Hyperparameter Optimization of the Genetic Algorithm Applied to the Economic Dispatch Problem

Giovanna Sboldrim Pascolat
Departamento de Engenharia Elétrica
Universidade Estadual Paulista
Bauru, Brasil
g.pascolat@unesp.br

Fabiano Borges da Silva
Departamento de Matemática
Universidade Estadual Paulista
Bauru, Brasil
fabiano.borges@unesp.br

Leonardo Nepomuceno
Departamento de Engenharia Elétrica
Universidade Estadual Paulista
Bauru, Brasil
leonardo.nepomuceno@unesp.br

***Abstract*—This paper presents a detailed study on hyperparameter optimization for genetic algorithms applied to the Economic Dispatch Problem (EDP) in electric power systems. Using the Optuna library for hyperparameter optimization, specific parameters were tuned for different systems with 3, 13, 19, and 40 generating units. The results obtained show that combining genetic algorithms with hyperparameter optimization via Optuna can yield competitive results, surpassing, in some cases, one of the best values found in the literature. The use of elitism also proved effective in improving the algorithm's performance, highlighting the importance of customizing parameters for each specific system.**

***Keywords*—Optimization; Metaheuristics; Genetic Algorithm; Hyperparameter optimization; Power generation dispatch.**

I. Introdução

As Metaeurísticas, tais como algoritmos evolutivos e bio-inspirados, entre os quais se destacam os Algoritmos Genéticos (AGs), têm sido amplamente utilizadas para resolver problemas complexos de otimização, devido à sua capacidade de explorar grandes espaços de busca e encontrar soluções próximas do ótimo. As meta-heurísticas têm sido usualmente aplicadas a problemas de otimização multimodais, e/ou problemas que apresentem descontinuidades, e/ou funções não convexas e/ou não diferenciáveis. Para esse tipo de problema complexo, os métodos de otimização determinísticos convencionais não são aplicáveis e/ou não são eficientes.

O Problema de Despacho Econômico (PDE) tem como objetivo calcular um despacho de geração que minimiza os custos de produção, atendendo à demanda do sistema e às restrições operacionais dos geradores. O PDE pode ser formulado como um problema multimodal (i.e. podem ocorrer milhares de ótimos locais), com função objetivo não diferenciável, e possível descontinuidade em seu domínio. Assim, devido às dificuldades inerentes ao problema do PDE, as meta-heurísticas têm sido amplamente utilizadas para a solução deste. Dentre as Metaheurísticas aplicadas para a solução do PDE, destacam-se: Evolução Diferencial, Enxame de Partículas, Algoritmo Genético, Algoritmo de Abelhas, Algoritmo de Formigas e diversos híbridos, como pode ser visto em Visutarron e Chiang (2024) [12] e Zaman et al (2015) [13], Mahor, Prasad e Rangnekar (2009) [5], Chen e Chang (1995) [3], Secui (2015) [9], Hou et al (2002) [4] e Sen e Mathur (2016) [10], respectivamente.

Os AGs são um dos algoritmos que têm sido utilizados com mais sucesso para a solução do PDE. No entanto, a eficiência dos AGs depende fortemente da escolha adequada de hiperparâmetros, como taxa de crossover, taxa de mutação, número de indivíduos na população, entre outros. Este artigo apresenta uma abordagem inovadora para a otimização de hiperparâmetros de AGs utilizando a biblioteca Optuna, que permite a personalização dos parâmetros de forma específica para cada sistema de geração. Em essência, Optuna automatiza o processo de tentativa e erro e é utilizado para encontrar os melhores valores de hiperparâmetros de forma eficiente, proporcionando um ajuste fino que pode melhorar consideravelmente o desempenho do AG.

A principal contribuição deste trabalho consiste em mostrar que a otimização de hiperparâmetros via Optuna pode levar a resultados competitivos e, em alguns casos, superiores aos melhores valores reportados na literatura. A análise é realizada para quatro sistemas diferentes: 3, 13, 19 e 40 unidades geradoras. Para cada sistema, os parâmetros do AG foram otimizados utilizando Optuna, e os resultados foram comparados com os melhores valores encontrados nas dissertações de Pavan (2023) [7] e Silva (2014) [11].

A seguir, apresentamos as etapas do algoritmo genético utilizado, o framework Optuna, os resultados obtidos com a otimização via Optuna e uma discussão detalhada sobre o desempenho do AG em comparação com os valores de referência da literatura. O uso do elitismo como uma estratégia para preservar os melhores indivíduos durante o processo evolutivo também é abordado, evidenciando sua eficácia na melhoria do desempenho do AG.

Este estudo destaca a importância da personalização de hiperparâmetros para cada sistema específico e a vantagem do uso de técnicas avançadas de otimização, como a oferecida pelo Optuna. Os resultados obtidos mostram que a combinação

de AGs com a otimização de hiperparâmetros pode ser uma abordagem promissora para a resolução de problemas de despacho econômico de energia elétrica.

II. Modelo de Despacho Econômico

O Problema de Despacho Econômico (PDE) é amplamente investigado e consiste em determinar o despacho ótimo de potência ativa P_i de cada unidade geradora i, buscando minimizar o custo total de produção, de tal forma que as potências geradas estejam dentro dos limites mínimo e máximo operacionais de cada unidade geradora dados por $P_i^{\min}$ e $P_i^{\max}$ respectivamente, a fim de suprir uma determinada demanda D.

Na formulação clássica, o PDE é modelado como um problema de otimização com função objetivo convexa e quadrática. Assim, tem-se que o PDE clássico é descrito pelo modelo:

$$\min \sum_{i=1}^{n_G} \left[a_i P_i^2 + b_i P_i + c_i\right] \tag{1}$$

sujeito à:

$$\sum_{i=1}^{n_G} P_i = D, \quad P_i^{\min} \leq P_i \leq P_i^{\max}, \quad i = 1, \ldots, n_G \tag{2}$$

em que n_G é a quantidade de geradores do sistema, D é a demanda a ser atendida, P_i é a potência ativa gerada pelo gerador i e a_i, b_i e c_i são os coeficientes da função de custo do gerador i, cujos limites da capacidade de geração são dados por $P_i^{\min}$ e $P_i^{\max}$, nessa ordem.

Para tornar o modelo mais realista, é necessário considerar fatores adicionais que afetam os custos de produção. Nos geradores termelétricos, por exemplo, existem válvulas de admissão de calor cuja abertura parcial provoca perdas de energia através da passagem de vapor, aumentando os custos de produção. Quanto mais estrangulado o vapor, isto é, quanto menor a abertura da válvula, menor o rendimento da mesma. Para melhorar o rendimento global, o vapor é controlado por múltiplas válvulas parciais. Portanto, no momento de abertura de cada uma dessas válvulas parciais há uma alteração na curva de rendimento global e, por conseguinte, na curva de custos. Este fenômeno é chamado de efeito de ponto de carregamento de válvula, e sua inserção no problema de PDE resulta no problema de despacho denominado PDEPV.

Na formulação do modelo de PDEPV, temos as mesmas restrições do problema de PDE clássico, porém sua função objetivo possui um termo que corresponde ao módulo de uma senoide, que representa os pontos de carregamento de válvula. Assim, tem-se que o PDEPV é descrito pelo modelo:

$$\min \sum_{i=1}^{n_G} \left[a_i P_i^2 + b_i P_i + c_i + \left|e_i \sin\left(f_i(P_i^{\min} - P_i)\right)\right|\right] \tag{3}$$

s.a.:

$$\sum_{i=1}^{n_G} P_i = D, \quad P_i^{\min} \leq P_i \leq P_i^{\max}, \quad i = 1, \ldots, n_G \tag{4}$$

em que a_i, b_i, c_i, e_i e f_i são os coeficientes da curva de custo de combustível do gerador i. Para a resolução do problema neste artigo, será adotado o modelo PDEPV em vez do PDE clássico.

Os dados detalhados para os sistemas de 3, 13, 19 e 40 geradores que serão utilizados neste trabalho, incluindo os limites mínimos e máximos de geração de potência para cada unidade geradora, os coeficientes das funções de custo e os parâmetros associados aos efeitos de ponto de carregamento de válvula, estão apresentados em Anexos, respectivamente, nas Tabelas V, VI, VII e VIII.

III. Algoritmo Genético

Os Algoritmos Genéticos (AGs) são técnicas de otimização inspiradas nos processos de seleção natural e genética de Charles Darwin. Eles são amplamente utilizados para resolver problemas complexos de otimização devido à sua capacidade de explorar grandes espaços de solução e encontrar soluções próximas do ótimo. A seguir, são descritas as principais etapas de um AG aplicado ao Problema de Despacho Econômico (PDE).

A. Representação Cromossomial

Cada solução potencial para o problema é representada como um cromossomo. No contexto do AG, um cromossomo é uma sequência de genes, onde cada gene representa a potência ativa gerada por um gerador específico. Essa representação facilita a manipulação das soluções durante o processo evolutivo.

B. Inicialização da População

A população inicial de cromossomos é gerada aleatoriamente dentro dos limites operacionais dos geradores. Esta população deve ser suficientemente grande e diversificada para cobrir uma ampla gama de possíveis soluções, garantindo que o algoritmo tenha um bom ponto de partida para a busca.

A lógica implementada em Python para inicializar a população é mostrada abaixo:

```
def inicializa_populacao(dim, nP, Xmin, Xmax):
    return Xmin + np.random.rand(nP, dim) * (
        Xmax - Xmin)
```

Listing 1. Função para inicializar a população

Essa função gera uma população inicial de indivíduos, onde cada indivíduo é um vetor de potências geradas pelos geradores, garantindo que os valores estejam dentro dos limites operacionais especificados por Xmin e Xmax.

C. Avaliação da População

Cada cromossomo na população é avaliado por meio de uma função de aptidão, que calcula o custo de produção de energia elétrica associado a essa solução. No caso do PDEPV, essa função inclui termos quadráticos e lineares dos custos, além de um termo modular que representa os efeitos de ponto de carregamento de válvula. A função de aptidão assegura que as soluções atendam às restrições de demanda e limites operacionais dos geradores.

A lógica implementada em Python para avaliar a população é mostrada abaixo:

```
def avaliacaoPOP(X, custo):
nP, dim = X.shape
F = np.zeros(nP)
for p in range(nP):
sum_ = 0
for g in range(dim):
a = custo['a'][g]
b = custo['b'][g]
c = custo['c'][g]
e = custo['e'][g]
f = custo['f'][g]
Pg = X[p, g]
Pmin = custo['Pmin'][g]
sum_ += a * Pg**2 + b * Pg + c + abs(e * np.
    sin(f * (Pmin - Pg)))
F[p] = sum_
return F
```

Essa função calcula o custo de operação para cada indivíduo na população, considerando os coeficientes de custo e os parâmetros específicos de cada gerador.

D. Seleção dos Pais

Os pais são selecionados para gerar novos cromossomos com base na sua aptidão. Um método comum de seleção é a seleção por torneio, onde grupos de indivíduos são escolhidos aleatoriamente da população, e o mais apto dentre eles é selecionado como pai. Este processo é repetido até que o número necessário de pais seja selecionado, garantindo que os melhores indivíduos tenham uma maior chance de se reproduzir. A lógica implementada aqui é a seguinte função:

```
def selecao_torneio(populacao, aptidao, k=3):
nP = len(populacao)
pais = np.empty_like(populacao)
for i in range(nP):
competidores = np.random.choice(nP, k)
melhor = np.argmin(aptidao[competidores])
pais[i] = populacao[competidores[melhor]]
return pais
```

E. Recombinação e Mutação

Após a seleção dos pais, novos cromossomos (filhos) são gerados através da recombinação (crossover) e mutação. A recombinação combina pares de pais para criar novos cromossomos, permitindo que as boas características dos pais sejam transmitidas para os filhos. A mutação introduz pequenas alterações aleatórias nos genes dos cromossomos, ajudando a manter a diversidade genética na população e evitando a convergência prematura para soluções subótimas.

A lógica implementada em Python para a recombinação (crossover) é mostrada abaixo:

```
def crossover(pais, taxa_crossover=0.8):
nP, dim = pais.shape
filhos = np.empty((nP, dim))
for i in range(0, nP - 1, 2):
if np.random.rand() < taxa_crossover:
ponto_corte = np.random.randint(1, dim)
filhos[i,
] = pais[i,
]
filhos[i, ponto_corte:] = pais[i + 1,
    ponto_corte:]
filhos[i + 1,
] = pais[i + 1,
]
filhos[i + 1, ponto_corte:] = pais[i,
    ponto_corte:]
else:
filhos[i, :] = pais[i, :]
filhos[i + 1, :] = pais[i + 1, :]
if nP % 2 != 0:
filhos[-1, :] = pais[-1, :]
return filhos
```

Essa função realiza o crossover entre pares de pais, combinando seus genes para gerar novos filhos. A probabilidade de crossover é determinada pela taxa de crossover.

A lógica implementada em Python para a mutação é mostrada abaixo:

```
def mutacao(filhos, Xmin, Xmax, taxa_mutacao
    =0.2):
nP, dim = filhos.shape
for i in range(nP):
if np.random.rand() < taxa_mutacao:
gene = np.random.randint(dim)
filhos[i, gene] = Xmin[gene] + np.random.rand
    () * (Xmax[gene] - Xmin[gene])
return filhos
```

Essa função aplica a mutação nos filhos, introduzindo variações aleatórias nos genes de alguns indivíduos com uma probabilidade determinada pela taxa de mutação.

F. Substituição da População

Após a geração dos filhos, a população atual é substituída pelos novos indivíduos. Um mecanismo de elitismo pode ser utilizado para garantir que os melhores indivíduos da população atual sejam preservados na nova geração. Este processo é repetido por um número definido de gerações ou até que uma condição de parada seja atingida, como alcançar um número máximo de gerações ou atingir uma solução de qualidade satisfatória.

A lógica implementada em Python para a execução do algoritmo genético, incluindo a substituição da população, é mostrada abaixo:

```
def rodar_ga():
populacao = inicializa_populacao(dim, nP, Xmin
    , Xmax)
populacao = factibiliza_pop(populacao, demanda
    , Xmin, Xmax)


aptidao = avaliacaoPOP(populacao, custo)

for geracao in range(num_geracoes):
    indices_melhores = np.argsort(aptidao)[:
        num_elitismo]
    elite = populacao[indices_melhores]
    aptidao_elite = aptidao[indices_melhores]
```

```
    pais = selecao_torneio(populacao, aptidao, tamanho_torneio)

    filhos = crossover(pais, taxa_crossover)

    filhos = mutacao(filhos, Xmin, Xmax, taxa_mutacao)

    filhos = factibiliza_pop(filhos, demanda, Xmin, Xmax)

    aptidao_filhos = avaliacaoPOP(filhos, custo)

    populacao = np.vstack((elite, filhos[num_elitismo:]))
    aptidao = np.concatenate((aptidao_elite, aptidao_filhos[num_elitismo:]))

melhor_aptidao = np.min(aptidao)
melhor_individuo = populacao[np.argmin(aptidao)]
return melhor_aptidao, melhor_individuo
```

Essa função coordena a execução do algoritmo genético, incluindo a inicialização da população, avaliação, seleção, crossover, mutação, factibilização e substituição da população com elitismo.

G. Factibilização

Para garantir que as soluções geradas atendam às restrições de demanda e limites operacionais dos geradores, um processo de factibilização é aplicado. Este processo ajusta as potências geradas pelos indivíduos para que a soma total da geração atenda exatamente à demanda e os valores de potência fiquem dentro dos limites permitidos.

Nesse processo, foi criada uma função no Python que ajusta as potências geradas por um único indivíduo para que a soma total atenda à demanda e as potências individuais estejam dentro dos limites mínimo e máximo operacionais dos geradores.

Inicialmente, a função calcula a soma das potências geradas pelo indivíduo. Enquanto a diferença entre a soma das potências geradas e a demanda for maior que uma pequena tolerância, a função realiza ajustes nas potências individuais. A diferença entre a soma das potências e a demanda é distribuída igualmente entre todos os geradores. As potências são ajustadas e limitadas aos valores mínimo e máximo permitidos. O processo é repetido até que a soma das potências esteja dentro da tolerância da demanda.

A lógica implementada em Python para a factibilização de um indivíduo é mostrada abaixo:

```
def factibiliza_part(Xi, demanda, Xmin, Xmax, zona_proibida=None):
dim = len(Xi)
EPS = 1e-2


s = np.sum(Xi)
while abs(s - demanda) > EPS:
    delta = s - demanda
    deltap = delta / dim
    Xi -= deltap
    Xi = np.clip(Xi, Xmin, Xmax)

    if zona_proibida:
        for g in range(dim):
            if any(lower <= Xi[g] <= upper for lower, upper in zona_proibida[g]):
                Xi[g] = Xmin[g] + np.random.rand() * (Xmax[g] - Xmin[g])

    s = np.sum(Xi)
return Xi
```

Essa função ajusta as potências geradas por um único indivíduo para que a soma total atenda à demanda e as potências individuais estejam dentro dos limites mínimo e máximo operacionais dos geradores.

A lógica implementada em Python para a factibilização da população é mostrada abaixo:

```
def factibiliza_pop(X, demanda, Xmin, Xmax, zona_proibida=None):
nP, dim = X.shape
for p in range(nP):
X[p, :] = factibiliza_part(X[p, :], demanda, Xmin, Xmax, zona_proibida)
return X
```

Essa função aplica o processo de factibilização a cada indivíduo na população, garantindo que todos os indivíduos atendam às restrições de demanda e limites operacionais dos geradores.

IV. Biblioteca Optuna

Um hiperparâmetro é um parâmetro que controla o comportamento de um algoritmo de aprendizado de máquina. O Optuna é uma biblioteca Python de código aberto para otimização desses hiperparâmetros com software de fácil utilização e bem projetado que suporta uma variedade de algoritmos de otimização (Pravin et al, 2022 [8]). Ele busca e encontra automaticamente valores ótimos de hiperparâmetros através de tentativa e erro para alcançar excelente desempenho. Atualmente, o software pode ser utilizado em Python.

Optuna utiliza um registro histórico de tentativas para determinar quais valores de hiperparâmetros devem seguir. Usando esses dados, ele estima uma área promissora e tenta valores nessa área. Optuna então estima uma região ainda mais promissora com base no novo resultado. Ele repete esse processo usando os dados históricos das tentativas concluídas até então. Segundo Optuna [6], ele emprega um algoritmo de

otimização bayesiana chamado Estimador Parzen Estruturado em Árvore (TPE).

O TPE é um algoritmo de otimização bayesiana que busca melhorar a eficiência na seleção de hiperparâmetros. Diferentemente de métodos tradicionais, o TPE usa estimativas probabilísticas para guiar a busca pelos melhores hiperparâmetros.

O TPE começa com uma exploração aleatória do espaço de busca, onde amostras de hiperparâmetros são escolhidas aleatoriamente e avaliadas. Com base nos resultados iniciais, o espaço de busca é dividido em duas distribuições: uma que contém hiperparâmetros "bons" (que produzem modelos de alto desempenho) e outra que contém hiperparâmetros "ruins" (que produzem modelos de baixo desempenho).

A otimização continua selecionando novos hiperparâmetros que maximizam a razão entre a probabilidade de um hiperparâmetro pertencer à distribuição "boa" e à distribuição "ruim". Essa abordagem permite que o TPE se concentre em áreas promissoras do espaço de busca, aumentando a eficiência da otimização. O TPE utiliza estimadores de densidade de Kernel (KDE) para modelar as distribuições de hiperparâmetros "bons" e "ruins". A KDE cria uma distribuição suave a partir das amostras observadas, facilitando a seleção de novos hiperparâmetros. Essa biblioteca foi desenvolvida por Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta e Masanori Koyama, para consultar mais detalhes ver em [1] e Bergstra et al. (2011) [2].

Para melhor entendimento do funcionamento do Optuna, apresentamos um exemplo de como usar essa biblioteca para otimizar os hiperparâmetros de um problema simples de otimização. Sendo assim, seja o problema de minimizar a função objetivo $f(x) = (x-2)^2$, sem restrições. Para utilizar o optuna podemos sugerir valores de hiperparâmetros no intervalo de -10 a 10, por exemplo, usando o objeto `trial`. E por fim, criamos um objeto `study` e invocamos o método `optimize` por 100 tentativas. Nota-se que o melhor hiperparâmetro encontrado para x foi 2. Segue a lógica implementada em código Python:

```
import optuna

def objective(trial):
    x = trial.suggest_float('x', -10, 10)
    return (x - 2) ** 2

study = optuna.create_study()
study.optimize(objective, n_trials=100)

study.best_params  # E.g. {'x': 2.002108042}
```

Além disso, para instalar o Optuna basta colocar o seguinte código no prompt de comando utilizado:

```
pip install optuna
```

Em suma, Optuna é uma biblioteca de otimização de hiperparâmetros de código aberto que automatiza o processo de tentativa e erro e é utilizado para encontrar os melhores valores de hiperparâmetros de forma eficiente. O processo de otimização com Optuna pode ser resumido nas seguintes etapas:

1) **Definição do Espaço de Hiperparâmetros**: O usuário define o espaço de busca para cada hiperparâmetro que deseja otimizar. Isso inclui os intervalos de valores possíveis e a distribuição de probabilidade (por exemplo, uniforme, log-uniforme) para a amostragem dos hiperparâmetros.
2) **Execução de Ensaios**: Optuna realiza uma série de experimentos (chamados de ensaios), onde cada ensaio consiste em amostrar um conjunto de valores de hiperparâmetros do espaço de busca e avaliar a performance do modelo com esses valores.
3) **Algoritmo de Otimização**: Optuna utiliza algoritmos de otimização, como o algoritmo de otimização bayesiana para guiar a amostragem dos hiperparâmetros de maneira a maximizar (ou minimizar) a função objetivo de forma eficiente.
4) **Avaliação e Melhoria**: A performance do modelo é avaliada para cada conjunto de hiperparâmetros amostrados, e os resultados são usados para atualizar o modelo de busca, melhorando continuamente a seleção dos próximos conjuntos de hiperparâmetros a serem testados.

V. Resultados e Discussões

Para avaliar a performance do Algoritmo Genético (AG) antes da otimização de hiperparâmetros, foram definidos os mesmos parâmetros padrão para todos os sistemas de gerações. Os hiperparâmetros (padrão) escolhidos são:

- Número de indivíduos na população (`nP`): 400
- Taxa de crossover (`taxa_crossover`): 0.9
- Taxa de mutação (`taxa_mutacao`): 0.2
- Número de gerações (`num_geracoes`): 1000
- Número de elitismo (`num_elitismo`): 1
- Tamanho do torneio (`tamanho_torneio`): 2

Os resultados obtidos utilizando a otimização de hiperparâmetros com o pacote Optuna (em Python) são apresentados a seguir. Optuna foi empregado para ajustar os parâmetros do algoritmo genético de forma a obter os melhores resultados possíveis para cada sistema de geração. Para cada sistema, foram realizados 100 ensaios.

- **Melhores hiperparâmetros encontrados para o sistema de 3 geradores**:
  - Número de indivíduos na população: 442
  - Taxa de crossover: 0.9645715284288756
  - Taxa de mutação: 0.2599314013401837
  - Número de gerações: 1077
  - Número de elitismo: 29
  - Tamanho do torneio: 2
- **Melhores hiperparâmetros encontrados para o sistema de 13 geradores**:
  - Número de indivíduos na população: 297
  - Taxa de crossover: 0.8225657306175698
  - Taxa de mutação: 0.2676038481695615

  - Número de gerações: 1739
  - Número de elitismo: 20
  - Tamanho do torneio: 3
- **Melhores hiperparâmetros encontrados para o sistema de 19 geradores**:
  - Número de indivíduos na população: 500
  - Taxa de crossover: 0.9903729893420303
  - Taxa de mutação: 0.26273935718541885
  - Número de gerações: 1242
  - Número de elitismo: 39
  - Tamanho do torneio: 3
- **Melhores hiperparâmetros encontrados para o sistema de 40 geradores**:
  - Número de indivíduos na população: 475
  - Taxa de crossover: 0.5002644112569006
  - Taxa de mutação: 0.2808035429383964
  - Número de gerações: 876
  - Número de elitismo: 38
  - Tamanho do torneio: 4

Tanto os hiperparâmetros (padrão) como os que foram otimizados e encontrados pelo Optuna foram aplicados aos sistemas de 3, 13, 19 e 40 geradores, Os resultados obtidos, em relação ao valor da função objetivo, para os hiperparâmetros (padrão) e para os hiperparâmetros otimizados são apresentados, respectivamente, na Tabela I e II a seguir.

TABLE I
ESTATÍSTICAS DOS RESULTADOS DO ALGORITMO GENÉTICO NÃO OTIMIZADO

Geradores	Média	Menor Valor	Desvio Padrão
3	8234.4263	8233.9408	1.03564
13	24235.6509	24197.0707	20.03945
19	19098.6659	19075.4377	13.4432
40	121679.45124	121431.25084	119.6450

TABLE II
ESTATÍSTICAS DOS RESULTADOS DO ALGORITMO GENÉTICO OTIMIZADO

Geradores	Média	Menor Valor	Desvio Padrão
3	8234.1243	8233.8888	1.2499
13	24232.2964	24171.4602	29.4519
19	17070.3224	16996.4506	33.1275
40	121591.2864	121380.8971	130.1316

Ao comparar as Tabelas I com II, claramente os resultados melhoraram e o Optuna provou o seu funcionamento.

Além disso, a Tabela III a seguir mostra os melhores resultados encontrados nas dissertações de Pavan (2023) e Silva (2014).

TABLE III
MELHORES RESULTADOS, SEGUNDO PAVAN (2023) E SILVA (2014).

Geradores	Média	Menor Valor	Desvio Padrão
3	-	8234.07	-
13	24228.3647	24169.9177	69.1505
19	16952.9393	16945.6023	10.9538
40	121496.0788	121476.6736	35.3006

Nota-se que os resultados obtidos com a otimização usando o algoritmo genético e os melhores hiperparâmetros identificados pelo Optuna mostram um desempenho promissor em comparação com os melhores valores encontrados na literatura. A seguir, temos uma análise mais detalhada dos resultados obtidos para cada sistema de geradores e uma comparação com os valores de referência da literatura.

Os melhores hiperparâmetros encontrados para o sistema de 3 unidades geradoras resultaram em uma média dos melhores valores de 8234.1243, com o menor valor encontrado sendo 8233.8888 e um desvio padrão de 1.2499. Comparando esses resultados com os mencionados na dissertação de SILVA (2014), onde o menor valor foi 8234.07, podemos observar que o algoritmo genético apresentou um desempenho ligeiramente superior em termos de menor valor encontrado.

Para o sistema de 13 unidades geradoras, os melhores hiperparâmetros levaram a uma média dos melhores valores de 24232.2964, com o menor valor encontrado sendo 24171.4602 e um desvio padrão de 29.4519. Em comparação, a dissertação de Pavan (2023) reporta um menor valor de 24169.9177, uma média de 24228.3647 e um desvio padrão de 69.1505. Embora o menor valor encontrado na literatura seja ligeiramente inferior, o algoritmo genético apresentou uma média um pouco superior, mas com um desvio padrão significativamente menor, indicando maior consistência nos resultados obtidos.

Para o sistema de 19 unidades geradoras, os melhores hiperparâmetros resultaram em uma média dos melhores valores de 17070.3224, com o menor valor encontrado sendo 16996.4506 e um desvio padrão de 33.1275. Em comparação com os resultados da dissertação de Pavan (2023), onde o menor valor foi 16945.6023, a média foi 16952.9393 e o desvio padrão foi 10.9538, observa-se que o menor valor encontrado pela literatura é inferior ao do algoritmo genético, assim como a média e o desvio padrão são melhores na literatura.

Para o sistema de 40 unidades geradoras, os melhores hiperparâmetros encontrados resultaram em uma média dos melhores valores de 121591.2864, com o menor valor encontrado sendo 121380.8971 e um desvio padrão de 130.1316. Comparando com a dissertação de Pavan (2023), que reporta um menor valor de 121476.6736, uma média de 121496.0788 e um desvio padrão de 35.3006, nota-se que o menor valor encontrado pelo algoritmo genético é inferior, indicando que o AG apresentou um desempenho superior em termos de menor valor encontrado.

Os valores dos despachos obtidos para o sistema de 3, 12, 19 e 40 unidades geradoras do Algoritmo Genético otimizado estão a seguir (mesmo que não precise, a precisão de mais casas decimais foi adotada para atender a demanda, mas basta arredondar para a precisão de duas casas):

TABLE IV
DESPACHO PARA O SISTEMA DE 3, 13, 19 E 40 GERADORES

Gerador	$P_{G,k}$
1	300.2568
2	149.7332
3	399.9999

Gerador	$P_{G,k}$
1	628.3184
2	298.9568
3	298.8303
4	159.7299
5	159.7317
6	159.7319
7	159.7331
8	159.7296
9	159.7298
10	75.1907
11	76.7953
12	91.7197
13	91.7962

Gerador	$P_{G,k}$
1	108.72665546
2	370.06680069
3	248.0489433
4	25.0
5	62.73767355
6	267.49342752
7	61.01015262
8	130.20768363
9	208.35531483
10	39.98312299
11	149.96991691
12	74.99839811
13	63.72762268
14	94.970337
15	212.68433776
16	79.99642425
17	80.0
18	228.71058823
19	401.30260423

Gerador	$P_{G,k}$
1	110.8258933
2	111.04646269
3	97.43804677
4	179.80994756
5	89.03910369
6	139.99698665
7	259.81390572
8	284.66413105
9	284.62090396
10	130.15370023
11	168.82929252
12	168.84703658
13	214.85295957
14	394.26227688
15	304.52082571
16	394.2805856
17	489.44103021
18	489.28037187
19	511.33300578
20	511.34408599
21	523.5540123
22	523.28084817
23	523.34588358
24	523.4853172
25	523.35733303
26	523.28012871
27	10.00854919
28	10.04930076
29	10.06109983
30	88.77069493
31	190.0
32	190.0
33	189.9921732
34	164.87231146
35	164.9727017
36	165.1801169
37	110.0
38	110.0
39	110.0
40	511.37919266

VI. CONSIDERAÇÕES FINAIS

Os resultados mostram que o uso de algoritmos genéticos, combinado com a otimização de hiperparâmetros via Optuna e a introdução do elitismo, pode produzir resultados competitivos e, em alguns casos, superiores aos melhores valores encontrados na literatura. A utilização do elitismo, como um parâmetro que permite escolher quantos melhores indivíduos manter, demonstrou ser uma estratégia eficaz para melhorar o desempenho do método. No entanto, a variabilidade dos resultados (desvio padrão) em alguns casos indica que ainda há espaço para melhorias na consistência do algoritmo. A continuidade da pesquisa e a exploração de outras técnicas de otimização podem levar a ainda melhores resultados.

Um diferencial significativo deste estudo é a utilização do Optuna para a otimização dos hiperparâmetros de cada sistema específico. O Optuna, uma biblioteca robusta e eficiente de otimização bayesiana, permitiu a personalização dos parâmetros do algoritmo genético de forma precisa para cada sistema de geradores. Esse ajuste fino proporcionou um desempenho superior e mais consistente, evidenciando a vantagem de utilizar técnicas avançadas de otimização de hiperparâmetros. A aplicação do Optuna demonstrou ser uma abordagem valiosa e promissora, destacando-se como um diferencial importante neste estudo.

Além disso, o estudo revelou um comportamento interessante do algoritmo genético: à medida que o número de geradores aumenta, o número de gerações necessárias para atingir o ótimo tende a diminuir. Por exemplo, o Optuna sugeriu 1.739 gerações para o sistema com 13 geradores, 1.242 para o sistema com 19 geradores e apenas 876 gerações para o sistema com 40 geradores. Isso indica que o Optuna ajusta os parâmetros de forma a otimizar não apenas o desempenho do algoritmo, mas também a eficiência computacional, economizando recursos ao reduzir o número de iterações em sistemas mais complexos. Assim, o uso do Optuna demonstra ser uma abordagem valiosa e promissora, destacando-se como um diferencial importante neste estudo.

VII. ANEXOS

A. Dados dos sistemas de 3, 13, 19 e 40 geradores

TABLE V
DADOS DO SISTEMA DE 3 GERADORES

Unidade	$P_{G,k}^{\min}$ [MW]	$P_{G,k}^{\max}$ [MW]	a_k \$/[MW]^2	b_k \$/[MW]	c_k \$	e_k \$	f_k 1/[MW]
1	100	600	0.001562	7.92	561	300	0.0315
2	50	200	0.00482	7.97	78	150	0.063
3	100	400	0.00194	7.85	310	200	0.042

Demanda: $P_D = 850\text{MW}$.

TABLE VI
DADOS DO SISTEMA DE 13 GERADORES

Unidade	$P_{G,k}^{\min}$ [MW]	$P_{G,k}^{\max}$ [MW]	a_k \$/[MW]^2	b_k \$/[MW]	c_k \$	e_k \$	f_k 1/[MW]
1	0	680	0.00028	8.1	550	300	0.035
2	0	360	0.00056	8.1	309	200	0.042
3	0	360	0.00056	8.1	307	200	0.042
4	60	180	0.00324	7.74	240	150	0.063
5	60	180	0.00324	7.74	240	150	0.063
6	60	180	0.00324	7.74	240	150	0.063
7	60	180	0.00324	7.74	240	150	0.063
8	60	180	0.00324	7.74	240	150	0.063
9	60	180	0.00324	7.74	240	150	0.063
10	40	120	0.00284	8.6	126	100	0.084
11	40	120	0.00284	8.6	126	100	0.084
12	55	120	0.00284	8.6	126	100	0.084
13	55	120	0.00284	8.6	126	100	0.084

Demanda: $P_D = 2520\text{MW}$.

TABLE VII
DADOS DO SISTEMA DE 19 GERADORES

Unidade	$P_{G,k}^{\min}$ [MW]	$P_{G,k}^{\max}$ [MW]	a_k \$/[MW]^2	b_k \$/[MW]	c_k \$	e_k \$	f_k 1/[MW]
1	100	300	0.0097	6.8	119	90	0.72
2	120	438	0.0055	4	90	79	0.05
3	100	250	0.0055	4	45	0	0
4	8	25	0.0025	0.85	0	0	0
5	50	63.75	0	5.28	0.891	0	0
6	150	300	0.008	3.5	110	0	0
7	50	63.75	0	5.439	21	0	0
8	100	500	0.0075	6	88	50	0.52
9	200	600	0.0085	6	55	0	0
10	15	40	0.009	5.2	90	0	0
11	50	150	0.0045	1.6	65	0	0
12	25	75	0.0025	0.85	78	58	0.02
13	50	63.75	0	2.55	49	0	0
14	0	95	0.0045	1.6	85	0	0
15	20	220	0.0065	4.7	80	92	0.75
16	15	80	0.0045	1.4	90	0	0
17	15	80	0.0025	0.85	10	0	0
18	50	230	0.0045	1.6	25	0	0
19	400	500	0.008	5.5	90	0	0

Demanda: $P_D = 2908\text{MW}$.

TABLE VIII
DADOS DO SISTEMA DE 40 GERADORES

Unidade	$P_{G,k}^{\min}$ [MW]	$P_{G,k}^{\max}$ [MW]	a_k \$/[MW]^2	b_k \$/[MW]	c_k \$	e_k \$	f_k 1/[MW]
1	36	114	0.00690	6.73	94.705	100	0.084
2	36	114	0.00690	6.73	94.705	100	0.084
3	60	120	0.02028	7.07	309.540	100	0.084
4	80	190	0.00942	8.18	369.030	150	0.063
5	47	97	0.01140	5.35	148.890	120	0.077
6	68	140	0.01142	8.05	222.330	100	0.084
7	110	300	0.00357	8.03	287.710	200	0.042
8	135	300	0.00492	6.99	391.980	200	0.042
9	135	300	0.00573	6.60	455.760	200	0.042
10	130	300	0.00605	12.9	722.820	200	0.042
11	94	375	0.00515	12.9	635.200	200	0.042
12	94	375	0.00569	12.8	654.690	200	0.042
13	125	500	0.00421	12.5	913.400	300	0.035
14	125	500	0.00752	8.84	1760.400	300	0.035
15	125	500	0.00752	8.84	1760.400	300	0.035
16	125	500	0.00752	8.84	1760.400	300	0.035
17	220	500	0.00313	7.97	647.850	300	0.035
18	220	500	0.00313	7.95	649.690	300	0.035
19	242	550	0.00313	7.97	647.830	300	0.035
20	242	550	0.00313	7.97	647.810	300	0.035
21	254	550	0.00298	6.63	785.960	300	0.035
22	254	550	0.00298	6.63	785.960	300	0.035
23	254	550	0.00284	6.66	794.530	300	0.035
24	254	550	0.00284	6.66	794.530	300	0.035
25	254	550	0.00277	7.10	801.320	300	0.035
26	254	550	0.00277	7.10	801.320	300	0.035
27	10	150	0.52124	3.33	1055.100	120	0.077
28	10	150	0.52124	3.33	1055.100	120	0.077
29	10	150	0.52124	3.33	1055.100	120	0.077
30	47	97	0.01140	5.35	148.890	120	0.077
31	60	190	0.00160	6.43	222.920	150	0.063
32	60	190	0.00160	6.43	222.920	150	0.063
33	60	190	0.00160	6.43	222.920	150	0.063
34	90	200	0.00010	8.95	107.870	200	0.042
35	90	200	0.00010	8.62	116.580	200	0.042
36	90	200	0.00010	8.62	116.580	200	0.042
37	25	110	0.01610	5.88	307.450	80	0.098
38	25	110	0.01610	5.88	307.450	80	0.098
39	25	110	0.01610	5.88	307.450	80	0.098
40	242	550	0.00313	7.97	647.830	300	0.035

Demanda: $P_D = 10500\text{MW}$.

REFERÊNCIAS

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery e Data Mining, pp. 2623-2631, July 2019.

[2] Bergstra, J., Bardenet, R., Bengio, Y., e Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems, 24*.

[3] P.-H. Chen and H.-C. Chang, "Large-scale economic dispatch by genetic algorithm," IEEE Transactions on Power Systems, vol. 10, no. 4, pp. 1919-1926, 1995.

[4] Y.-H. Hou et al., "Generalized ant colony optimization for economic dispatch of power systems," in *Proceedings. International Conference on Power System Technology*, IEEE, 2002, pp. 225-229.

[5] A. Mahor, V. Prasad, and S. Rangnekar, "Economic dispatch using particle swarm optimization: A review," Renewable and Sustainable Energy Reviews, vol. 13, no. 8, pp. 2134-2141, 2009.

[6] Optuna. *Optuna: A hyperparameter optimization framework*. Disponível em: https://optuna.readthedocs.io/en/stable/. Acesso em: 16 jul. 2024.

[7] R. Pavan, "Método híbrido de enxame de partículas de aprendizagem abrangente com branch-and-bound e programação quadrática sequencial para resolução de problemas do fluxo de potência ótimo com variáveis discretas," 2023.

[8] P. S. Pravin, J. Z. M. Tan, K. S. Yap, and Z. Wu, "Hyperparameter optimization strategies for machine learning-based stochastic energy efficient scheduling in cyber-physical production systems," Digital Chemical Engineering, vol. 4, p. 100047, 2022.

[9] D. C. Secui, "A new modified artificial bee colony algorithm for the economic dispatch problem," Energy Conversion and Management, vol. 89, pp. 43-62, 2015.

[10] T. Sen and H. D. Mathur, "A new approach to solve Economic Dispatch problem using a Hybrid ACO–ABC–HS optimization algorithm," International Journal of Electrical Power e Energy Systems, vol. 78, pp. 735-744, 2016.

[11] D. N. da Silva, "Método primal-dual previsor-corretor de pontos interiores e exteriores com estratégias de correção de inércia e suavização hiperbólica aplicado ao problema de despacho econômico com ponto de carregamento de válvula e representação da transmissão," 2014.

[12] T. Visutarron and T.-C. Chiang, "Economic dispatch using metaheuristics: Algorithms, problems, and solutions," Applied Soft Computing, vol. 150, p. 110891, 2024.

[13] M. F. Zaman et al., "Evolutionary algorithms for dynamic economic dispatch problems," IEEE Transactions on Power Systems, vol. 31, no. 2, pp. 1486-1495, 2015.