

# Optimizasyon Kavramı

Optimizasyon Nedir ?

Optimizasyon ya da başka bir deyişle 'en iyileme' kavramı, genel olarak bir problemin çözümünde en uygun parametrelerin tespit edilmesi olarak bilinmektedir.

Matematiksel anlamda optimizasyon, denklemlerle modellenmesi yapılmış bir problem için bilinmeyen değerlerin bulunmasına karşılık gelmektedir.



**Örneğin**, y ile gösterilen yazılı sınavı puanı ve s ile gösterilen sözlü sınavı puanı dikkate alınmak üzere, y'nin %60'ı ve s'nin %40'ının alınması hesaplanacak ders başarı puanı DB şu şekilde gösterilebilmektedir:

$$DB(y, s) = (0,6 * y) + (0,4 * s)$$

DB fonksiyonunda bir dersten başarılı olmak için başarı puanının en az 60 olması gerektiğini kabul edersek, yazılıdan (y) 50 alan bir öğrencinin, sözlüden (s) en az kaç alması gerektiğini bulmak, optimizasyona tekabül etmektedir.

Burada denklemdeki 0,6 ve 0,4 katsayı olarak bilinmekte, y ve s ise değişken olarak ifade edilmektedir. Eğer fonksiyona +10 puan kanaat puanı eklenecek olursa bu değer de sabit olarak isimlendirilmektedir.

Eğitmen öğrencilere, 50 yazılı puanı karşısında, sözlüden en az 75 puan alınırsa başarılı olur.

Linner Optimizasyondan.

Bir oyuncak firması, pilli ve pilsiz oyuncak arabalar üretmektedir. Firma oyuncak arabaları üretirken M1, M2 ve M3 makinelerini kullanmaktadır. Pilsiz oyuncak araba üretilirken bir günde, M1' in 2 saat, M2'nin 1 saat ve M3'ün 1 saat çalışması gerekmektedir. Pilli oyuncak araba üretilirken ise bir günde, M1 makinesinin 1 saat, M2'nin 2 saat ve M3'ün 1 saat çalışması gerekmektedir. M1, M2 ve M3'lerin aylık çalışma saatleri en fazla 180, 160 ve 100 dür. Firma üretilen bir adet pilsiz oyuncak arabadan 40 TL, bir adet pilli oyuncak arabadan ise 60 TL kar etmektedir. Oyuncak firması günlük karını en büyük yapmak için her bir oyuncak arabadan kaç tane üretmelidir? Buna göre problemi, d.p.p. biçiminde modelleyiniz.

**Çözüm:**

Çalışma süreleri	M1	M2	M3	Birim kar
Pilsiz oyuncak için günlük süre (sa)	2	1	1	40
Pilli oyuncak için günlük süre (sa)	1	2	1	60
Aylık toplam çalışma süresi (sa)	180	160	100	

$X_1$  : Bir günde üretilen pilsiz oyuncak araba sayısı (adet)

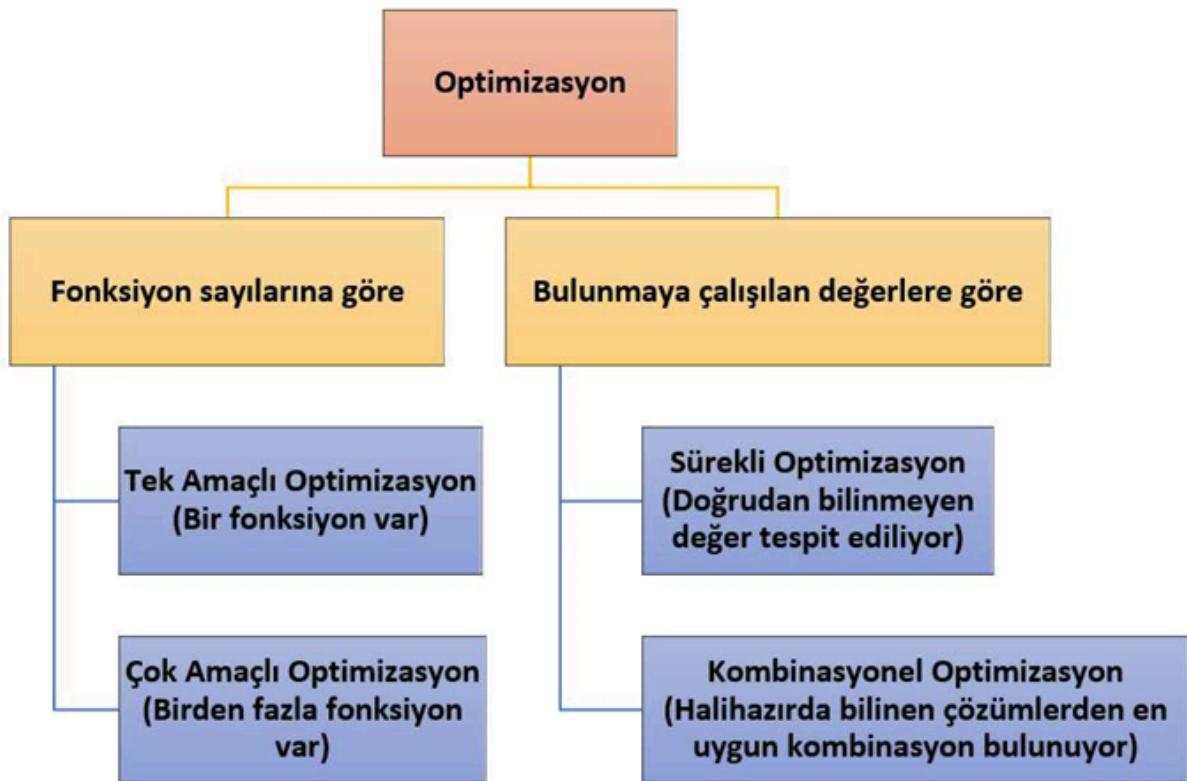
$X_2$  : Bir günde üretilen pilli oyuncak araba sayısı (adet)

$$\begin{aligned}
 P: \max Z &= 40X_1 + 60X_2 \\
 2X_1 + X_2 &\leq (180 / 30) \\
 X_1 + 2X_2 &\leq (160 / 30) \\
 X_1 + X_2 &\leq (100 / 30) \\
 X_1, X_2 &\geq 0
 \end{aligned}$$

Özellikle mühendislik tabanlı optimizasyon problemleri daha yüksek dereceli olabilmekte (yani değişkenler daha yüksek üstel sayıarda temsil edilmekte), aynı anda çok daha fazla değerin bulunmasını gereklili kılabilmekte ve daha karmaşık fonksiyonlara tekabül edebilmektedir. Böyle durumlarda türev gibi daha ileri düzey matematiksel çözümler kullanılmaktadır.

Problem modellenirken karşılaşılan olaylar:

- Aynı anda birbirleriyle ilişkili birden fazla fonksiyonun dikkate alınması gerekebilmektedir.
- Değeri bulunması gereklili olan değişkenlerin sınır değerleri
- Bazı optimizasyon problemleri doğrudan bilinmeyen değerlerin bulunmasına odaklanırken (sürekli optimizasyon) bazıları bilinen çözümlerin hangi kombinasyonlarının daha iyi olacağını bulmak için (kombinasyonel optimizasyon) (orn gezgin adam problemi) modellenmektedir.



### Kombinatöryal (Ayrık) Optimizasyon:

- Karar değişkenleri **ayırık (discrete)** yapıdadır.
- Çözüm uzayı **sonlu ama genellikle çok büyük**tür.
- Amaç: En iyi kombinasyonu bulmak.

TSP Bu Kapsamda Neden Kombinatöryaldır?

- $n$  şehir var.
- Amaç: Her şehri **bir kez ziyaret edip başlangıç noktasına dönmek**.
- Karar: **Hangi sırayla gezilecek?**

Toplam olası tur sayısı:

$$(n-1)!/2$$

Yani çözüm uzayı **permütasyonlardan oluşur**.

Burada:

- Değişkenler: "Şehir sırası"
- Değerler: Ayrık (1., 2., 3. şehir gibi)
- Çözüm: Bir permutasyon

Bu yüzden TSP klasik bir **kombinatöryal optimizasyon problemidir**.

## Sürekli (Continuous) Optimizasyon

- Karar değişkenleri **gerçek sayılar** ( $\mathbb{R}$ ) kümesindedir.
- Çözüm uzayı **sonsuzdur**.
- Türev, gradyan, Hessian gibi matematiksel araçlar kullanılır.

Örnek:

$$\min f(x) = x^2 + 3x + 2$$

Burada:

- $x \in \mathbb{R}$
- Çözüm türev alarak bulunur
- Problem yapısı çözüm yöntemini belirler.
- Eğer değişkenler ayrık ise  $\rightarrow$  Kombinatöryal teknikler
- Eğer değişkenler sürekli ise  $\rightarrow$  Analitik / türev tabanlı tekn

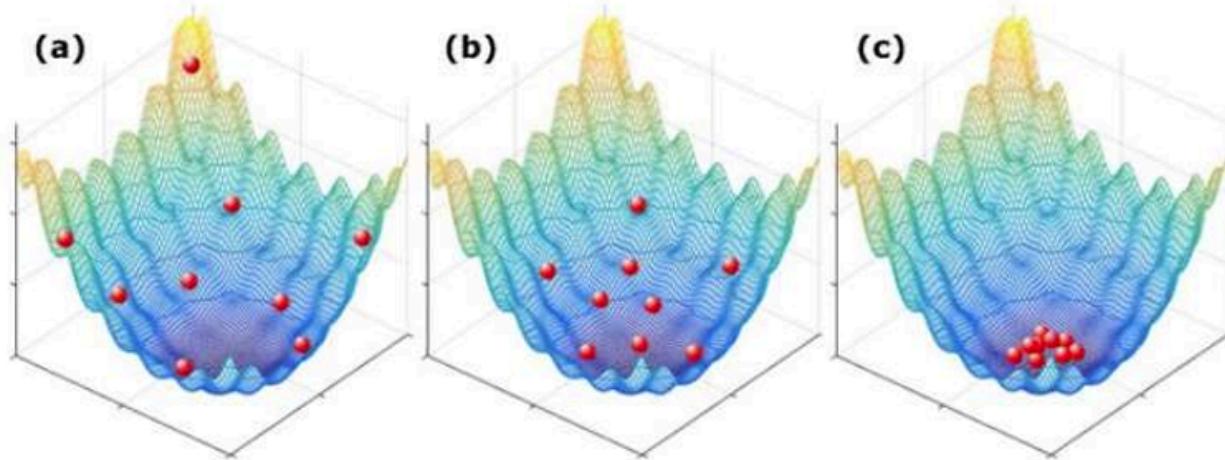
## Zeki Optimizasyon Kavramı

Matematikte bilinen klasik optimizasyon yöntemleri, karmaşıklığı yüksek ve oldukça fazla değişken içeren problem modellerinde başarılı sonuç verememektedir. Bunun neticesinde temellerini doğadaki canlı sürülerinin iş birliği içerisindeki, şans faktörü içeren eylemlerinden alan yapay zekâ tabanlı optimizasyon algoritmaları geliştirilmiştir.

Zeki optimizasyon, makine öğrenmesi kapsamında olduğu gibi öğrenme süreci gerektirmemektedir. Daha çok döngüler yardımıyla en uygun değerlerin tespit edilmeye çalışılmasına dayanmaktadır. Bunu sağlamak için de algoritmalar tasarılanırken şu mekanizmalar işletilmektedir:

- Típkı etmen tabanlı modellemede olduğu gibi N sayıda etmenin tanımı yapılmaktadır. Bu noktada etmen tabanlı modellemeden farklı olarak bu çözüm unsurları sadece sayı ya da kombinasyon aramakla sorumludur. Her etmen algoritma başlangıcında rastgele değerlerle eşlenerek aramaya başlamaktadır. Bu süreç típkı çok sayıda kişinin aynı problemi çözmek için uğraşıp elde ettikleri sonuçlara göre birbirleriyle yardımlaşmaları gibidir.
- Problemle bağlantılı olarak N sayıda etmen rastgele hareketlerle problem modeline (denkleme/fonksiyona) konulacak değerler türetmekte ve her bir etmenin hesapladığı nihai değerlerle en uygun (en küçük/minimum ya da en büyük/maksimum) sonucu veren değerler tespit edilmektedir.
- N adet etmen arasında belli bir aşamada en uygun değeri verene diğerleri sayısal olarak yaklaştırılmaya çalışılmakta, bu süreç belli bir döngü sayısı boyunca işletilmektedir.

- Zeki optimizasyonu sağlamak için farklı matematiksel mekanizmalarla tasarlanan ve doğadaki canlılardan (kuşlar, böcekler vs.), insan topluluklarından ve hatta dinamik olgulardan (mevsimsel değişimler, fizik kanunlarıyla bağlantılı olaylar, çiçek tozlaşması, akarsuların akışı vs.) esinlenen, bilinmeyen değerleri arama yolunda kendi iç farklılıklarını içeren algoritmalar tasarlanabilmektedir. Farklı algoritmalar N adet çözüm unsurlarını tarif etmek için algoritma esin kaynağına göre parçacık, birey, arı, karınca gibi isimler kullanılabilmektedir.

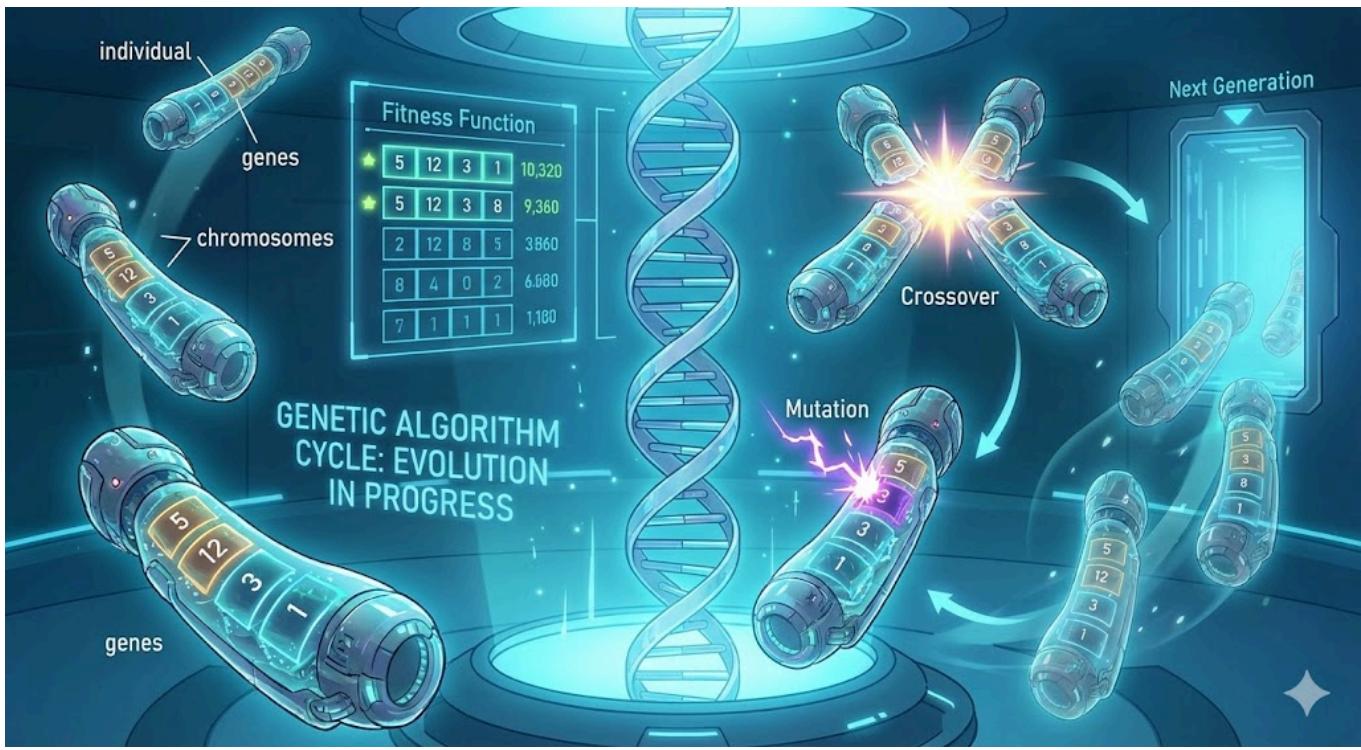


Şekil 7.2. Görsel olarak tipik bir optimizasyon süreci (Web Kaynağı 7.1)

Şekilde görüldüğü gibi, bulunmaya çalışılan en uygun düşük değeri bulmak soldan sağa doğru belli bir süreci gerekliliğe sahiptir. Bu sırada istenilen değere ulaşma sırasında çeşitli engeller ve yanlış tespitlerde bulunmak da olasıdır. Bu nedenle zeki optimizasyon garanti çözüm sunmayan, ancak mümkün olduğunda etkili çözümlerle gerçek hayatı problemlere başarılı sonuçlar üreten algoritmalar içermektedir.

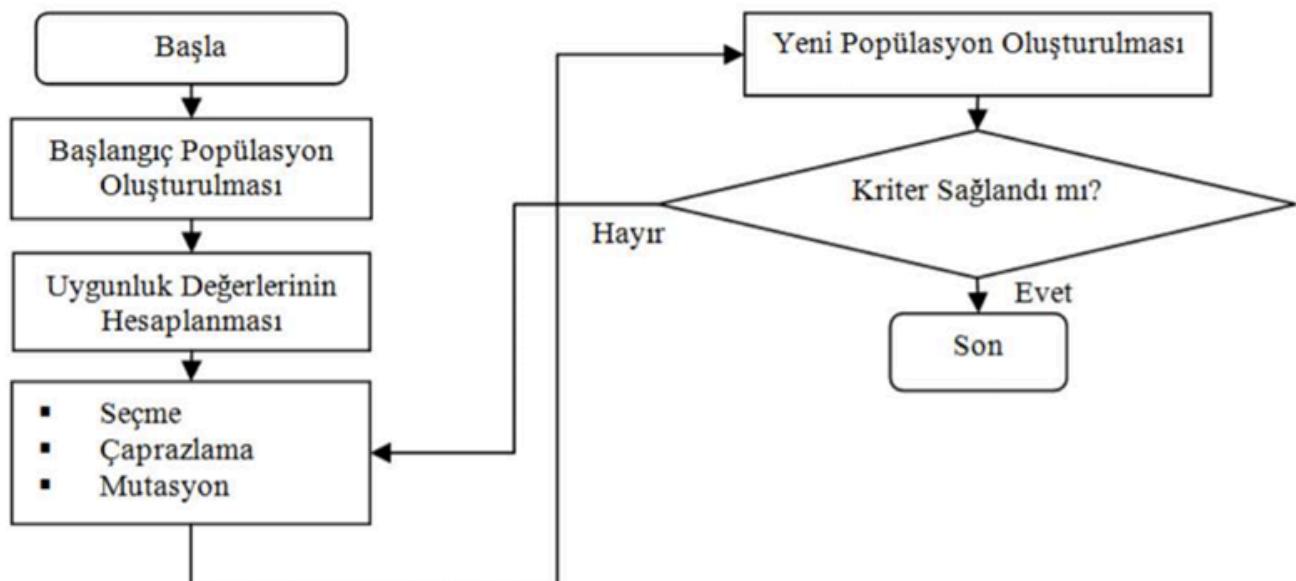
## Genetik Algoritma ile Zeki Optimizasyon

Zeki optimizasyon konusunda yüzlerce algoritma bulunmaktadır. Ancak bunlardan bazıları konuya anlamak ve başlangıç için daha idealdir. Genetik Algoritma da yapısı itibariyle buna en uygun algoritmaların biridir. Genetik Algoritma, güçlü ve başarılı olan çözüm unsurlarının özelliklerinin yeni çözüm unsurlarına aktarılırarak başarı şansının artırıldığı bir zeki optimizasyon algoritmasıdır. Buna göre mevcut popülasyonlardan (çözüm unsurlarından) problem fonksiyonu için nispeten daha iyi değer üretenler birbirleriyle eşleştirilerek daha başarılı yeni nesil bireylerden popülasyonlar elde edilmektedir. Hedeflenen döngü adımları boyunca nesillerin aynı şekilde türetilmesine devam edilmektedir. Genetik Algoritma orijinal yapısı itibariyle bir sürekli optimizasyon algoritmasıdır.

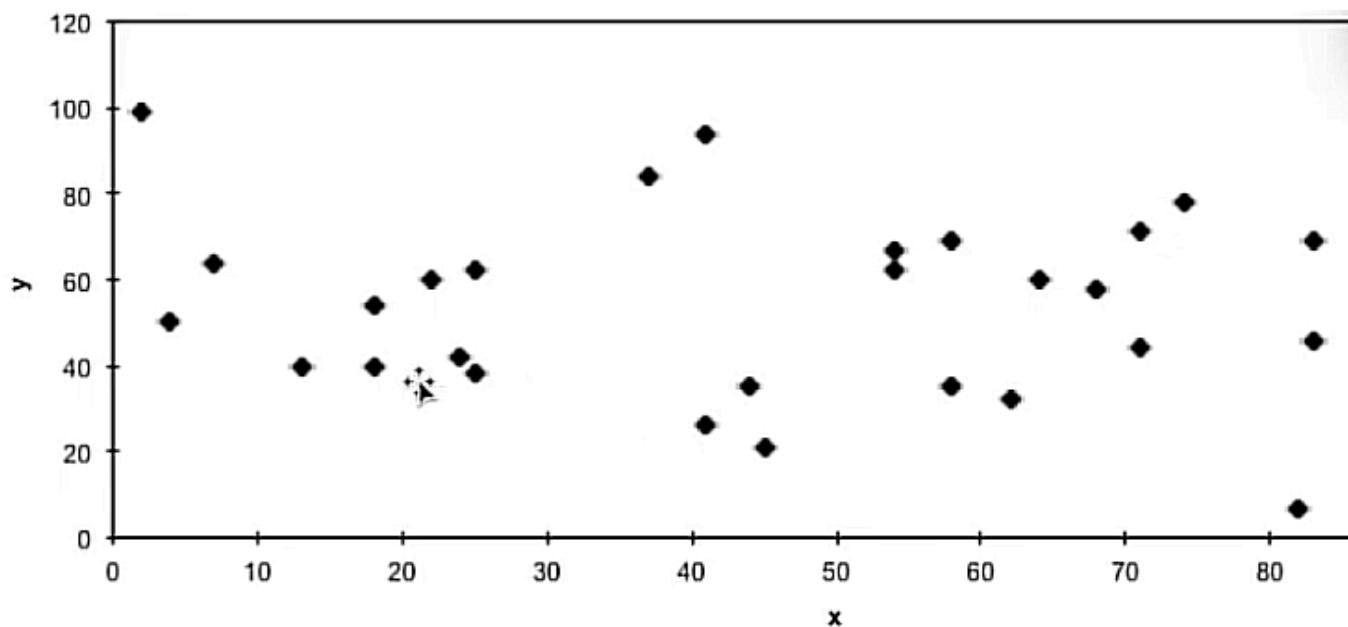
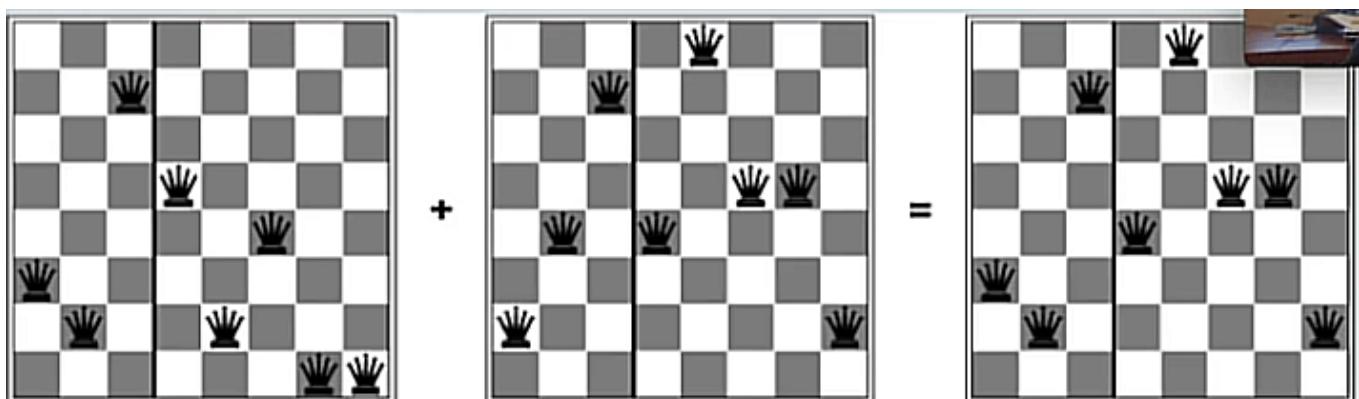


Genetik Algoritma'da bireylerin her biri problem fonksiyona ait aranılan değerleri tutmak adına gen adı verilen bileşenlerle tanımlanmaktadır. Genel olarak kodlama adı verilen bu süreç ile örneğin beş bilinmeyen değişkeni olan bir problem için N tane bireyin her birinde beşer adet değer/gen tutulmaktadır. Buradan hareketle her birey bir tür kromozom ile temsil edilir.

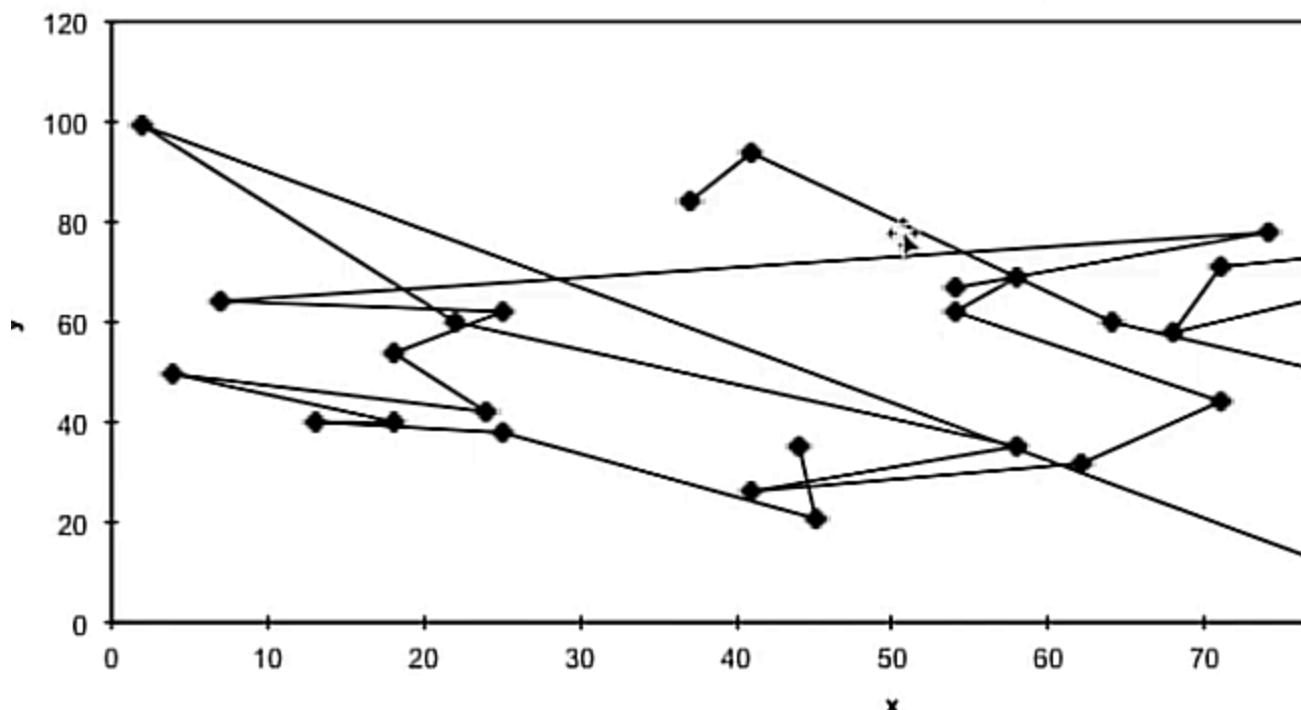
Genetik Algoritma'nın her yeni döngüsünde, ilgili bireylerin taşıdığı değerler (genler) problem fonksiyonuna konularak fonksiyon sonuçları (uygunluk fonksiyonu değerleri) üretilmekte, belli sayıdaki en iyi değerleri bulan bireyler arası gen değişimleri (çaprazlama/crossover) ve belli genlerin de yeni değerlerle değiştirilmesi (mutasyon) suretiyle eski popülasyon yerine yeni nesil popülasyon (ebeveynlerin yerine çocuklar) elde edilerek bir sonraki döngüye geçilmektedir. Genetik Algoritma başarılı olan çözümleri gelecek süreçlere taşıyabilmesi nedeniyle Evrimsel Algoritmalar arasında kabul edilmektedir. Şekil 7.3 bu çerçevede Genetik Algoritma adımlarını kısaca görselleştirmektedir.



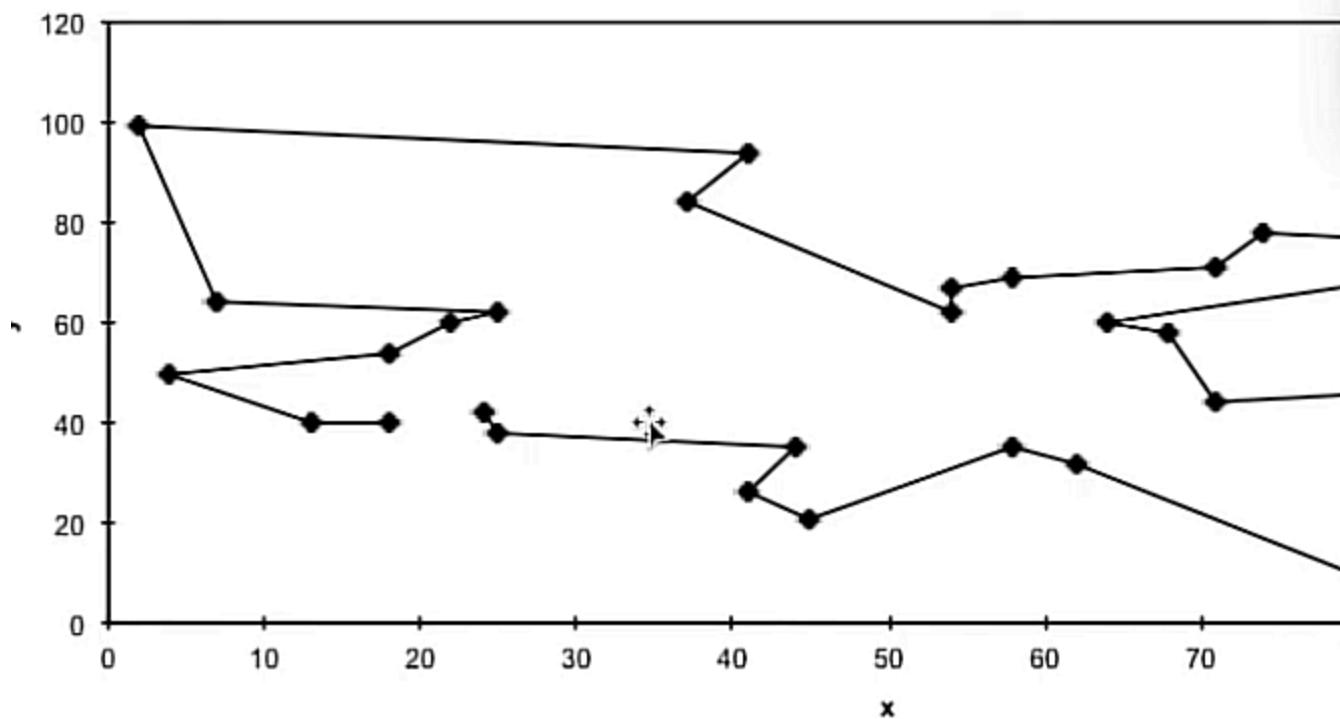
Şekil 7.3. Genetik Algoritma'nın genel akışı (Web Kaynağı 7.2)



### TSP30 (Performance = 800)



### TSP30 Solution (Performance = 420)



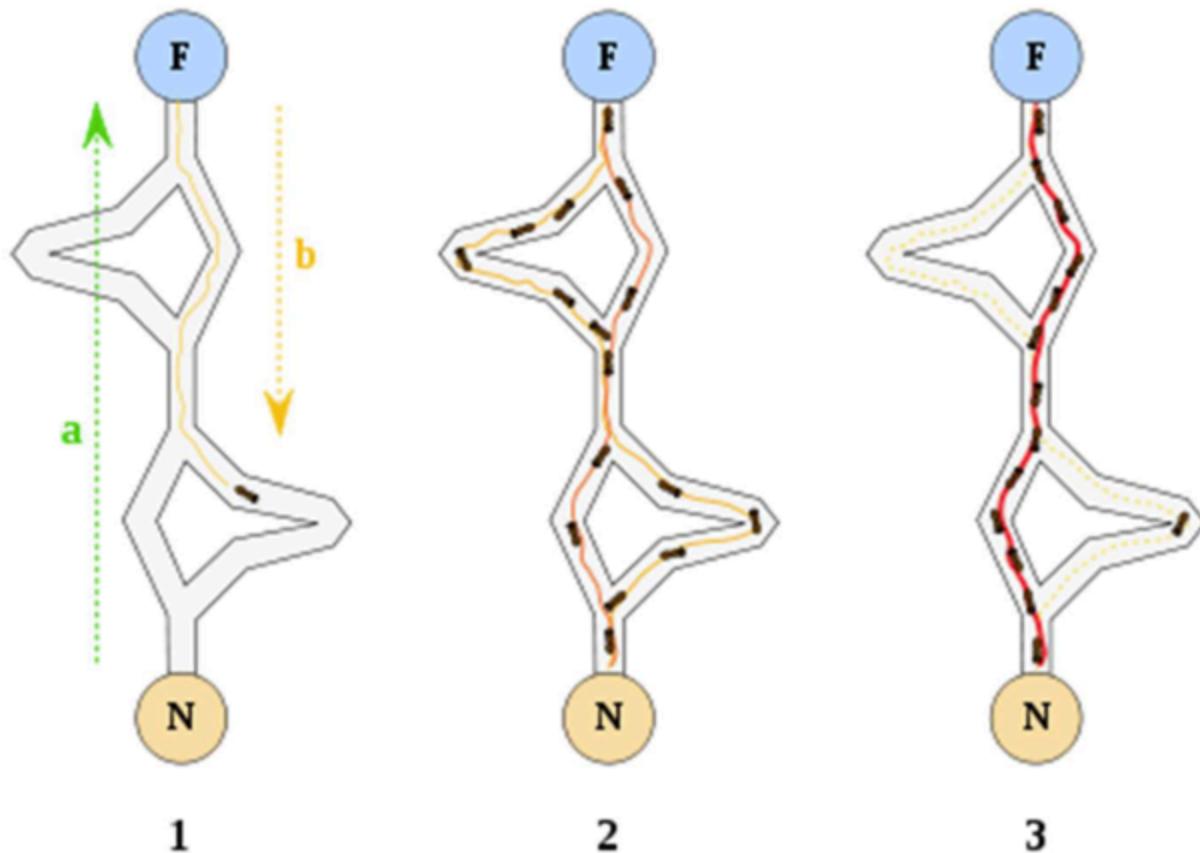
## Karınca Koloni Optimizasyonu ile Zeki Optimizasyon

Karınca Koloni Optimizasyonu, kombinasyonel optimizasyon söz konusu olduğunda, konuyu anlamak için en uygun algoritmadır. Söz konusu algoritma, karıncaların yiyecek kaynaklarına giden yolları birbirlerine işaret etmek için kullandıkları biyolojik mekanizmlara dayanan bir zeki optimizasyon algoritmasıdır.

Buna göre karıncalar, gerçek dünyada yiyecek yollarını işaretlemek için feremon (pheremone) adı verilen bir madde salgılamaktadır. Bu yolla yiyeceklerle giden en uygun yollar, sürüler için algılanır hale gelmektedir. Benzer şekilde kombinasyonal optimizasyon çözümleri için algoritma içerisinde tanımlanmış değişkenler birer feromon görevi görmekte, feromon değişken değerleri yüksek olan çözüm kombinasyonları bir fonksiyon üzerinden tespit edilebilmektedir.

Karınca Koloni Optimizasyonunda karınca adı verilen her bir parçacık ve kurulan problem modeli, şu mekanizmalara dayanarak optimizasyon çözümünü desteklemektedir:

- N adet her bir karıncanın taşıdığı feromon değerleri vardır. Benzer şekilde, problem kapsamında aralarında potansiyel bağ olduğu düşünülen unsurlar da (örneğin en kısa yol bulunması istenen farklı şehirler arası yollar) feromon değerleriyle temsil edilmektedir.
- Her döngü adımında karıncalar kombinasyonel adımlarla tespit ettikleri potansiyel unsurları problem fonksiyonundaki uygunluk/başarı değerini hesaplamak için kullanırlar.
- Uygun değer üreten karıncaların ve potansiyel çözüm unsurlarının feromon değerleri geliştirilir.
- Bir karıncanın aynı anda birden fazla potansiyel çözüm unsuruna yönelme durumu varsa, olasılıksal hesaplamalarla hangi tarafı tercih edeceği belirlenir.
- Algoritma tamamen çalışıp sona erdiğinde en çok feromon değeriyle desteklenen (en uygun) çözüm problemin çözümü olmaktadır. Karınca Koloni Optimizasyonu'nun genel çözüm süreci Şekil 7.4'tekine benzer bir akışı ortaya çıkarmaktadır. Şekilde N ve F arası en uygun rota zamanla olgunlaşan; optimum çözümü oluşturmaktadır.



Karınca Koloni Optimizasyonu kapsamında farklı potansiyel çözüm unsurları arası geçişlerde feromon değerlerinin güncellenmesi konusunda farklı matematiksel hesaplamalar gerçekleştirilmektedir (Şekil 7.5). Bu hesaplamalar doğrultusunda karıncalar problem çözümünde tipki etmen tabanlı modellemede olduğu gibi yeni adımlar atabilmektedir.

$$P_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_{c_{il} \in N(s^p)} \tau_{il}^\alpha \eta_{il}^\beta} & c_{il} \in N(s^p) \text{ ise} \\ 0 & \text{aksi takdirde} \end{cases} \quad (1)$$

$\tau_{ii}$  : ( $i, j$ ) köşelerindeki feromon iz miktarıdır.

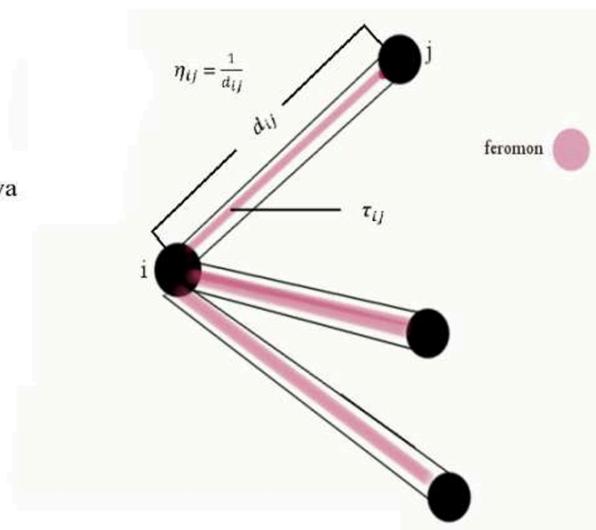
$\eta_{ij}$  : ( $i, j$ ) köşeleri arasındaki görünürlik (visibility) veya bağlantının amaç fonksiyonunun özelliğidir.

$$\eta_{ij} = \frac{1}{d_{ij}}$$

$d_{ij}$  =  $i$  ve  $j$  noktaları arasındaki uzaklıktır.

$$\tau_{ij} \leftarrow (1 - \rho) \tau_{ij} + \sum_{k=1}^m \Delta \tau_{ij}^k \quad (2)$$

$\rho$ : Feromon izinin buharlaşma oranı. ( $0 < \rho < 1$ )



Şekil 7.5. Karınca Koloni Optimizasyonu içerisinde atılacak yeni adımların hesaplanması

<https://www.youtube.com/watch?v=gkGa6WZpcQg> swarm opt örneği

## KOD

Aşağıdaki fonksiyonu 44 değerine eşitleyen degiskenlerin degerlerini bulalım.

$$f(x_1, x_2, x_3, x_4, x_5, x_6) = 4x_1 - 2x_2 + 3,5x_3 + 5x_4 - 11x_5 - 4,7x_6 = 44$$

Kütüphaneyi kurun (genetik algoritma için).

```
pip install pygad
```

```
import pygad
import numpy
```

```
function_inputs = [4, -2, 3.5, 5, -11, -4.7] ## fonksiyonun kat sayıları
desired_output = 44
```

```
def fitness_func(ga_instance, solution, solution_idx):
    output = numpy.sum(solution*function_inputs) # aslında burada 4x1-2x2+...
    fitness = 1.0 / numpy.abs(output - desired_output) # hata hesaplanır hata
    azaldıkça fitness artar . fitness fun hangi genin hayatı kalabileceğini
    artırır
```

```

    return fitness

fitness_function = fitness_func

fitness_function = fitness_func

num_generations = 100 # optimizasyonun toplam döngü sayısı
num_parents_mating = 7 # en iyi 7 birey çaprazlaması yeni nesil popülasyon
# için

sol_per_pop = 50 # popülasyon kaç kişiden oluşmakta buna bakılıyor
num_genes = len(function_inputs) # gen sayısı 6 çünkü 6 değişken var
last_fitness = 0

def nesil_ozeti(ga_instance): #Her nesil sonunda çalışacak fonksiyon
    global last_fitness
    print("Nesil = "
{generation}.format(generation=ga_instance.generations_completed))

    print("Fonksiyon Sonucu =
{fitness}".format(fitness=ga_instance.best_solution()[1]))
    # ga_instance.best_solution()[1] En iyi bireyin fitness değeri.
    Best_solution() şunu döndürür: (solution, fitness, index)

    print("Değişim = {change}".format(change=ga_instance.best_solution()[1] -
last_fitness))
    # fitness değişimini verir. Eğer pozitif ise fitness artmış, negatif ise
azalmış demektir.

    last_fitness = ga_instance.best_solution()[1]

ga_instance = pygad.GA(num_generations=num_generations,
                      num_parents_mating=num_parents_mating,
                      fitness_func=fitness_function,
                      sol_per_pop=sol_per_pop,
                      num_genes=num_genes,
                      on_generation=nesil_ozeti)

ga_instance.run()
"""

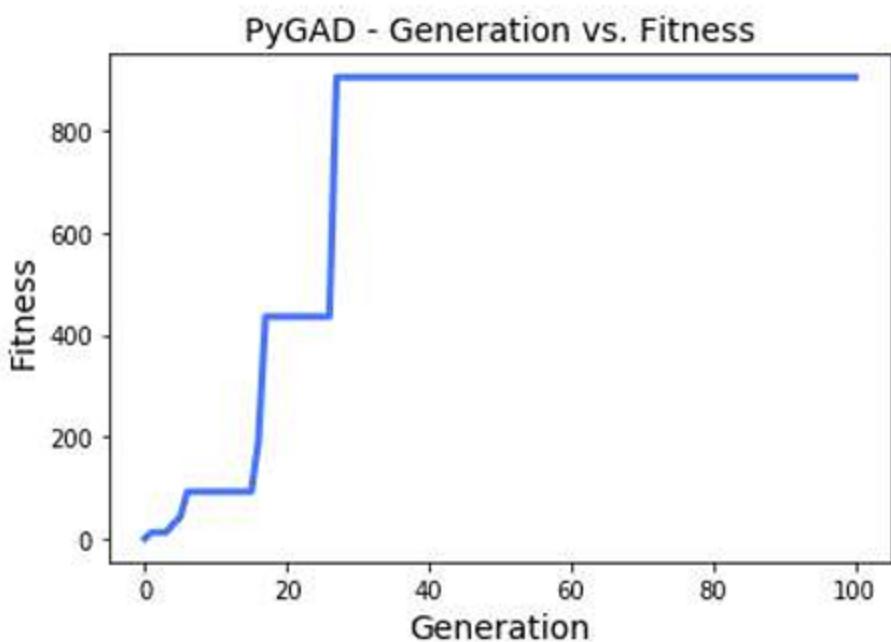
Rastgele popülasyon oluşturulur
Fitness hesaplanır

```

```
En iyiler seçilir  
Crossover yapılır  
Mutation yapılır  
Yeni nesil oluşur  
100 kere tekrar edilir  
"""
```

```
solution, solution_fitness, solution_idx = ga_instance.best_solution() # En  
iyi çözüm, o çözümün fitness değeri ve o çözümün popülasyondaki indeks  
numarası  
  
print("En uygun cozum degisken degerleri :  
{solution}".format(solution=solution))  
print("En uygun cozumu veren birey indeks no.:  
{solution_idx}".format(solution_idx=solution_idx))  
  
prediction = numpy.sum(numpy.array(function_inputs)*solution) # tahmin edilen  
değer  
print("En uygun cozum ile fonksiyon sonucu :  
{prediction}".format(prediction=prediction))  
  
if ga_instance.best_solution_generation != -1:  
    print("En uygun cozum {best_solution_generation} nesil sonra elde edildi."  
.format(best_solution_generation=ga_instance.best_solution_generation))
```

```
ga_instance.plot_fitness()
```



Şekil 7.12. En iyi/uygun değerin değişim grafiği

## Kod açıklamasız

```
import pygad

import numpy

function_inputs = [4,-2,3.5,5,-11,-4.7]

desired_output = 44

def fitness_func(solution, solution_idx):

    output = numpy.sum(solution*function_inputs)

    fitness = 1.0 / numpy.abs(output - desired_output)

    return fitness

fitness_function = fitness_func
```

```
num_generations = 100

num_parents_mating = 7

sol_per_pop = 50

num_genes = len(function_inputs)

last_fitness = 0

def nesil_ozeti(ga_instance):

    global last_fitness

    print("Nesil =
{generation}".format(generation=ga_instance.generations_completed))

    print("Fonksiyon Sonucu =
{fitness}".format(fitness=ga_instance.best_solution()[1]))

    print("Degisim = {change}".format(change=ga_instance.best_solution()[1] -
last_fitness))

    last_fitness = ga_instance.best_solution()[1]

ga_instance = pygad.GA(num_generations=num_generations,
                      num_parents_mating=num_parents_mating,
                      fitness_func=fitness_function,
                      sol_per_pop=sol_per_pop,
                      num_genes=num_genes,
                      on_generation=nesil_ozeti)
```

```
ga_instance.run()

solution, solution_fitness, solution_idx = ga_instance.best_solution()

print("En uygun cozum degisken degerleri :"
      "{solution}".format(solution=solution))

print("En uygun cozumu veren birey indeks no. :"
      "{solution_idx}".format(solution_idx=solution_idx))

prediction = numpy.sum(numpy.array(function_inputs)*solution)

print("En uygun cozum ile fonksiyon sonugu :"
      "{prediction}".format(prediction=prediction))

if ga_instance.best_solution_generation != -1:

    print("En uygun cozum {best_solution_generation} nesil sonra elde edildi."
          .format(best_solution_generation=ga_instance.best_solution_generation))
```