.5	215	**215**	**0,000**	216,5	0,698	0,85	123,00
d.1	60	**60**	**0,000**	**60,8**	**1,333**	0,42	55,20
d.2	66	**66**	**0,000**	**66,9**	**1,364**	0,32	43,20
d.3	72	**72**	**0,000**	**72,9**	**1,250**	0,32	142,00
d.4	62	**62**	**0,000**	**62,0**	**0,000**	0,00	25,40
d.5	61	**61**	**0,000**	**61,0**	**0,000**	0,00	36,30
e.1	5	**5**	**0,000**	**5,0**	**0,000**	0,00	0,01
e.2	5	**5**	**0,000**	**5,0**	**0,000**	0,00	0,01
e.3	5	**5**	**0,000**	**5,0**	**0,000**	0,00	0,01
e.4	5	**5**	**0,000**	**5,0**	**0,000**	0,00	0,01
e.5	5	**5**	**0,000**	**5,0**	**0,000**	0,00	0,01
clr.10	25	**25**	**0,000**	**25,0**	**0,000**	0,00	14,00
clr.11	23	**23**	**0,000**	23,3	1,304	1,15	31,10
clr.12	23	**23**	**0,000**	**23,0**	**0,000**	0,00	104,00
clr.13	23	**23**	**0,000**	28,2	22,609	1,94	367,00
cyc.06	60	**60**	**0,000**	60,1	0,167	0,30	16,70

Instância	BKS	Obj Min	Gap Min (%)	Obj Med	Gap Med (%)	σ	Tempo (s)
cyc.07	144	**144**	**0,000**	149,0	3,472	3,01	58,50
cyc.08	344	351	2,035	362,1	5,262	4,35	197,00
cyc.09	780	833	6,795	843,3	8,115	6,75	894,00
cyc.10	1792	1956	9,152	1968,9	9,872	9,63	1443,00
cyc.11	4103	4585	11,748	4599,3	12,096	18,40	1802,00

Concluímos ao observar a coluna do GAP mínimo que o HGG foi capaz de alcançar a solução ótima em pelo menos uma execução para 53,33% dos problemas, enquanto foi capaz de alcançar a solução ótima em todas as execuções para 26,67% dos problemas. Em paralelo, em 48% dos problemas o desvio padrão foi 0, ou seja, o HGG foi consistente em chegar a uma mesma solução, seja ela um mínimo local ou global. De maneira complementar, também foi calculado a média dos valores de GAP médio e mínimo do HGG e da população inicial para todas as 75 instâncias, de forma a avaliar o desempenho do algoritmo no geral. O GAP médio da população inicial (gerada pelo método *greedy*) foi de 6,69% enquanto o GAP médio do HGG foi de 1,761%. E ainda, para a população inicial o HGG alcançou o GAP mínimo de 0,885%.

6 Comparação com Algoritmos Modernos

Para comprovar a eficácia do HGG foi comparado os resultados obtidos com outros algoritmos que também usaram a OR-Library como base para testes, são eles o *greedy randomized adaptive search procedure* (GRASP) [21] e o *Multi Dynamic Binary Black Hole Algorithm* (MDBBH) [9]. A comparação é focada nos problemas das classes NRE, NRF, NRG e NRH, por consequência de conterem os maiores problemas, de acordo com a tabela 1, que são o foco deste trabalho. Dessa forma, na tabela 3 estão expostos o objetivo mínimo e médio alcançado por cada algoritmo e o tempo de CPU, em segundos, que cada algoritmo levou.

Como cada meta-heurística foi executada em uma CPU diferente, foi utilizado dados do PassMark Software [19] para comparar a taxa de threads por segundo entre os processadores do HGG e do GRASP. O processador Intel Core i7-4700MQ @ 2.40GHz [21], usado para o GRASP, possui um *single thread rating* de 1768 enquanto o processador Intel Core i5-8250U @ 1.60GHz, usado para o HGG, possui um *single thread rating* de 1943. Logo, tendo como referência o tempo do HGG, foi multiplicado o tempo do GRASP por $1768/1943 = 0,91$. O mesmo não pode ser feito com o MDBBH pois não há informações da CPU utilizada. Como resultado, observa-se ao comparar tempo de execução que o GRASP obteve o menor tempo para todos os casos, levando, em média, 15 vezes menos tempo do que o HGG para executar e 7 vezes menos tempo do que o MDBBH.

Observando a tabela 3 (melhores resultados de cada instância encontram-se destacados em negrito) constata-se que o HGG e o GRASP encontraram o melhor resultado possível em pelo menos uma das execuções em 13 dos 20 problemas, ou seja, 65% das vezes. Enquanto isso, o MDBBH fez o mesmo em 11 dos 20 problemas, ou seja, 55% das vezes, apresentando o pior resultado. Entretanto, a média dos objetivos mínimos mostra que o HGG apresentou um melhor resultado do que os outros dois algoritmos nesse quesito.

Analisando o objetivo médio, o HGG obteve o melhor resultado (mesmo que ocorra um empate) em 12 casos, seguido pelo GRASP com 8 dos casos e por fim o MDBBH com 6 dos casos. Este resultado também pode ser visualizado no gráfico da figura 4, sendo que, ao calcular os GAPs médios foi encontrado 1,43% para o HGG, contra 2,31% para o GRASP e 2,91% para o MDBBH. . Sob outro ponto de vista, no gráfico da figura 5 encontra-se uma análise estatística do desempenho dos algoritmos em todos os problemas das classes NRE, NRF, NRG e NRH, apontando maior consistência do HGG em apresentar bons resultados, dado que obteve a menor mediana, os menores quartis e o menor limite superior.

7 Conclusão

Este trabalho desenvolveu uma meta-heurística para solucionar não somente o problema de posicionamento de agregadores em *smart grids* como SCPs de forma geral, visto que o problema do planejamento de *smart grids* pode ser modelado como um SCP sem qualquer perda de informação ou obstáculo. O

Tabela 3: Resultados do HGG e outras meta-heurísticas para algumas classes de problemas da OR-Library

		HGG			GRASP			MDBBH		
Instância	BKS	Obj min	Obj med	Tempo (s)	Obj min	Obj med	Tempo (s)	Obj min	Obj med	Tempo (s)
nre.1	29	**29**	**29,0**	26,1	**29**	**29,0**	**0,12**	**29**	**29,0**	87
nre.2	30	**30**	30,8	138,0	**30**	**30,5**	**1,18**	31	31,6	127
nre.3	27	**27**	**27,0**	36,0	**27**	27,8	**4,17**	**27**	27,4	118
nre.4	28	**28**	28,4	234,0	**28**	**28,1**	**0,81**	**28**	29,1	127
nre.5	28	**28**	**28,0**	32,3	**28**	**28,0**	**0,27**	**28**	**28,0**	114
nrf.1	14	**14**	**14,0**	103,0	**14**	14,2	**0,18**	**14**	14,1	116
nrf.2	15	**15**	**15,0**	18,2	**15**	**15,0**	**0,07**	**15**	15,3	93
nrf.3	14	**14**	**14,5**	155,0	**14**	14,8	**1,17**	**14**	14,8	125
nrf.4	14	**14**	**14,0**	65,7	**14**	14,2	**0,84**	**14**	14,9	101
nrf.5	13	**13**	**13,0**	160,0	**13**	13,8	**0,04**	14	14,1	125
nrg.1	176	**176**	177,6	364,0	**176**	**177,2**	**73,07**	177	178,5	187
nrg.2	154	**156**	**157,5**	600,0	**156**	**157,5**	**53,96**	157	160,6	206
nrg.3	166	**168**	**168,8**	569,0	169	171,0	**50,77**	**168**	170,4	209
nrg.4	168	170	172,9	494,0	172	174,3	**7,54**	**169**	**170,9**	213
nrg.5	168	**168**	170,4	588,0	170	172,5	**60,60**	**168**	**169,8**	182
nrh.1	63	**64**	**64,3**	481,0	**64**	65,4	**39,49**	**64**	64,9	186
nrh.2	63	**64**	64,5	472,0	**64**	65,0	**39,58**	**64**	**64,0**	154
nrh.3	59	61	61,0	413,0	60	61,0	**30,39**	**59**	**60,0**	179
nrh.4	58	59	59,9	552,0	**58**	**59,6**	**43,40**	59	60,4	201
nrh.5	55	**55**	**55,4**	637,0	**55**	55,8	**33,49**	**55**	56,4	159
Média:	67,10	**67,65**	**68,30**	306,92	67,80	68,73	**22,06**	67,70	68,71	150,45

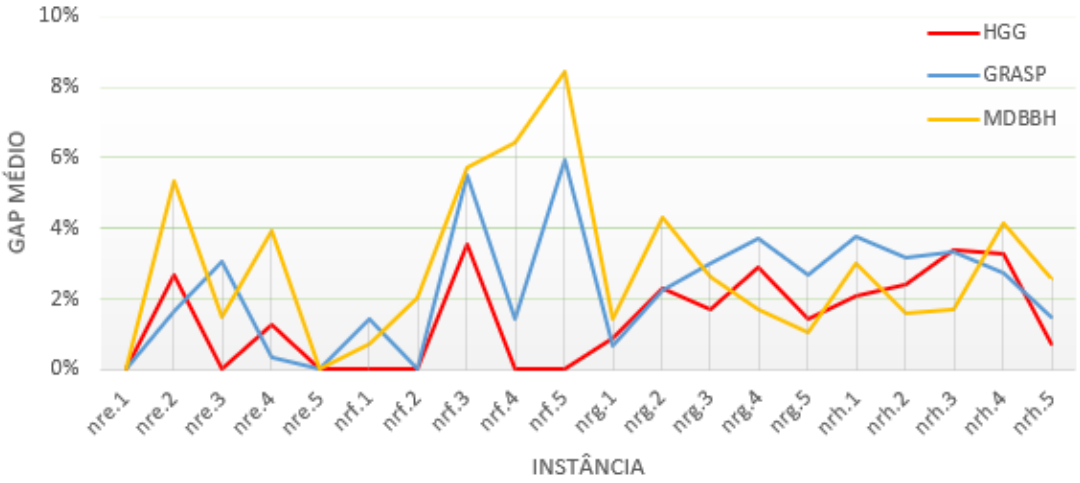


Figura 4: Gráfico de Linhas para comparação do GAP médio dos algoritmos.

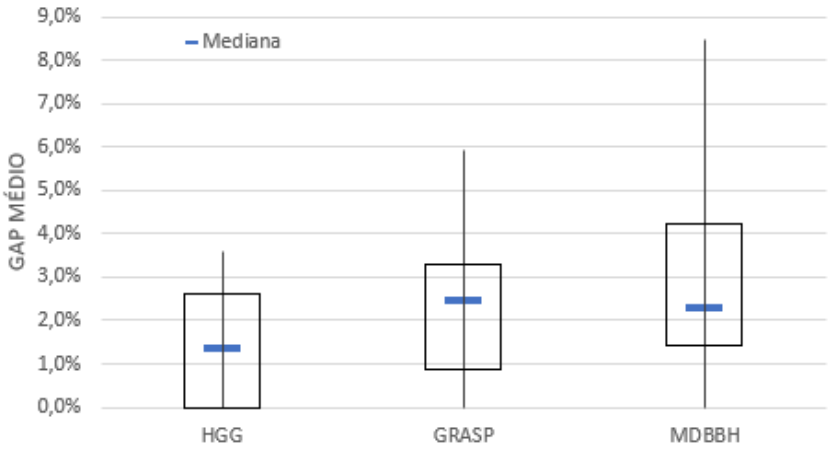


Figura 5: Gráfico Boxplot de comparação entre os algoritmos.

algoritmo HGG apresenta a inicialização, mutação e recombinação auxiliadas pelo método *greedy*, o que é uma inovação em relação a outros AGs presentes na literatura.

Os resultados computacionais apontam que o algoritmo é capaz de encontrar uma solução ótima em problemas de pequena e média escala e soluções próximas da ótima em problemas de grande escala, dado que foram alcançadas as soluções ótimas em 40 instâncias num total de 75. Comprova-se a eficácia do algoritmo mediante problemas de cobertura de conjunto quando se compara seus resultados com o de outras meta-heurísticas modernas, apresentando média do GAP mínimo e médio menor que o GRASP e o MDBBH. Entretanto, para problemas médios e pequenos o HGG pode levar consideravelmente mais tempo do que estas meta-heurísticas.

A metodologia desenvolvida tem potencial para ser ainda mais aprimorada tanto no método *greedy* quanto no método baseado no Algoritmo Genético. Trabalhos futuros podem dar continuidade ao algoritmo desenvolvido ou criar um novo aplicando conceitos empregados no HGG, com foco em diminuir o tempo de execução e melhorar os resultados para problemas de grande porte. Uma possibilidade para pesquisas futuras seria modificar a recombinação, de modo que seja mais raro gerar uma nova solução igual aos pais, a fim de aumentar a variabilidade genética na etapa de mutação e evitar mínimos locais.

Agradecimentos

Os autores agradecem as recomendações dos revisores que enriqueceram esse trabalho de pesquisa. Os autores agradecem ainda ao Ifes pelo financiamento parcial deste trabalho, Edital PRPPG 03/2020 Pibiti/Piviti, projeto nº PT00008999.

References

[1] Fariba Aalamifar, Ghasem Naddafzadeh Shirazi, Moslem Noori, and Lutz Lampe. Cost-efficient data aggregation point placement for advanced metering infrastructure. In *2014 IEEE International conference on smart grid communications (SmartGridComm)*, pages 344–349. IEEE, 2014.

[2] T. Alquthami, A. AlAmoudi, Abdullah Alsubaie, Abdulrahman Bin Jaber, Nassir Alshlwan, Murad Anwar, and Shafi Al Husaien. Analytics framework for optimal smart meters data processing. *Electrical Engineering*, 102(3):1241–1251, 2020.

[3] John E Beasley. Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072, 1990.

[4] John E Beasley and Paul C Chu. A genetic algorithm for the set covering problem. *European journal of operational research*, 94(2):392–404, 1996.

[5] Alberto Caprara, Matteo Fischetti, and Paolo Toth. A heuristic method for the set covering problem. *Operations research*, 47(5):730–743, 1999.

[6] Fabio Colombo, Roberto Cordone, and Guglielmo Lulli. A variable neighborhood search algorithm for the multimode set covering problem. *Journal of Global Optimization*, 63(3):461–480, 2015.

[7] Ademir Aparecido Constantino, PA dos Reis, CFX de Neto, and Mauricio Fernandes Figueiredo. Aplicação de algoritmos genéticos ao problema de cobertura de conjunto. *XXXV SBPO A Pesquisa Operacional e os Recursos Renováveis*, 4, 2003.

[8] Reza Zanjirani Farahani, Nasrin Asgari, Nooshin Heidari, Mahtab Hosseininia, and Mark Goh. Covering problems in facility location: A review. *Computers & Industrial Engineering*, 62(1):368–407, 2012.

[9] José García, Broderick Crawford, Ricardo Soto, and Pablo García. A multi dynamic binary black hole algorithm applied to set covering problem. In *International Conference on Harmony Search Algorithm*, pages 42–51. Springer, 2017.

[10] Michael R Garey. Computers and intractability: A guide to the theory of np-completeness. *Revista Da Escola De Enfermagem Da USP*, 44(2):340, 1979.

[11] Jenniffer Guerrero, Wilfredo Alfonso, and Eduardo Caicedo Bravo. A data analytics/big data framework for advanced metering infrastructure data. *Sensors*, 21(16):5650, 08 2021.

[12] Mhand Hifi, Vangelis Th Paschos, and Vassilis Zissimopoulos. A neural network for the minimum set covering problem. *Chaos, Solitons & Fractals*, 11(13):2079–2089, 2000.

[13] CGEE Redes Elétricas Inteligentes. Contexto nacional. *Centro de Gestão e Estudos Estratégicos*, 16:172, 2012.

[14] Guanghui Lan, Gail W DePuy, and Gary E Whitehouse. An effective and simple heuristic for the set covering problem. *European journal of operational research*, 176(3):1387–1403, 2007.

[15] João F Martins, Anabela Gonçalves Pronto, Vasco Delgado-Gomes, and Mihai Sanduleac. Smart meters and advanced metering infrastructure. In *Pathways to a Smarter Power System*, pages 89–114. Elsevier, 2019.

[16] Antti Mutanen, Pertti Jarventausta, Matti Karenlampi, and Pentti Juuti. Improving distribution network analysis with new amr-based load profiles. 2013.

[17] Aziz Naamane and NK Msirdi. Towards a smart grid communication. *Energy Procedia*, 83:428–433, 2015.

[18] Byron O Palate, Tatiana P Guedes, Ahda Grilo-Pavani, Antonio Padilha-Feltrin, and Joel D Melo. Aggregator units allocation in low voltage distribution networks with penetration of photovoltaic systems. *International Journal of Electrical Power & Energy Systems*, 130:107003, 2021.

[19] PassMark. Cpu benchmarks. *PassMark Software Pty Ltda. Disponível em https://www.cpubenchmark.net/compare/*, Acesso em 11 de Outubro de 2021.

[20] Charles ReVelle, Constantine Toregas, and Louis Falkson. Applications of the location set-covering problem. *Geographical analysis*, 8(1):65–76, 1976.

[21] Victor Reyes and Ignacio Araya. A grasp-based scheme for the set covering problem. *Operational Research*, pages 1–18, 2019.

[22] Guilherme Rolim, Célio Vinicius Neves de Albuquerque, and Igor Monteiro Moraes. Modelo e solução para o problema de posicionamento de agregadores em redes elétricas inteligentes. 2015.

[23] Zaire de Assis Ferreira Souza, Jorge Henrique Angelim, and Carolina Mattos Affonso. Identificação do número mínimo de unidades de medição fasorial sincronizada (pmu) para o monitoramento em tempo real da margem de estabilidade de tensão.

[24] Yi Wang, Qixin Chen, Tao Hong, and Chongqing Kang. Review of smart meter data analytics: Applications, methodologies, and challenges. *IEEE Transactions on Smart Grid*, 10(3):3125–3148, 2019.

[25] Todd Zhen, Tarek Elgindy, SM Shafiul Alam, Bri-Mathias Hodge, and Carl D Laird. Optimal placement of data concentrators for expansion of the smart grid communications network. *IET Smart Grid*, 2(4):537–548, 2019.