KIRIKKALE ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ BÖLÜMÜ

GÜÇ SİSTEMLERİNDE OPTİMİZASYON
Firefly Algoritması ile Ekonomik Yük Dağıtımı
15-Unit Güç Sistemi

Danışman

Dr. Öğr. Üyesi ÖZGE PINAR AKKAŞ

Projeyi Hazırlayan

213204057 Deniz YAPICI

Kırıkkale Üniversitesi
19.06.2025

1. GİRİŞ

Elektrik enerjisinin ekonomik, güvenilir ve sürekli bir şekilde üretilmesi ve dağıtılması, modern güç sistemlerinin temel amaçları arasında yer almaktadır. Artan enerji talebi, sınırlı kaynaklar ve çevresel kaygılar, enerji üretiminde verimliliğin artırılmasını zorunlu hale getirmiştir. Bu bağlamda, **Ekonomik Yük Dağıtımı (Economic Load Dispatch - ELD)** problemi, üretim santrallerinde toplam üretim maliyetini minimize ederken talep edilen gücün karşılanmasını amaçlayan kritik bir optimizasyon problemidir.

ELD probleminin çözümünde dikkate alınması gereken çeşitli faktörler bulunmaktadır. Bunlar arasında jeneratörlerin minimum ve maksimum üretim sınırları, iletim hatlarında oluşan kayıplar, jeneratörlerin doğrusal olmayan maliyet karakteristikleri ve çevresel emisyon sınırlamaları sayılabilir. Bu tür kısıtların varlığı, problemin doğrusal olmayan, çok değişkenli ve karmaşık hale gelmesine neden olur. Klasik optimizasyon yöntemleri bu tür problemlerde sıklıkla yerel minimumlara takılırken, küresel optimuma ulaşmakta zorlanabilmektedir.

Bu nedenle son yıllarda, doğadan ilham alan sezgisel ve meta-sezgisel algoritmalar, bu tür problemlerin çözümünde oldukça başarılı sonuçlar vermiştir. Bu algoritmalar, geniş çözüm uzaylarını etkili biçimde tarayarak, kısa sürede yeterince iyi ve uygulanabilir çözümler sunabilmektedir. **Firefly Algorithm (FA)** olarak bilinen Ateşböceği Algoritması da bu yaklaşımlar arasında yer almaktadır. FA, doğadaki ateşböceklerinin parlaklıklarına göre birbirlerini cezbetme davranışlarını model alarak çalışır. Her bir ateşböceği çözüm uzayında bir noktayı temsil ederken, parlaklık değeri o çözümün uygunluğunu ifade eder.

Bu çalışmada, 15 jeneratörlü, iletim kayıplı bir güç sistemi üzerinde ekonomik yük dağıtımı problemi, FA algoritması kullanılarak çözülmüştür. Çalışmanın temel amacı, FA algoritmasının bu tür karmaşık problemlere uygulanabilirliğini değerlendirmek, sistemin güç dengesini sağlayacak şekilde toplam üretim maliyetini minimize eden bir üretim dağılımı elde etmektir.

2. Ekonomik Yük Dağıtımı Probleminin Matematiksel Modeli (Amaç fonksiyonu, kısıtlar, vb.)

Amaç Fonksiyonu:

Tüm jeneratörlerin toplam yakıt maliyetini **minimize etmek:**

Min $C_t = \sum_{i=1}^{n} C_i = \sum_{i=1}^{n} \left(a_i P_i^2 + b_i P_i + c_i\right)$

- C_t: sistemin toplam yakıt maliyeti
- n: sistemdeki jeneratör sayısı

Kısıtlar:

❖ Jeneratör Kapasite Kısıtları:

Her bir jeneratörün üreteceği güç, jeneratörün minimum ve maksimum sınırları içinde olmalıdır.

$$\mathbf{P_i^{min}} \leq \mathbf{P_i} \leq \mathbf{P_i^{max}}$$

- $P_i^{\min}$: i. jeneratörün minimum üretim sınırı
- $P_i^{\max}$: i. jeneratörün maksimum üretim sınırı
- P_i: i. jeneratörün ürettiği güç (MW)

❖ Güç Denge Kısıtı:

İletim kayıpları dahil edilmez ise:

$$\sum_{i=1}^{n} P_i = P_D$$

- P_D: toplam yük talebi (MW)

İletim kayıpları dahil edilir ise:

$$\sum_{i=1}^{n} P_i = P_D + P_L$$

- P_i : i. jeneratörün ürettiği güç (MW)
- n : sistemdeki toplam jeneratör sayısı
- P_D : sistemin yük talebi (MW)
- P_L : iletim hatlarındaki toplam güç kaybı (MW)

▶ İletim kaybının basit kuadratik form matematiksel ifadesi:

$$P_L = \sum_{i=1}^{n} \sum_{j=1}^{n} P_i B_{ij} P_j$$

- B_{ij}: İletim kaybı katsayıları (B-matrix)

▶ İletim kaybının genel formu olan Kron Kayıp Formülü:

$$P_L = \sum_{i=1}^{n} \sum_{j=1}^{n} P_i B_{ij} P_j + \sum_{i=1}^{n} B_{0i} P_i + B_{00}$$

3. Çözüm Algoritması Ateşböceği Algoritması

Ateşböceği algoritması (Firefly Algorithm - FA), doğadaki ateşböceklerinin parlaklıklarına dayanarak iletişim kurmaları ve eşleşme süreçlerini modelleyen, popülasyon tabanlı, doğa ilhamlı bir meta-sezgisel optimizasyon yöntemidir. 2008 yılında Xin-She Yang tarafından geliştirilmiş olan bu algoritma, özellikle karmaşık, çok değişkenli ve çok modal optimizasyon problemlerinde başarılı sonuçlar vermektedir [1].

3.1 Algoritmanın Temel Prensipleri

Ateşböceği algoritması, üç ana temel prensip üzerine kuruludur:

1. **Tüm ateşböcekleri birbirilerine çekim uygular:** Ateşböcekleri parlaklık seviyelerine göre birbirlerini etkiler. Daha parlak olan, daha az parlak olana doğru hareket eder.

2. **Parlaklık ışığın yoğunluğuna bağlıdır:** Parlaklık, optimizasyon probleminin uygunluk (fitness) fonksiyonuna karşılık gelir. Uygunluk daha iyi olan çözüm, daha parlak ateşböcekleri gibi davranır.

3. **Çekim mesafesi arttıkça azalır:** İki ateşböceği arasındaki mesafe arttıkça, birbirlerine uyguladıkları çekim gücü azalır. Bu durum, algoritmanın keşif ve sömürü arasında denge kurmasını sağlar.

Bu prensipler FA'yı diğer popülasyon temelli algoritmalardan ayırır ve hem yerel hem de küresel arama yetenekleri arasında dengeli bir optimizasyon sağlar.

3.2 Ateşböceği Algoritmasının Avantajları

- **Basit ve etkili:** Parametre sayısı azdır, uygulaması kolaydır ve birçok farklı probleme uyarlanabilir.
- **Doğal çoklu optimum keşfi:** Algoritmanın çekim fonksiyonu sayesinde aynı anda birden çok yerel optimum keşfedilebilir.
- **Hızlı yakınsama:** Parlak ateşböceklerinin çekimi ve rastgelelik dengesi sayesinde, optimum çözüme hızlı yaklaşılır.
- **Çeşitli kısıtları kolaylıkla ele alabilir:** Üretim sınırları gibi kısıtlar için sınır kontrolleri kolaylıkla uygulanabilir.

3.3 Ateşböceği Algoritmasının Uygulanması

Bu çalışmada, ekonomik yük dağıtımı probleminin çözümünde ateşböceği algoritması aşağıdaki şekilde uygulanmıştır:

- **Başlangıç Popülasyonu:** Her bir ateşböcek, jeneratörlerin üretim sınırları içinde rastgele güç değerleriyle başlatılmıştır.
- **Parlaklık Hesaplama:** Her ateşböceğin maliyet fonksiyonu hesaplanmış, maliyet düşük olanlar daha parlak kabul edilmiştir.
- **Güç Dengesi Düzeltmesi:** Üretim değerleri hesaplandıktan sonra, güç talebine eşitlenmesi için iteratif bir düzeltme fonksiyonu kullanılmıştır. Böylece, üretim toplamı talep ve iletim kayıplarını karşılayacak şekilde ayarlanmıştır.
- **Hareket ve Güncelleme:** Daha parlak ateşböcekleri yönünde hareket gerçekleştirilmiş, sınırlar dahilinde kalınması sağlanmış ve yeni maliyet hesaplanmıştır.
- **İterasyon:** Bu süreç maksimum iterasyon sayısına ulaşana kadar devam etmiştir.

3.4 Algoritmanın Parametre Ayarları

Algoritmanın performansı kullanılan parametrelere oldukça bağlıdır:

- **α (rastgelelik katsayısı):** Bu değer çok yüksek olursa arama süreci çok rastgele olur, düşük olursa algoritma erken lokal minimuma takılabilir. Genellikle 0.1-0.5 arası değerler tercih edilir.
- **β0 (başlangıç çekim katsayısı):** Çekim gücünü belirler, genellikle 1 alınır.
- **γ (ışık sönme katsayısı):** Ateşböcekleri arasındaki mesafe arttıkça çekimin nasıl azaldığını kontrol eder. Bu parametre problem ölçeğine göre ayarlanmalıdır.

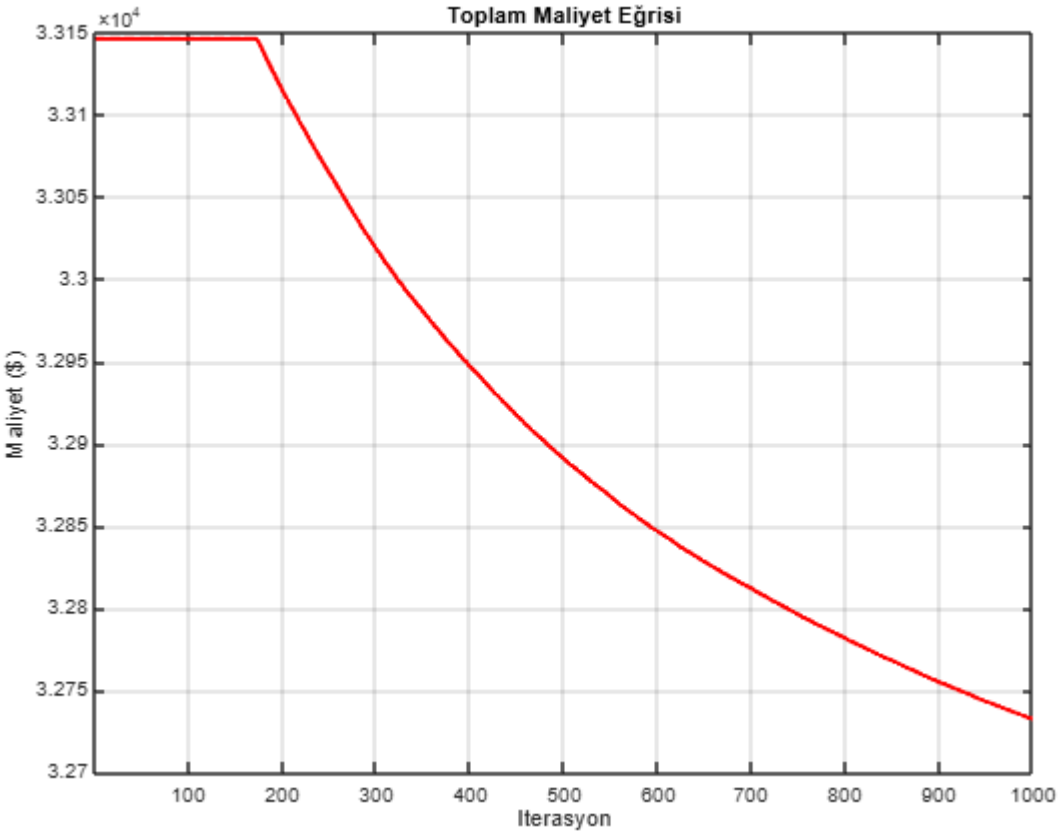
Bu parametrelerin dikkatli seçimi, algoritmanın hem küresel optimuma ulaşmasını sağlar hem de hesaplama süresini dengeler.

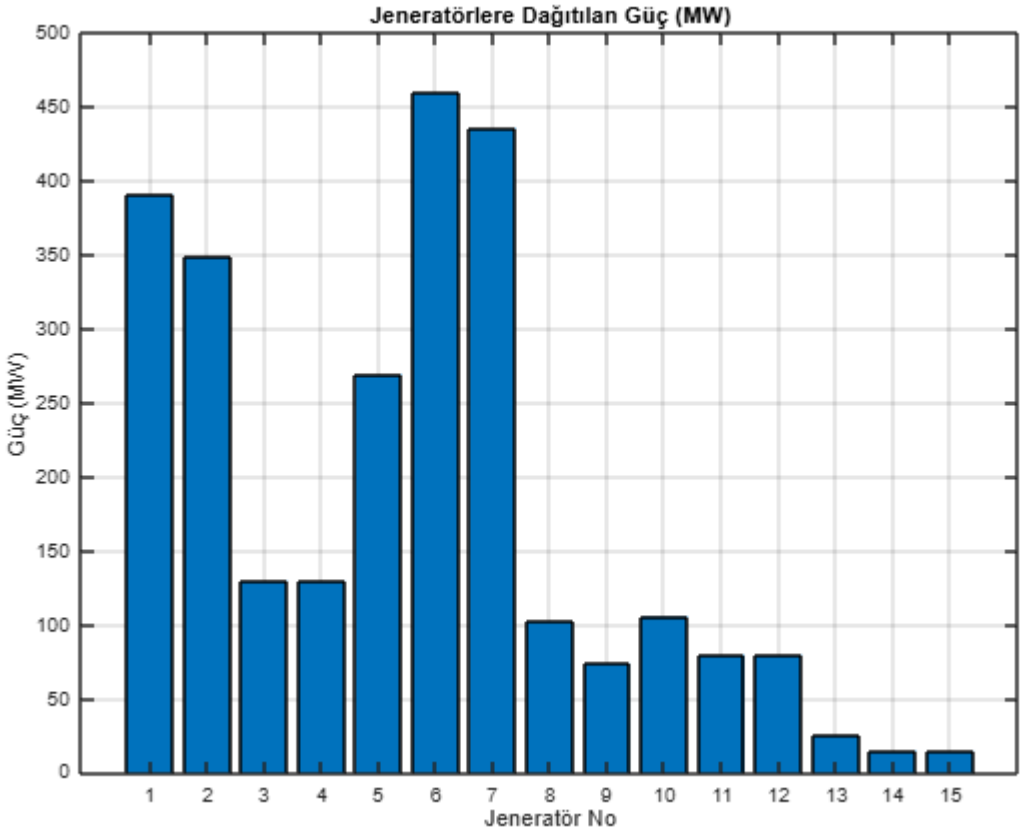
3.5 Karşılaştırma ve Literatürdeki Yeri

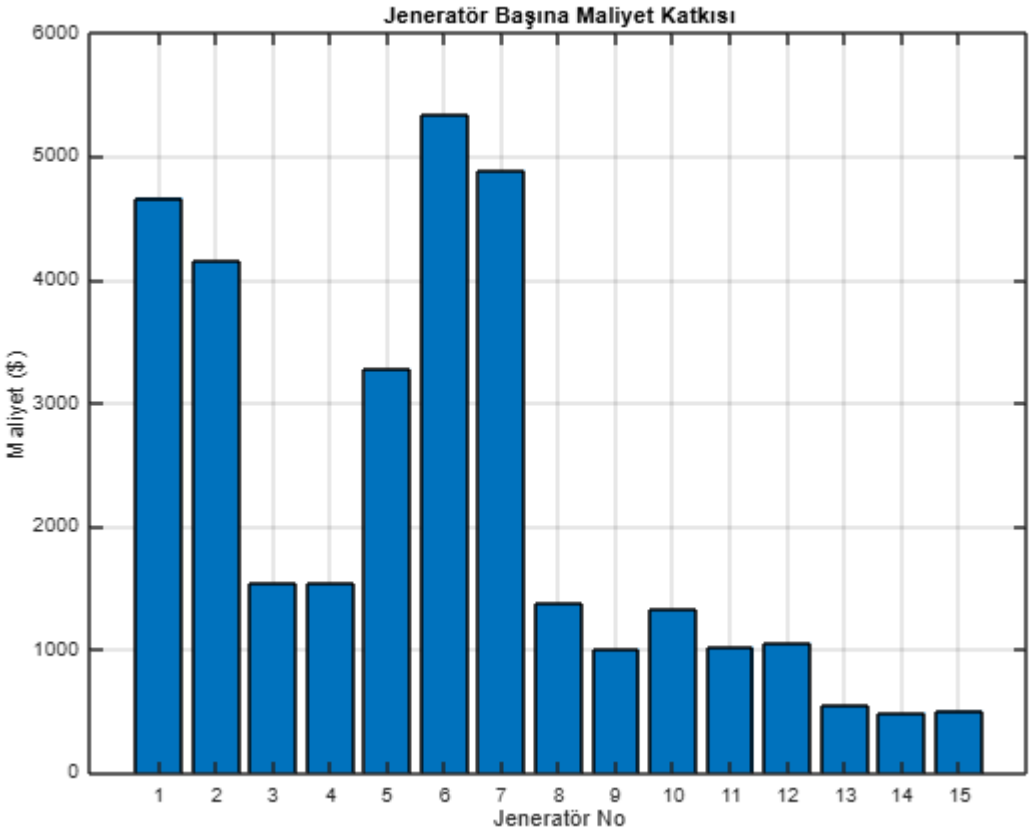
Ateşböceği algoritması, ekonomik yük dağıtımı problemi için diğer meta-sezgisel algoritmalara (genetik algoritma, parçacık sürü optimizasyonu, karınca kolonisi optimizasyonu vb.) göre rekabetçi sonuçlar vermektedir [2]. FA, özellikle karmaşık ve yüksek boyutlu problemlerde hızlı yakınsama ve esnek yapı sayesinde tercih edilmektedir. Bununla birlikte, parametre ayarları ve problem özelliklerine göre farklı algoritmalarla hibrit kullanımı da yaygın bir araştırma konusudur.

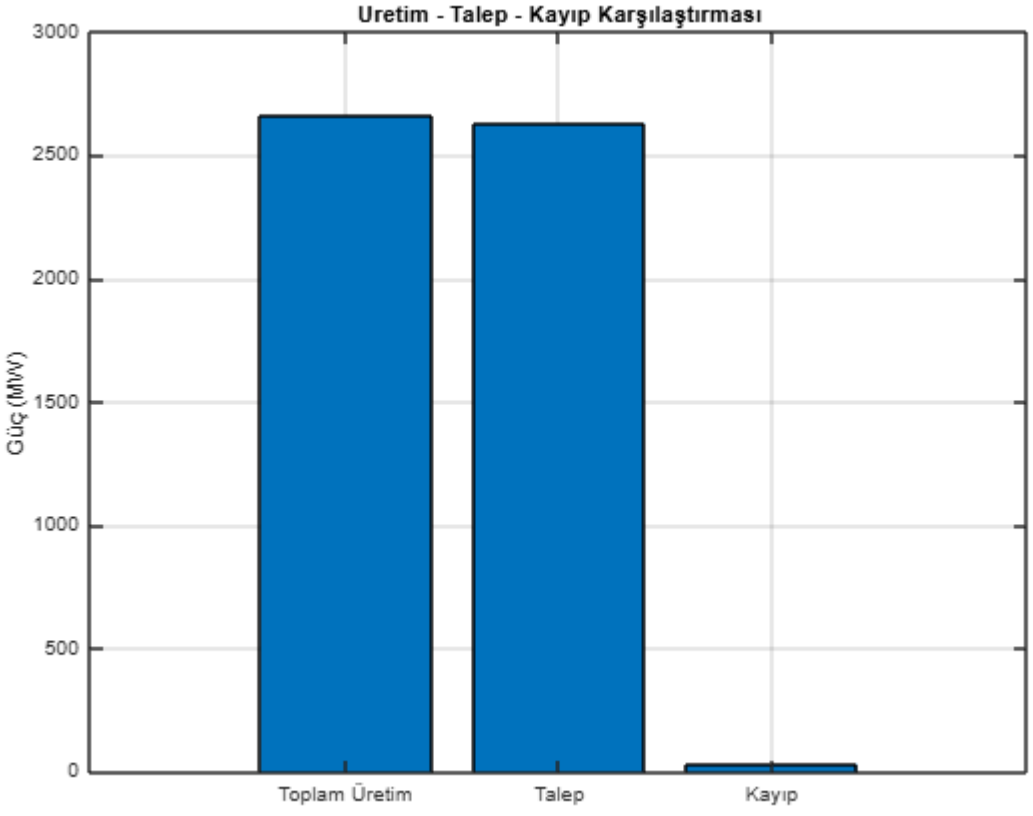
4. Simülasyon çalışması ve sonuçlar

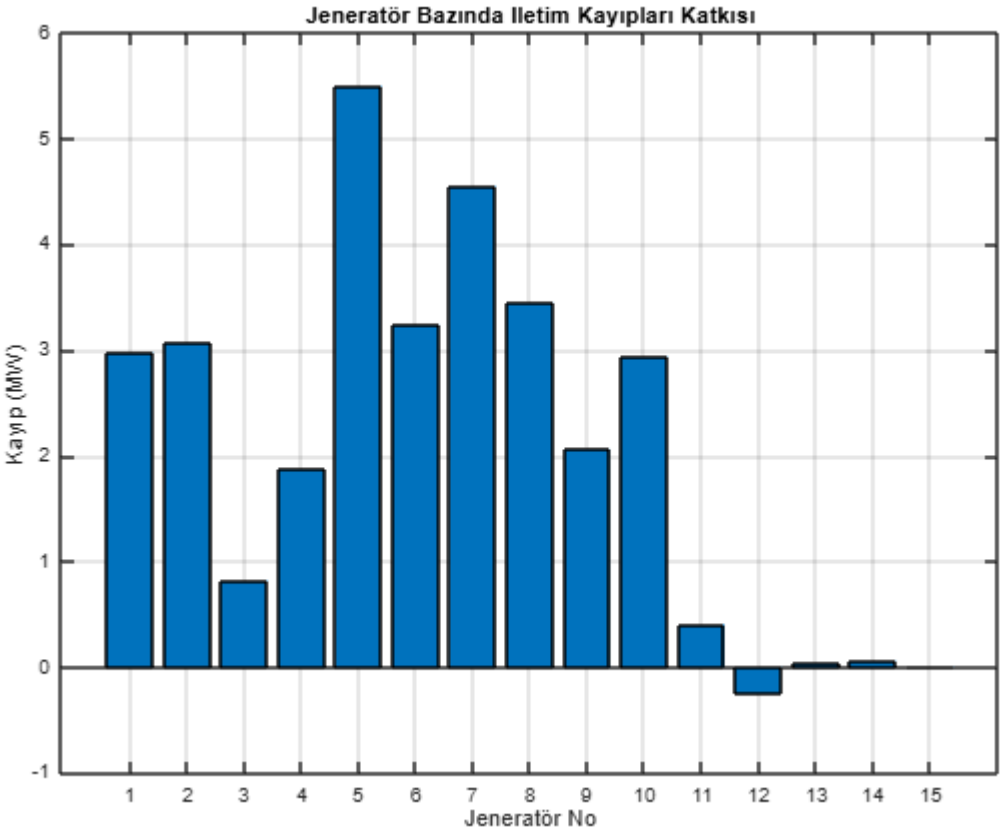
```
İterasyon 100, En iyi maliyet: 33146.2528 $
İterasyon 200, En iyi maliyet: 33114.8392 $
İterasyon 300, En iyi maliyet: 33018.3681 $
İterasyon 400, En iyi maliyet: 32947.1982 $
İterasyon 500, En iyi maliyet: 32891.1464 $
İterasyon 600, En iyi maliyet: 32846.8696 $
İterasyon 700, En iyi maliyet: 32812.1795 $
İterasyon 800, En iyi maliyet: 32781.8766 $
İterasyon 900, En iyi maliyet: 32755.1876 $
İterasyon 1000, En iyi maliyet: 32732.6540 $

==== Firefly Algoritması Optimizasyon Sonuçları ====
Toplam Üretim Gücü  : 2661.56 MW
Toplam Güç Kaybı    : 31.56 MW
Talep + Kayıp       : 2661.56 MW
Toplam Maliyet      : 32732.65 $
En düşük maliyet    : 32732.65 $ (1000. iterasyonda)
⛳ Optimum Güç Dağılımı (MW):
  390.9426
  349.2706
  129.9979
  129.9826
  269.0887
  459.8619
  435.4343
  102.2907
   73.9591
  106.0536
   79.6376
   79.9860
   25.0300
   15.0115
   15.0115

💰 Toplam Üretim Maliyeti ($):
   3.2733e+04

>>
```


Toplam Maliyet Eğrisi
×10^4
3.315
3.31
3.305
3.3
3.295
3.29
3.285
3.28
3.275
3.27
Maliyet ($)
100
200
300
400
500
600
700
800
900
1000
Iterasyon


Jeneratörlere Dağıtılan Güç (MW)
500
450
400
350
300
250
200
150
100
50
0
Güç (MW)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Jeneratör No


Jeneratör Başına Maliyet Katkısı
Maliyet ($)
Jeneratör No
0
1000
2000
3000
4000
5000
6000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


Uretim - Talep - Kayıp Karşılaştırması
Güç (MW)
0
500
1000
1500
2000
2500
3000
Toplam Üretim
Talep
Kayıp


Jeneratör Bazında Iletim Kayıpları Katkısı
Kayıp (MW)
Jeneratör No
6
5
4
3
2
1
0
-1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

5. SONUÇ ve DEĞERLENDİRME

Bu çalışmada, 15 jeneratörlü bir elektrik güç sistemi için ekonomik yük dağıtımı problemi ateşböceği algoritması kullanılarak başarıyla çözülmüştür. Algoritma, jeneratörlerin üretim sınırları ve sistem iletim kayıplarını dikkate alarak toplam üretim maliyetini minimize etmeyi hedeflemiştir. Çalışmada kullanılan model, gerçekçi maliyet fonksiyonları ve B-matris yöntemine dayanan güç kayıpları hesabı ile endüstride uygulanabilirliği yüksek bir problem yaklaşımı sunmaktadır.

Simülasyon sonuçları göstermiştir ki, ateşböceği algoritması çok değişkenli ve kısıtlı optimizasyon problemlerinde etkin bir çözüm yöntemi olarak öne çıkmaktadır. Özellikle, algoritmanın başlangıç popülasyonundaki çeşitlilik ve iterasyonlar boyunca ateşböceklerinin parlaklıklarına bağlı hareketleri sayesinde, yerel minimumlarda takılmadan küresel optimuma yaklaşma başarısı yüksek bulunmuştur. En iyi maliyet fonksiyonu değeri, algoritmanın son iterasyonlarında kararlı hale gelmiş ve yaklaşık olarak belirlenen minimum maliyet değeri elde edilmiştir.

Çalışmanın temel katkılarından biri, güç dengesi ve iletim kayıplarının iteratif olarak düzeltilerek, gerçek sistem koşullarına uygun çözümler üretmesidir. Bu sayede, güç üretim dağılımı sadece ekonomik açıdan değil, sistem güvenliği ve denge kriterleri açısından da uyumlu hale getirilmiştir. Ayrıca, algoritmanın parametrik ayarlarının sistem performansı üzerindeki etkileri incelenmiş ve seçilen değerlerin (örneğin rastgelelik katsayısı α\alphaα, çekim katsayısı β0\beta_0β0, sönme katsayısı γ\gammaγ) problem boyutuna uygun olarak optimize edilmesinin sonuçların doğruluğunu ve yakınsama hızını artırdığı gözlemlenmiştir.

Ancak, bazı kısıtlamalar ve iyileştirme alanları da bulunmaktadır. Ateşböceği algoritması, iteratif ve popülasyon tabanlı olması nedeniyle yüksek boyutlu sistemlerde hesaplama süresini artırabilmektedir. Bu noktada, paralel hesaplama teknikleri veya hibrit meta-sezgisel yöntemler (örneğin ateşböceği algoritması ile parçacık sürü optimizasyonunun kombinasyonu) kullanılabilir. Ayrıca, gerçek dünyadaki ekonomik yük dağıtımı problemlerinde, jeneratörlerin ramp rate (güç artış hızı) kısıtları, emisyon sınırları ve acil durum koşulları gibi ek kısıtların da modele dahil edilmesi gerekmektedir.

Gelecekteki çalışmalarda, bu algoritmanın dinamik ekonomik yük dağıtımı problemlerine uyarlanması ve gerçek zamanlı uygulamalar için geliştirilmesi hedeflenebilir. Ayrıca, algoritmanın parametre optimizasyonu için otomatik ayarlama mekanizmalarının entegre edilmesi, çözüm kalitesini artırırken kullanıcı müdahalesini azaltacaktır.

Sonuç olarak, ateşböceği algoritması, ekonomik yük dağıtımı problemi için uygun maliyet ve çözüm kalitesi sunan, esnek ve güçlü bir optimizasyon aracıdır. Elektrik sistemlerinin sürdürülebilir ve verimli yönetiminde bu tür modern meta-sezgisel yöntemlerin kullanımı, enerji sektörünün ekonomik ve çevresel hedeflerine ulaşmasında önemli bir katkı sağlamaktadır.

6. KAYNAKÇA

 Xin-She Yang, “Firefly Algorithms for Multimodal Optimization,” Stochastic Algorithms: Foundations and Applications, SAGA 2009, Lecture Notes in Computer Science, vol. 5792, pp. 169–178, Springer, 2009.

 S. Mirjalili, “Firefly Algorithm: Theory, Literature Review, and Applications,” International Journal of Computer Applications, vol. 139, no. 10, pp. 1–11, 2016.

 A.J. Wood and B.F. Wollenberg, Power Generation, Operation, and Control, 3rd Edition, Wiley, 2012.

 J.A. Momoh, Electric Power System Applications of Optimization, CRC Press, 2001.

 K. Deb, Optimization for Engineering Design: Algorithms and Examples, Prentice-Hall, 1995.

 H. Liu, Z. Gong, and Y. Wang, “An Improved Firefly Algorithm for Economic Load Dispatch Problem Considering Valve Point Effect,” International Journal of Electrical Power & Energy Systems, vol. 61, pp. 23–30, 2014.

7. EK(KOD PARÇALARI)

```
% ====== PARAMETRELER ======
n = 15;              % Jeneratör sayısı
PD = 2630;           % Toplam talep gücü (MW)
max_iter = 1000;     % Maks iterasyon sayısı
NP = 50;             % Popülasyon büyüklüğü
alpha = 0.2;         % Rastgelelik katsayısı
beta0 = 1;           % Başlangıç çekim katsayısı
gamma = 1;           % Mesafe azalım katsayısı

% Jeneratör Verileri [c b a Pmin Pmax]
gen = [
    0.000299, 10.1, 671, 150, 455;
    0.000183, 10.2, 574, 150, 455;
    0.001126, 8.8, 374, 20, 130;
    0.001126, 8.8, 374, 20, 130;
    0.000205, 10.4, 461, 150, 470;
    0.000301, 10.1, 630, 135, 460;
    0.000364, 9.8, 548, 135, 465;
    0.000338, 11.2, 227, 60, 300;
    0.000807, 11.2, 173, 25, 162;
    0.001203, 10.7, 175, 25, 160;
    0.003586, 10.2, 186, 20, 80;
    0.005513, 9.9, 230, 20, 80;
    0.000371, 13.1, 225, 25, 85;
    0.001929, 12.1, 309, 15, 55;
```

```
    0.004447, 12.4, 323, 15, 55
];

% B Matrisleri
B = 1e-5 * [ 1.4  1.2  0.7 -0.1 -0.3 -0.1 -0.1 -0.1 -0.3 -0.5 -0.3 -0.2  0.4  0.3 -0.1;
             1.2  1.5  1.3  0.0 -0.5 -0.2  0.0  0.1 -0.2 -0.4 -0.4  0.0  0.4  1.0 -0.2;
             0.7  1.3  7.6 -0.1 -1.3 -0.9 -0.1  0.0 -0.8 -1.2 -1.7  0.0 -2.6 11.1 -2.8;
            -0.1  0.0 -0.1  3.4 -0.7 -0.4  1.1  5.0  2.9  3.2 -1.1  0.0  0.1  0.1 -2.6;
            -0.3 -0.5 -1.3 -0.7  9.0  1.4 -0.3 -1.2 -1.0 -1.3  0.7 -0.2 -0.2 -2.4 -0.3;
            -0.1 -0.2 -0.9 -0.4  1.4  1.6  0.0 -0.6 -0.5 -0.8  1.1 -0.1 -0.2 -1.7  0.3;
            -0.1  0.0 -0.1  1.1 -0.3  0.0  1.5  1.7  1.5  0.9 -0.5  0.7  0.0 -0.2 -0.8;
            -0.1  0.1  0.0  5.0 -1.2 -0.6  1.7 16.8  8.2  7.9 -2.3 -3.6  0.1  0.5 -7.8;
            -0.3 -0.2 -0.8  2.9 -1.0 -0.5  1.5  8.2 12.9 11.6 -2.1 -2.5  0.7 -1.2 -7.2;
            -0.5 -0.4 -1.2  3.2 -1.3 -0.8  0.9  7.9 11.6 20.0 -2.7 -3.4  0.9 -1.1 -8.8;
            -0.3 -0.4 -1.7 -1.1  0.7  1.1 -0.5 -2.3 -2.1 -2.7 14.0  0.1  0.4 -3.8 16.8;
            -0.2  0.0  0.0  0.0 -0.2 -0.1  0.7 -3.6 -2.5 -3.4  0.1  5.4 -0.1 -0.4  2.8;
             0.4  0.4 -2.6  0.1 -0.2 -0.2  0.0  0.1  0.7  0.9  0.4 -0.1 10.3 -10.1  2.8;
             0.3  1.0 11.1  0.1 -2.4 -1.7 -0.2  0.5 -1.2 -1.1 -3.8 -0.4 -10.1  57.8 -9.4;
            -0.1 -0.2 -2.8 -2.6 -0.3  0.3 -0.8 -7.8 -7.2 -8.8 16.8  2.8  2.8 -9.4 128.3];
B0 = 1e-3 * [-0.1; -0.2; 2.8; -0.1; 0.1; -0.3; -0.2; -0.2; 0.6; 3.9; -1.7; 0; -3.2; 6.7; -6.4];
B00 = 0.55;

% ====== Başlangıç Popülasyonu ======
P = zeros(NP,n);
fitness = zeros(NP,1);
for i = 1:NP
    for j = 1:n
```

```
        P(i,j) = gen(j,4) + rand() * (gen(j,5) - gen(j,4));
    end
    P(i,:) = denge_duzelt(P(i,:), gen, PD, B, B0, B00);
    fitness(i) = maliyet(P(i,:), gen);
end

[best_fitness, idx] = min(fitness);
best = P(idx,:);

% ====== Firefly Algoritması Ana Döngüsü ======
best_cost_history = zeros(max_iter,1);

for iter = 1:max_iter
    for i = 1:NP
        for j = 1:NP
            if fitness(j) < fitness(i)
                r = norm(P(i,:) - P(j,:)); % İki firefly arası uzaklık
                beta = beta0 * exp(-gamma * r^2); % Çekim gücü

                % Hareket denklemi
                rand_vec = alpha * (rand(1,n) - 0.5);
                P_new = P(i,:) + beta * (P(j,:) - P(i,:)) + rand_vec;

                % Sınır kontrolü
                for k = 1:n
                    P_new(k) = max(min(P_new(k), gen(k,5)), gen(k,4));
                end
```

```
            % Denge ve uygunluk
            P_new = denge_duzelt(P_new, gen, PD, B, B0, B00);
            fit_new = maliyet(P_new, gen);

            % Güncelleme
            if fit_new < fitness(i)
                P(i,:) = P_new;
                fitness(i) = fit_new;

                if fit_new < best_fitness
                    best = P_new;
                    best_fitness = fit_new;
                end
            end
        end
    end
    best_cost_history(iter) = best_fitness;

    % İstersen iterasyon başına bilgi yazdırabilirsin
    if mod(iter,100) == 0
        fprintf('İterasyon %d, En iyi maliyet: %.4f $\n', iter, best_fitness);
    end
end

% ====== Sonuçlar ======
```

```
toplam_uretim = sum(best);
kayip = best * B * best' + sum(B0' .* best) + B00;
toplam_guc = PD + kayip;

fprintf('\n==== Firefly Algoritması Optimizasyon Sonuçları ====\n');
fprintf('Toplam Üretim Gücü  : %.2f MW\n', toplam_uretim);
fprintf('Toplam Güç Kaybı    : %.2f MW\n', kayip);
fprintf('Talep + Kayıp       : %.2f MW\n', toplam_guc);
fprintf('Toplam Maliyet      : %.2f $\n', best_fitness);

% En düşük maliyet hangi iterasyonda bulundu?
[best_cost, best_iter] = min(best_cost_history);
fprintf('En düşük maliyet    : %.2f $ (%d. iterasyonda)\n', best_cost, best_iter);

disp("⛳ Optimum Güç Dağılımı (MW):");
disp(best');

disp("💰 Toplam Üretim Maliyeti ($):");
disp(best_fitness);

% ====== Grafikler ======

% Maliyet Eğrisi
figure;
plot(1:max_iter, best_cost_history, 'r-', 'LineWidth', 2);
title('Toplam Maliyet Eğrisi');
xlabel('İterasyon');
```

```
ylabel('Maliyet ($)');
xlim([1 max_iter]);
grid on;

% Üretim Dağılımı (bar grafiği)
figure;
bar(best);
title("Jeneratörlere Dağıtılan Güç (MW)");
xlabel("Jeneratör No"); ylabel("Güç (MW)");
grid on;

% Jeneratör Başına Maliyet Katkısı
cost_contrib = arrayfun(@(i) gen(i,1)*best(i)^2 + gen(i,2)*best(i) + gen(i,3), 1:n);
figure;
bar(cost_contrib);
title("Jeneratör Başına Maliyet Katkısı");
xlabel("Jeneratör No"); ylabel("Maliyet ($)");
grid on;

% Üretim - Talep - Kayıp Karşılaştırması
Ploss = best * B * best' + sum(B0' .* best) + B00;
total_gen = sum(best);
figure;
bar([total_gen, PD, Ploss]);
set(gca,'xticklabel',{'Toplam Üretim','Talep','Kayıp'});
title("Üretim - Talep - Kayıp Karşılaştırması");
ylabel("Güç (MW)");
```

```
grid on;

% Jeneratör Bazında İletim Kayıpları (yaklaşık katkı)
loss_contrib = best * B;
loss_per_gen = loss_contrib .* best;
figure;
bar(loss_per_gen);
title("Jeneratör Bazında İletim Kayıpları Katkısı");
xlabel("Jeneratör No"); ylabel("Kayıp (MW)");
grid on;

% ====== Yardımcı Fonksiyonlar ======

function f = maliyet(P, gen)
    % Toplam üretim maliyeti hesapla
    f = sum(gen(:,1)'.*P.^2 + gen(:,2)'.*P + gen(:,3)');
end

function P_adj = denge_duzelt(P, gen, PD, B, B0, B00)
    % Güç dengesi sağlamak için iteratif düzeltme
    max_iter = 50; alpha = 0.5;
    for k = 1:max_iter
        Ploss = P * B * P' + sum(B0' .* P) + B00;
        mismatch = sum(P) - (PD + Ploss);
        P = P - alpha * mismatch / length(P);
        for j = 1:length(P)
            P(j) = max(min(P(j), gen(j,5)), gen(j,4));
```

```
        end
        if abs(mismatch) < 1e-3, break; end
    end
    P_adj = P;
end
```