24.05.2023

OPTİMİZASYONDA METASEZGİSEL YÖNTEMLER
MBO Algoritmasının 0-1 Sırt Çantası Optimizasyon Problemine Uygulanması

İshak KUTLU
Bilişim Enstitüsü/Bilgisayar Bilimleri

1. Giriş

Bu raporda ayrık (discrete) ve ikili (binary) optimizasyon problemlerinden olan 0-1 sırt çantası (knapsack) problemi, aşağıda sunulan makalede önerilen göçmen kuşlar optimizasyon (migrating birds optimization, MBO) algoritması esas alınarak incelenmiştir.

- Ulker, E. ve Tongur, V. (2017). Migrating Birds Optimization (MBO) Algorithm to Solve Knapsack Problem. *Procedia Computer Science*, 111(2017), 71-76.

Ayrıca MBO algoritmasını daha detaylı incelemek amacıyla, aynı yazarın aşağıda sunulan doktora tezinden de yararlanılmıştır.

- Tongur, V. (2018). Ayrık Optimizasyon Problemlerinin Çözümünde Göçmen Kuşlar Optimizasyon (MBO) Algoritmasının İyileştirilmesi. *Doktora Tezi*, T.C. Selçuk Üniversitesi Fen Bilimleri Enstitüsü.

Bir seçim problemi olan 0-1 sırt çantası (knapsack) optimizasyon problemi, belirli kısıtlar altında toplam faydayı maksimize etmeyi amaçlar. Bu çerçevede d problemin boyutu, p_i ve w_i sırasıyla her boyutun/nesnenin fayda değeri ve ağırlığı/maliyeti ve C sırt çantasının maksimum kapasitesi olmak üzere, sırt çantası optimizasyon probleminin matematiksel formu aşağıda sunulmuştur.

$$f(x)_{max} = \sum_{i=1}^{d} p_i x_i \tag{1.1}$$

$$\sum_{i=1}^{d} w_i x_i \leq C \tag{1.2}$$

$$x_i \in \{0,1\}, i = 1, 2, \ldots, d \tag{1.3}$$

1 ve 2 no.lu denklemlerde yer alan x_i ifadesi, i. nesnenin çantaya konulması durumunda 1, konulmaması durumunda 0 değerini alan d boyutlu bir çözüm vektörünü temsil eder.

Sırt çantası optimizasyon problemi, faydaları ve ağırlıkları farklı bir dizi nesnenin, en yüksek faydayı sağlayacak şekilde kapasitesi C olan bir sırt çantasına yerleştirilmesi/seçilmesi olarak ifade edilebilir. Diğer bir ifadeyle 0-1 sırt çantası problemi kombinatoryal bir maksimizasyon problemine işaret eder. Sırt çantası problemi, aşağıdaki temsili 0-1 sırt çantası problemi üzerinden açıklanmıştır.

$w_i = \{2, 5, 4, 3, 1\}$, $p_i = \{5, 10, 15, 10, 20\}$, d = 5, C = 10

$x_i = [1, 1, 0, 0, 1]$

$$f(x)_{max} = \sum_{i=1}^{d} p_i x_i = 5x1 + 10x1 + 15x0 + 10x0 + 20x1 = 35 \quad (1.4)$$

$$\sum_{i=1}^{d} w_i x_i \leq C = 2x1 + 5x1 + 4x0 + 3x0 + 1x1 = 8 \quad (1.5)$$

Yukarıdaki x_i çözümünün ağırlığı/maliyeti 8 olduğu, diğer bir ifadeyle sırt çantasının kapasitesi C=10'u aşmadığı için geçerli bir çözümdür ve sunduğu çözümün faydası 35 değerine eşittir. Bu çerçevede 0-1 sırt çantası optimizasyon problemi, 0 ve 1'lerin dizilimine işaret eden bir permütasyon yapısı içerir.

Kesirli (fractional), sınırlı (bounded) ve 0-1 (zero-one) gibi çok sayıda farklı sırt çantası optimizasyon problemi bulunmaktadır. Tüm metasezgisel (metaheuristic) optimizasyon yöntemlerinde olduğu gibi sırt çantası optimizasyon problemi de geleneksel yöntemlerle makul bir sürede çözülmesi mümkün olmayan büyük boyutlu kombinatoryal optimizasyon problemlerin çözümünde kullanılır.

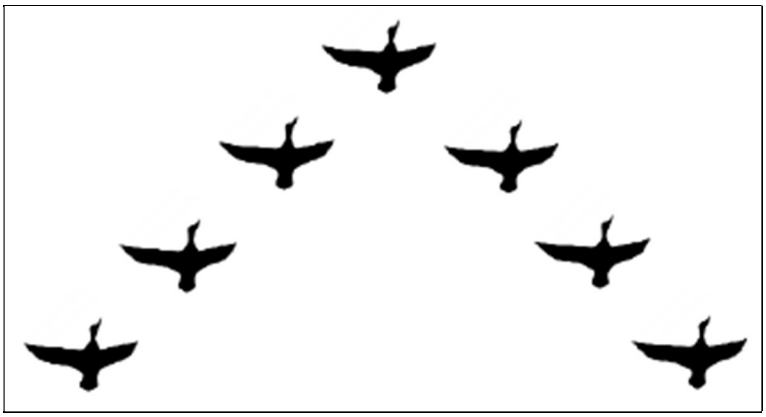
İncelenen makalede, 0-1 sırt çantası (zero-one knapsack) optimizasyon yöntemiyle çözülen ve kuşların göç sırasında oluşturduğu "V" biçiminden esinlenerek geliştirilen "göçmen kuşların optimizasyonu" (migrating birds optimization, MBO) isimli bir çalışma gerçekleştirilmiştir. Bu çerçevede "V" biçiminden esinlenen 0-1 sırt çantası optimizasyon algoritmasının, küçük boyutlu 10 farklı problemin çözümünde performansı araştırılmıştır. Buna göre algoritma, rassal bir çözüm kümesi ile başlatılır ve her adımda komşuluk (neighbor) yapısı ve fayda aktarım mekanizması kullanılarak çözüm kümesi iyileştirilmeye çalışılır.

Bir optimizasyon problemini çözümünde anahtar bir rol oynayan arama uzayını, çözüm temsilleri (solution representation) oluşturur. Arama uzayındaki tüm çözüm temsilleri amaç fonksiyonuna gönderildiğinde ise problemin manzarası (landscape) elde edilir. Çözüm temsilinde bütünlük, yakınlık, etkinlik şeklinde 3 temel özelliğin bulunması beklenir. Bütünlük, probleme ilişkin olası tüm çözümleri; yakınlık, iki çözüm arasında birinden diğerine en az bir geçişin bulunmasını; etkinlik ise çözümler arasında geçişin kolay bir şekilde gerçekleştirilebilmesini ifade eder. Problemin çözüm temsillerinin doğrusal ya da doğrusal olmayan bir şekilde kodlanması (encoding) mümkündür. Doğrusal olmayan temsilde, problemin çözümü bir graf ile temsil edilir. Doğrusal temsilde ise çözümler bir alfabe kullanılarak temsil edilir.

MBO algoritmasında kullanılan 0-1 sırt çantası probleminin çözüm temsilinde kullanılan alfabe [0,1} elemanlarından oluşur. Çünkü problemin çözüm temsilini ifade eden x_i vektörü 0 ve 1 şeklinde ikili (binary) değerlerden oluşur. Bu kapsamda d problemin boyutu olmak üzere, ayrık bir problemi ifade eden 0-1 sırt çantası probleminin arama uzayının büyüklüğü 2^d olur. İncelenen makalede 4, 5, 7, 10, 15, 20 ve 23 şeklinde farklı boyutlarda sırt çantası problemlerinin MBO algoritmasıyla çözümü araştırılmıştır.

2. MBO Algoritmasının Yapısı ve İşleyişi

Göçmen kuşlar optimizasyon (migrating birds optimization, MBO) algoritması, göçmen kuşların göç ederken sergiledikleri "V" diziliminden sağladıkları enerji tasarrufundan esinlenerek geliştirilmiştir (Şekil 2.1). Bu algoritma 2012 yılında Duman ve arkadaşları tarafından, ayrık problemlerden biri olan karesel atama problemi (quadratic assignment problem, QAP) için tasarlanmıştır. MBO kombinatoryal optimizasyon problemlerine kolaylıkla uygulanabilir olduğu için 0-1 sırt çantası gibi ikili (binary) optimizasyon problemlerinin çözümünde de kullanılmaktadır.



Şekil 2.1. MBO algortimasında V biçimli sürü

MBO, rassal bir başlangıç çözümüyle başlatılır. Sonrasında, her adımda çözümler geliştirilmeye çalışılır. MBO, çözümleri geliştirmek için komşuluk yapısını ve fayda aktarım mekanizmasını kullanılır. Komşuluk yapısı, algoritmanın yerel arama yapabilmesi için her mevcut çözümden komşu çözümler üretilmesi anlamına gelir. Fayda mekanizması ise lider kuş (çözüm) dahil herbir kuşun (çözümün) kullanmadığı kendi komşu çözümlerinden en iyilerinin, sürüdeki diğer kuşlarla (çözümlerle/bireylerle) paylaşılmasını ifade eder. Komşu çözüm paylaşımı, MBO algoritmasını diğer metasezgisel algoritmalardan ayıran en önemli özelliktir. Fayda aktarım mekanizması yoluyla sürüdeki her kuş birbirleriyle etkileşim halinde olmakta ve çözüme daha hızlı bir şekilde yakınsamaktadır.

Kuşların komşu çözüm üretme ve paylaşma süreci lider kuşun kanat çırpma sayısı kadar tekrar ettirilir. Diğer bir ifadeyle sürü kanat çırpma parametresi/sayısı kadar aynı formasyonda devam eder. Algoritmaya parametre olarak verilen lider kuşun kanat çırpma sayısı, her iterasyonda kuşların komşu çözüm üretme (ya da genel çözümü arama) süreci olarak düşünülebilir. Kanat çırpma değerine ulaşıldığında lider kuş, ilk olarak sürünün sol tarafının en sonuna gönderilirken, sol taraftan lideri takip eden diğer kuş liderin pozisyonuna geçer ve sol kanatta yer alan diğer kuşlar da sırasıyla birbirlerinin boşalan konumlarını doldururlar. Bu aşamada kanat çırpma parametresi sıfırlanır ve yeni bir dizilim elde edilir. Komşu çözüm üretme ve paylaşma işlemi yeni dizilim üzerinden devam ettirilir. Kanat çırpma değerine tekrar ulaşıldığında lider kuş, sürünün (sol değil) sağ tarafının en sonuna gönderilirken, sağ taraftan lideri takip eden diğer kuş liderin pozisyonuna geçer ve sağ kanatta yer alan diğer kuşlar da sırasıyla birbirlerinin boşalan konumlarını doldururlar. Bu aşamada kanat çırpma parametresi tekrar sıfırlanır ve başka bir yeni dizilim elde edilir. Komşu çözüm üretme ve paylaşma işlemi ikinci aşamada

üretilen yeni dizilim üzerinden devam ettirilir. Bu süreç sürünün sol-sağ kanadı üzerinde algoritma sonlandırılıncaya kadar devam ettirilir. Böylece mevcut çözümler optimum sonuca yaklaştırılmaya çalışılır.

Parametre	Açıklama
p	Popülasyonun başlangıç çözüm sayısı
k	Liderin komşu çözüm sayısı
n	Lider dışındaki her bireyin komşu çözüm sayısı
x	Komşu çözümlerin paylaşım sayısı
m	Kanat çırpma sayısı
d	Durdurma kriteri olarak problemin boyutu

Tablo 2.1. MBO algoritmasının parametreleri

MBO algoritmasında kullanılan parametreler Tablo 2.1'de verilmiştir. Bu parametreler takip eden alt başlıklarda detaylı olarak açıklanmıştır.

2.1. Başlangıç Popülasyonu (p)

Popülasyondaki her birey (kuş), bir permütasyon çözümüne sahip olup arama uzayında bir çözümü temsil eder. Permütasyonun uzunluğu, her biri 0 ya da 1 değerine sahip nesnelerin sayısına (yani problemin boyutuna) eşittir. Tüm alanları 0 değerlerinden oluşan başlangıç permütasyonu, kapasiteyi aşmayacak şekilde, permütasyon uzunluğu kadar rassal bir şekilde üretilen 1 değerleri ile doldurulur.

MBO algoritması başlangıç popülasyonu ile başlatılır. Popülasyon büyüklüğü parametresi sabit olup algoritma sonlanıncaya kadar aynı kalır. Sürünün "V" formasyonunu oluşturabilmesi için popülasyon büyüklüğü en az üç olmak üzere 3, 5, 7, 9 gibi tek sayılardan seçilmelidir. Örneğin popülasyon değeri 3 seçildiğinde bireylerden biri lider atanır; liderin sağına ve soluna ise birer birey yerleştirilir. Benzer şekilde eğer sürü 17 bireyden oluşturulursa, bireylerden birisi lider atanır; geriye kalan 16 bireyden 8'i liderin soluna diğer 8'i de liderin sağına yerleştirilir.

2.2. Komşuluk ve Paylaşım (k, n, x)

Mevcut çözüme yakın farklı yeni çözümlere komşu çözümler denir. MBO'da komşu çözüm üretmek için probleme göre değişebilen takas (swap), araya ekleme (insertion) gibi tekil ya da hibrit yöntemler kullanılabilir. Bunlar problemin çözümünde test edilir ve en başarılı komşu çözüm üretme yöntemi tercih edilir.

MBO'da lider (kuş) en az üç komşu çözüm üretir. Üretilen en iyi çözüm, mevcut çözümü iyileştirmek için kullanılır. Kalan çözümler, sol ve sağ tarafta lider kuşu takip eden bireylerle (kuşlarla) paylaşılır. Oluşturulan ve paylaşılan komşu çözümlerin sayısı, bir parametre olarak verilir. Bu parametreler aşağıdaki şekilde belirlenir.

$$s \in N^+; k = 2s+1; k = \{3, 5, 7, \ldots\} \quad (2.1)$$

$$x \in N^+; 1 \leq x \leq \left(\frac{k-1}{2}\right); x = \{1, 2, 3, \ldots, (k-1)/2\} \quad (2.2)$$

$$n = k - x \quad (2.3)$$

Burada k ifadesi liderin komşu çözümlerinin sayısını, x herbir tarafla (kanatla) paylaşılan komşu çözümlerin sayısını, n ise lider kuş dışında kalan kuşların komşu çözümlerinin sayısını temsil eder. Liderin en az üç komşu çözüme (k) sahip olması gerektiği göz önüne alındığında, s ifadesi 0'dan büyük bir doğal sayı olmak zorundadır.

Komşu çözüm sayısı parametresi sabit olup algoritma sonlanıncaya kadar aynı kalır. Ancak komşu çözüm üretme sayısı, lider (kuş) ve diğer bireyler (kuşlar) için farklı değerdedir. Toplamda her birey aynı sayıda komşu çözüme sahip olmasına rağmen, lider ve diğer bireyler tarafından üretilen komşu çözümlerin sayısındaki farklılık, paylaşılan komşu çözümlerden kaynaklanmaktadır. Örneğin Denklem 2.1'e göre s=1 iken lider 3 komşu çözüm üretir (k=3). Denklem 2.2'ye göre k=3 iken paylaşılan komşu çözüm sayısı 1 olarak hesaplanır (x=1). Denklem 2.3'e göre k=3 ve x=1 iken lider dışındaki her bireyin komşu çözüm üretme sayısı 2 olur (n=2). O halde, lider dışında sürüdeki her bireyin komşu çözüm üretme sayısı 2 ile paylaşım yoluyla kendisinden önceki bireyden aldığı çözüm sayısı 1'in toplamı, liderin komşu çözüm sayısı 3 değerine eşit olacaktır. Diğer bir ifadeyle liderin ve diğer bireylerin (kendisinden önceki bireyden aldığı paylaşılan çözüm dahil) komşu çözüm sayıları birbirine eşittir.

Her birey için komşuluk metodu, diğer bir ifadeyle komşu çözüm üretme yöntemi aynıdır. Değeri 0 olan permütasyon alanlarından biri rassal şekilde seçilir ve belirlenen alana 1 değeri atanır. Sırt çantasının kapasitesi aşılırsa, bu defa değeri 1 olan permütasyon alanlarından biri rassal olarak seçilir ve değeri 0 olarak değiştirilir. Böylece yeni bir komşu çözüm elde edilir. Üretilen komşu çözüm amaç (objective) fonksiyonuna göre değerlendirilir. Aşağıdaki örnekte w_i her nesnenin ağırlığını, p_i faydasını, d problemin boyutunu ve C ise kapasitesini temsil etmektedir.

$w_i = \{2, 5, 4, 3, 1\}$, $p_i = \{5, 10, 15, 10, 20\}$, d = 5, C = 10

0	1	0	1	0		0	1	1	1	0		0	1	1	0	0
Adım 1						Adım 2						Adım 3				

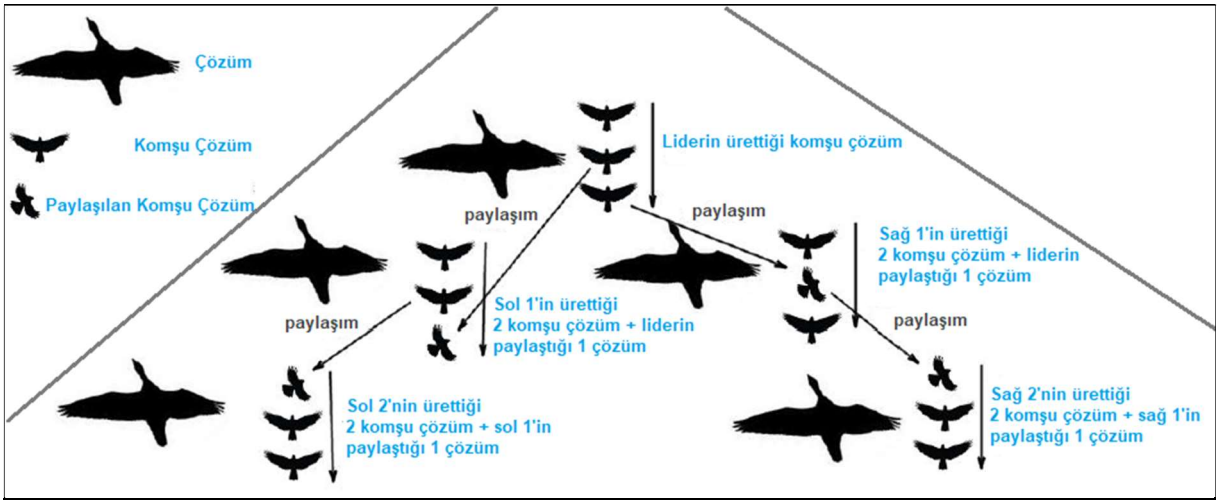
Tablo 2.2. MBO algoritmasının komşu çözüm üretimi

Tablo 2.2'de Adım 1, herhangi bir aşamada üretilen komşu çözümleri temsil etmektedir. Adım 1'de ağırlıkların toplamı (1x5+1x3=8) 8, faydaların toplamı ise (10+10=20) 20'dir. Adım 2'de yeni bir komşu çözüm üretmek için değeri 0 olan permütasyon alanlarından üçüncüsü rassal olarak seçilmiş ve belirlenen alana 1 değeri atanmıştır. Ancak Adım 2'de gri alan ile işaretlenen komşu çözümle birlikte, ağırlıkların toplamı (1x5+1x4+1x3=12) kapasiteyi (10) aşmıştır. Bu yüzden Adım 3'te bu defa değeri 1 olan permütasyon alanlarından dördüncüsü rassal olarak seçilmiş ve değeri 0 olarak değiştirilmiştir. Böylece Adım 3'te gri alan ile işaretlenen komşu çözümün çıkarılmasıyla (yani değerinin 0 yapılmasıyla) birlikte, ağırlıkların toplamı (1x5+1x4=9) kapasitenin altında gerçekleşmiştir.

Üretilen komşu çözümlerin faydası büyükten küçüğe doğru sıralanır. En iyi komşu çözüm, mevcut çözümü iyileştirmek için kullanılır. Kalan x adet komşu çözüm, lideri takip eden bireylerle paylaşılır. Lider, sürünün sol ve sağ tarafı ile x'er adet komşu çözümü, önce sol taraftan başlamak suretiyle sol-sağ adımlarını tekrar ederek paylaşır. Bu yüzden, liderin ürettiği komşu çözümlerin sayısını temsil eden k ifadesi, tek sayılardan oluşur. Çünkü lider, en iyi çözümü kendisine ayırdığı için (sol ve sağ tarafla) paylaşılan komşu çözümlerin sayısı (beklendiği gibi) çift sayı olacaktır. Lider dışında kalan bireyler, x komşu çözümlerini bir sonraki bireyle paylaşır.

Örneğin s=2, dolayısıyla k=5 iken, liderin (herbir taraftaki) diğer bireylerle paylaştığı komşu çözüm sayısı x=2 olacaktır. Diğer bir ifadeyle, liderin kendisine ayırdığı en iyi çözüm dışında kalan 4 komşu çözümün 2'si, sol taraftan lideri takip eden bir bireyle; kalan 2'si ise sağ taraftan lideri takip eden bir bireyle paylaşılır. Bu durumda liderin ürettiği komşu çözümlerden ikinci en iyi liderin soluna, üçüncü en iyi liderin sağına, dördüncü en iyi yine liderin soluna ve beşinci en iyi (son komşu çözüm) liderin yine sağına olacak şekilde ardışık paylaştırılır. Denklem 2.2'de yer alan eşitsizlik göz önüne alınırsa, k=7 ve x=2 durumunda ise liderin ürettiği komşu çözümler sıralandıktan, kendisi ve kendisini takip eden bireylerle gerekli paylaşımlar yapıldıktan sonra, kalan son iki komşu çözüm atılır.

Benzer şekilde s=1, dolayısıyla k=3 iken, liderin (herbir taraftaki) diğer bireyle paylaştığı komşu çözüm sayısı x=1 olacaktır. Liderin kendisine ayırdığı en iyi çözüm dışında kalan 2 komşu çözümün 1'i, sol taraftan lideri takip eden bir bireyle; kalan 1'i ise sağ taraftan lideri takip eden bir bireyle paylaşılır. Bu durum başlangıç popülasyonunun p=5, liderin komşu çözüm sayısının k=3 ve paylaşılan çözüm sayısının x=1 kabul edildiği Şekil 2.2'de gösterilmiştir.



Şekil 2.2. Komşu çözüm üretimi ve paylaşımı (k=3 ve x=1)

Şekil 2.2'ye göre lider 3 komşu çözüm üretir. Söz konusu çözümler diğer bireylerle paylaşılmadan önce amaç fonksiyonunda iyiden kötüye doğru sıralanır. Lider tarafından üretilen en iyi (birinci sıradaki) komşu çözüm mevcut çözümü (yani liderin çözümünü) geliştirmek için ayrılır. En iyi ikinci ve üçüncü çözümler, sırasıyla, lideri soldan ve sağdan takip eden bireylerin çözümü için kullanılır. Lider dışındaki bireylerin ise 2'şer çözüm ürettiği görülmektedir. Ancak soldan birinci sıradaki bireyin, kendi ürettiği 2 komşu

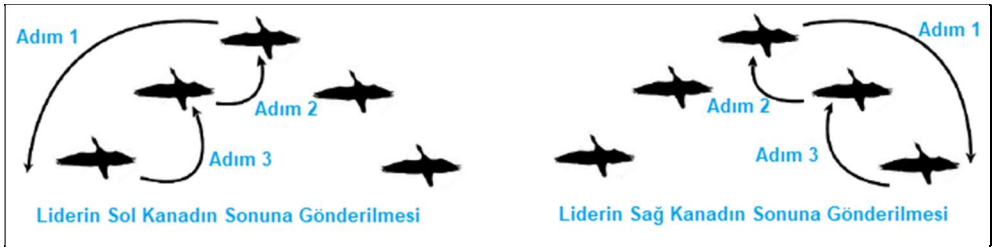
çözüm ve liderin paylaştığı 1 komşu çözümle birlikte 3 komşu çözümü; soldan ikinci sıradaki bireyin ise kendi ürettiği 2 komşu çözüm ve soldan birinci sıradaki bireyin paylaştığı 1 komşu çözümle birlikte 3 komşu çözümü vardır. Benzer şekilde sağdan birinci sıradaki bireyin, kendi ürettiği 2 komşu çözüm ve liderin paylaştığı 1 komşu çözümle birlikte 3 komşu çözümü; sağdan ikinci sıradaki bireyin ise kendi ürettiği 2 komşu çözüm ve sağdan birinci sıradaki bireyin paylaştığı 1 komşu çözümle birlikte 3 komşu çözümü vardır. Dolayısıyla popülasyonda yer alan tüm bireyler eşit sayıda komşu çözüme sahiptir.

Soldan ve sağdan, Şekil 2.2'ye göre birinci ve ikinci sıradaki tüm bireyler de paylaşım yoluyla aldığı çözümler dahil olmak üzere, (liderin yaptığı gibi) komşu çözümlerinin tümünü, öncelikle, amaç fonksiyonunda iyiden kötüye doğru sıralar ve en iyi çözümü kendilerine ayırır. İkinci en iyi çözümü ise kendilerini takip eden diğer bireyle paylaşırlar. Paylaşımdan sonra mevcut çözümün kendisinde kalan üçüncü komşu çözüm ise atılır. Diğer bir ifadeyle çözüm kümesindeki eleman sayısı sabit olduğundan dolayı (ki örneğe göre her bireyin çözüm kümesi 3 elamanlı olup) paylaşımdan sonra mevcut çözümün kalan en kötü komşu çözümü ya da çözümleri atılır.

Sürüde kendi komşu çözümlerini diğer bireylerle paylaşamayan sadece iki birey vardır. Bunlar sürünün sol ve sağ tarafın en sonunda bulunan bireylerdir. Bu bireyler, takip ettikleri bireylerden kendilerine gönderilen (paylaşılan) çözümleri kabul ederler. Ancak kendilerinden sonra birey olmadığı için ürettikleri komşu çözümleri paylaşamazlar. Bununla birlikte en sonda bulunan bireyler, kendi komşu çözümlerini paylaşamama dışında, diğer bireylerle aynı davranışları sergiler. Bunlar paylaşım yoluyla aldıkları komşu çözümler dahil, kendi ürettikleri komşu çözümleri amaç fonksiyonunda en iyiden kötüye doğru sıralar ve en iyi birinci komşu çözümü kendilerine ayırır. Kalan komşu çözümleri ise atarlar.

2.3. Kanat Çırpma Parametresi ve Lider Değişimi (m)

Kanat çırpma parametresi, "V" biçimli sürünün liderinin, diğer bir ifadeyle sürünün diziliminin ne sıklıkta değiştirileceğini ifade eder. Her değişim gerçekleştirildiğinde sürünün belirli bir iterasyon boyunca aynı dizilimde kalması sağlanır. Bu iterasyon sayısını kanat çırpma parametresi belirler.



Şekil 2.3. Sol ve sağ kanat üzerinde lider değişimi

Sürünün ilk lider değişimi, sol tarafında gerçekleştirilir (Şekil 2.3). Bu aşamada sürünün lideri, sol tarafın en arkasına gönderilirken, lideri soldan takip eden birinci sıradaki birey liderin konumuna geçer. Soldan birinci sıradaki bireyin liderliğe geçmesinden kaynaklanan (birinci sıradaki) boşluğu, soldan ikinci sıradaki birey; soldan ikinci sıradaki bireyin boşalan yerini ise (varsa) soldan üçüncü sıraki birey doldurur. Diğer bir ifadeyle soldaki tüm bireyler birer basamak lidere doğru yaklaştırılmak suretiyle aradaki

boşluklar doldurulur. Bu değişim gerçekleştikten sonra ise kanat çırpma parametresi sıfırlanır.

Sürünün ikinci kez lider değişimi ise sağ tarafında gerçekleştirilir (Şekil 2.3). Bu aşamada sürünün lideri, sağ tarafın en arkasına gönderilirken, lideri sağdan takip eden birinci sıradaki birey liderin konumuna geçer. Sağdan birinci sıradaki bireyin liderliğe geçmesinden kaynaklanan (birinci sıradaki) boşluğu, sağdan ikinci sıradaki birey; sağdan ikinci sıradaki bireyin boşalan yerini ise (varsa) sağdan üçüncü sıraki birey doldurur. Diğer bir ifadeyle sol tarafta gerçekleşen sürecin aynısı sağ tarafta da gerçekleşir. Bu değişim gerçekleştikten sonra kanat çırpma parametresi tekrar sıfırlanır.

Lider değişimi algoritmanın çalışması sonlandırılıncaya kadar, yukarıda açıklandığı gibi sürünün sol ve sağ taraflarında sıralı ve yinelemeli bir şekilde gerçekleştirilir. Diğer parametrelerde olduğu gibi kanat çırpma parametresi de algoritmanın başlangıcında belirlenir ve algoritma sonlandırılıncaya kadar aynı kalır.

2.4. Durdurma Kriteri (d)

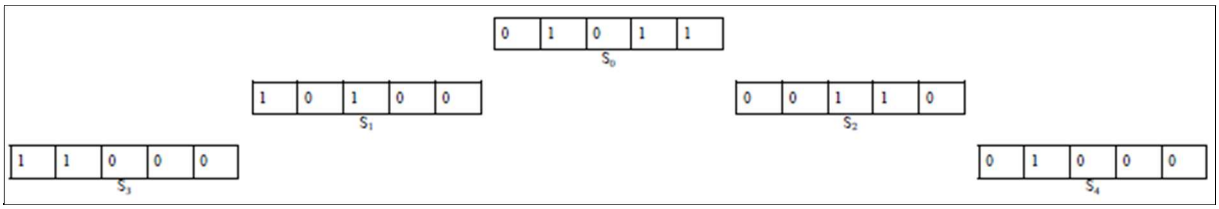
Durdurma kriteri ya da iterasyon parametresi, algoritmanın yineleme sayısının belirlendiği parametredir. Durdurma kriteri, sırt çantası problemindeki nesne sayısını ifade eden problemin boyutuna veya belirli bir kalitedeki çözümün garanti edilmesine bağlı olarak belirlenebilir. Duman ve arkadaşları, algoritmanın başlangıcından itibaren üretilen toplam komşu çözüm sayısı, problem boyutunun küpüne eşit olacak şekilde belirlemiştir. Bu yaklaşıma göre MBO algoritmasında ele alınan problemin boyutu büyüdükçe algoritmanın çalışma zamanı da artmaktadır.

2.5. MBO Algoritmasının Örnek Bir Sırt Çantası Probleminde İşleyişi

C kapasite, d problemin boyutu, w_i ağırlık ve p_i fayda olmak üzere, MBO ile temsili bir sırt çantası probleminin çözümü aşağıda gösterilmektedir.

$w_i = \{2,5,7,3,1\}$, $p_i = \{10,30,70,50,1\}$, $d = 5$, $C = 10$

Temsili problemin boyutu d=5 olduğu için popülasyonun herbir bireyi 5 alandan ve ikili değerlerden oluşan permütasyon çözümüne sahip olur. Şekil 2.4'te popülasyonun 1 lider ve 2'şer de sol ve sağ kanatta olmak üzere toplam p=5 bireyi/çözümü olduğu görülmektedir. Popülasyonun her bireyi/çözümü ise yine 5 alandan oluşan birer permütasyondan meydana gelir. Permütasyonların 5 alandan oluşması problemin boyutunun d=5 olmasından kaynaklanmasına rağmen, popülasyonun başlangıç çözümü/birey sayısı p=5 ise problemin boyutundan tamamen bağımsız bir şekilde belirlenir. Eğer popülasyonun başlangıç çözümü (birey sayısı) p=7 olsaydı, herbiri 5 alandan oluşan 7 farklı permütasyon çözümü olurdu. Diğer taraftan p_i ifadesinin, popülasyonun başlangıç sayısını temsil eden p ifadesinden tamamen farklı bir parametre olduğu belirtilmelidir.



Şekil 2.4. Komşu çözüm üretimi ve paylaşımı (k=3 ve x=1)

Şekil 2.4'te S_0 lideri, S_1 ve S_3 sürünün sol tarafında yer alan bireyleri, S_2 ve S_4 ise sürünün sağ tarafında yer alan bireyleri temsil eder.

Neighbor solutions of each bird for *k=3* and *x=1*.

Bird	Permutation	Fitness (Sum of Profit)	Neighbor	Permutation	Fitness (Sum of Profit)
			N_{01}	00110	80
S_0	01011	81	N_{02}	00101	71
			N_{03}	11001	41
			N_{11}	10101	81
S_1	10100	80	N_{12}	00110	80
			N_{13} (Shared=N_{02})	00101	71
			N_{21}	10010	60
S_2	00110	120	N_{22}	00011	51
			N_{23} (Shared=N_{03})	11001	41
			N_{31} (Shared=N_{12})	00110	80
S_3	11000	40	N_{32}	00010	50
			N_{33}	11001	41
			N_{41}	01010	80
S_4	01000	30	N_{42} (Shared=N_{22})	00011	51
			N_{43}	01001	31

Tablo 2.3. Her bireyin komşu çözüm üretimi ve paylaşımı (k=3 ve x=1)

Tablo 2.3'te yer alan N ifadeleri, her bireyin sahip olduğu komşu çözümleri temsil eder ve kendi içlerinde en iyiden kötüye doğru sıralanmıştır. Her birey sahip olduğu komşu çözümlerden en iyisini (yani birincisi sıradakini) kendi çözümünü geliştirmek için kullanır. Diğerini (yani ikincisini) ise kendisinden sonra gelen birey ile paylaşır. Temsili örnekte x=1 olduğu için her bireyin en kötü (yani üçüncü) komşu çözümü atılır.

Tablo 2.3'te yer alan N_{01}, N_{02} ve N_{03} ifadeleri S_0 ile temsil edilen liderin ürettiği komşu çözümleri (neighbor solutions) ifade eder. N_{11} ve N_{12} ifadeleri ise sol kanatta yer alan S_1 bireyinin ürettiği komşu çözümleri, N_{13} ise liderin (yani S_0'ın) S_1 bireyi ile paylaştığı komşu çözümü temsil eder. Benzer şekilde N_{21} ve N_{22} ifadeleri sağ kanatta yer alan S_2 bireyinin ürettiği komşu çözümleri, N_{23} ise liderin (yani S_0'ın) S_2 bireyi ile paylaştığı komşu çözümü temsil eder. Diğer taraftan N_{32} ve N_{33} ifadeleri sol kanatta yer alan S_3

bireyinin ürettiği komşu çözümleri, N_{31} ise yine sol tarafta yer alan S_1 bireyinin S_3 bireyi ile paylaştığı komşu çözümü temsil eder. Benzer şekilde N_{41} ve N_{43} ifadeleri sağ kanatta yer alan S_4 bireyinin ürettiği komşu çözümleri, N_{42} ise yine sağ kanatta yer alan S_2 bireyinin S_4 bireyi ile paylaştığı komşu çözümü temsil eder.

Bu çerçevede MBO algoritması lider için özetle şu adımlardan oluşur:

- Üretilen komşu çözümlerin uygunluğu (fitness) en iyiden kötüye doğru sıralanır.
- En iyi (birinci sıradaki) komşu çözüm liderin kendi çözümünü geliştirmede kullanılır.
- İkinci sıradaki komşu çözüm, sürünün sol tarafında yer alan lideri takip eden birey ile paylaşılır.
- Üçüncü sıradaki komşu çözüm, sürünün sağ tarafında yer alan lideri takip eden birey ile paylaşılır.

Yukarıdaki süreç sadece lider kuş içindir. Lider dışındaki diğer bireyler için komşu çözümlerin paylaşımı sürünün kendi tarafında gerçekleşir. Örneğin sürünün sol tarafında yer alan S_1'in komşu çözümleri, yine sürünün sol tarafında yer alan S_3 ile paylaşılır. Benzer şekilde sürünün sağ tarafında yer alan S_2'nin komşu çözümleri, yine sürünün sağ tarafında yer alan S_4 ile paylaşılır.

Bu çerçevede MBO algoritması lider dışındaki diğer bireyler için özetle şu adımlardan oluşur:

- Üretilen ve önündeki bireyden alınan komşu çözümlerin uygunluğu (fitness) en iyiden kötüye doğru sıralanır.
- En iyi (birinci sıradaki) komşu çözüm bireyin kendi çözümünü geliştirmede kullanılır.
- İkinci sıradaki komşu çözüm, (kendi kanadında) bireyi takip eden diğer birey ile paylaşılır.
- Üçüncü sıradaki komşu çözüm atılır.

$f(.)$ amaç fonksiyonu olmak üzere, eğer $f(S_i) < f(N_{i1})$ ise $S_i = N_{i1}$, değil ise S_i güncellenmez. Yukarıdaki açıklamalar ışığında Tablo 2.3'ün Fitness kolonunda yer alan bireylerin yeni çözümleri $S_0 = 81$, $S_1 = 81$, $S_2 = 120$, $S_3 = 80$ ve $S_4 = 80$ olacaktır.

3. Makalede MBO Algoritmasının Sırt Çantası Problemine Uygulanması

MBO algoritmasının performansı, literatürden seçilen 10 sırt çantası problemi üzerinde test edilmiştir. Problemlerin çözümünde uygulanan MBO algoritmasının parametreleri Tablo 3.1'de yer almaktadır.

Parameter	Value
Bird (*p*)	71
Neighbor (*k*)	5
Flap (*m*)	30
Shared neighbor (*x*)	1
Stop criteria	problem dimension

Tablo 3.1. MBO parametreleri

Her problem için 30 kez çalıştırılan MBO algoritmasının ürettiği en iyi çözümler aşağıda yer alan Tablo 3.2'de sunulmuştur.

Problem	Optimum Value [9]	Best	Average	Worst	Time(*s*)
P_1	295	**295**	294.6666	293	0.002
P_2	1024	**1024**	1011.1	991	0.003
P_3	35	**35**	35	35	0.0004
P_4	23	**23**	23	23	0.0004
P_5	481.0694	**481.0694**	431.4180	399.8016	0.002
P_6	50	**52**	52	52	0.001
P_7	107	**107**	107	107	0.0008
P_8	9767	9765	9765	9765	0.013
P_9	130	**130**	130	130	0.0005
P_{10}	1025	**1025**	1011.2	990	0.003

Tablo 3.2. MBO algoritmasının herbir problem için ürettiği en iyi çözümler

Tablo 3.2'de görüldüğü üzere, MBO, 8 no.lu problemin dışındaki tüm problemlerin optimal çözümlerini bulmuştur. MBO, optimal çözümleri çok kısa bir sürede bulmanın yanı sıra, 6 no.lu problem için optimal çözümün üzerinde bir çözüm sunmuştur.

Tablo 3.3'te ise her problemin boyutu ve en iyi çözüm permütasyonları sunulmuştur.

Problem	Dimension	Solution Permutation	Best Result
P_1	10	0, 1, 1, 1, 0, 0, 0, 1, 1, 1	295
P_2	20	1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1	1024
P_3	4	1, 1, 0, 1	35
P_4	4	0, 1, 0, 1	23
P_5	15	0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1	481.0694
P_6	10	0, 0, 1, 1, 0, 1, 1, 1, 1, 1	52
P_7	7	1, 0, 0, 1, 0, 0, 0	107
P_8	23	1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0	9765
P_9	5	1, 1, 1, 1, 0	130
P_{10}	20	1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1	1025

Tablo 3.3. MBO algoritmasının en iyi çözüm permütasyonları

MBO, literatürden seçilen 10 adet 0-1 sırt çantası probleminin 9'unun optimal çözümlerini hızlı bir şekilde bulmuştur.

4. Python ile MBO, 0-1 Sırt Çantası, Kesirli Sırt Çantası ve Sınırlandırılmış Sırt Çantası Algoritmalarının Kodlanması ve Karşılaştırılması

Bu araştırma, Spyder IDE'si kullanılarak Python 3.9 programlama dilinde gerçekleştirilmiştir. Çalışmada numpy, ve random kütüphanelerinden yararlanılmıştır.

Çalışmanın yapıldığı bilgisayarın teknik özellikleri aşağıda listeleniştir.

- CPU: AMD Ryzen 5 3500X 6-Core Processor, 3600 Mhz, 6 Core(s), 6 Logical Processor(s)
- GPU: NVIDIA GeForce RTX 3060, VRAM 12 GB
- RAM: 16 GB
- İşletim Sistemi: Windows 10

Problem, 0-1 sırt çantası problemine ilişkin literatürde yaygın kullanılan bir optimizasyon problemidir. Çalışmada (ya da incelenen makalede) kullanılan 10 adet 0-1 sırt çantası problemine aşağıdaki bağlantıdan ulaşılabilir.

http://artemisa.unicauca.edu.co/~johnyortega/instances_01_KP/

Algoritmanın Python ile kodlanması aşamasında, sözde kodlarda yer almayan, başlangıç çözümleri optimize edilmiştir. Başlangıç çözümleri optimize edildikten sonra algoritmanın bir miktar daha verimli çalıştığı, diğer bir ifadeyle optimal çözümlere ulaşma kararlığının arttığı gözlenmiştir. Bu çerçevede Python ile kodlama aşamasında başlangıç çözümleri optimize edilen MBO algoritmasının sonuçları Tablo 4.1'de sunulmuştur. Algoritma, makalede kullanılan parametre değerlerine göre çalıştırılmıştır.

```
Migrating Bird Optimization (MBO)

P 1   |  profit:  295   |  [0, 1, 1, 1, 0, 0, 0, 1, 1, 1]
P 2   |  profit:  1024  |  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1]
P 3   |  profit:  35    |  [1, 1, 0, 1]
P 4   |  profit:  23    |  [0, 1, 0, 1]
P 5   |  profit:  481   |  [0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1]
P 6   |  profit:  52    |  [0, 0, 1, 0, 1, 1, 1, 1, 1, 1]
P 7   |  profit:  107   |  [1, 0, 0, 1, 0, 0, 0]
P 8   |  profit:  9767  |  [1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
P 9   |  profit:  130   |  [1, 1, 1, 1, 0]
P 10  |  profit:  1025  |  [1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1]
```

Tablo 4.1. Python ile kodlanan MBO algoritmasının herbir problem için ürettiği çözümler

Makalede, 8 no'lu problemin optimum çözümü küçük bir sapmayla elde edilmesine rağmen, Python ile kodlanan algoritma tüm problemlerin optimum çözümünü bulmuştur. Makalede ulaşılan sonuca benzer şekilde 6 no.lu problemin çözümü, optimumdan bir miktar daha iyi gerçekleşmiştir. Diğer taraftan makalede algoritmanın 30 kez çalıştırılması sonucu ürettiği en iyi çözümlere yer verilirken, Tablo 4.1'de yer alan sonuçlara algoritmanın neredeyse her çalıştırılmasında ulaşılabilmektedir.

MBO ile kıyaslanması amacıyla Python ile kodlanan repair ve optimization yöntemiyle, algoritmanın 30 kez çalıştırılması sonucunda ulaştığı en iyi çözümler Tablo 4.2'de yer almaktadır.

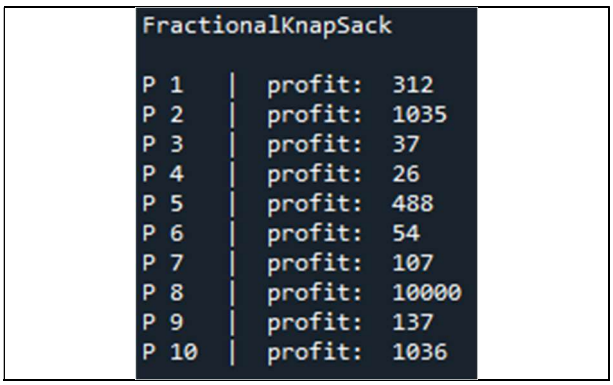
```
ZeroOneKnapSack

P 1   |  profit:  295   |  [0, 1, 1, 1, 0, 0, 0, 1, 1, 1]
P 2   |  profit:  1018  |  [1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
P 3   |  profit:  35    |  [1, 1, 0, 1]
P 4   |  profit:  22    |  [0, 1, 1, 0]
P 5   |  profit:  481   |  [0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1]
P 6   |  profit:  52    |  [0, 0, 1, 0, 1, 1, 1, 1, 1, 1]
P 7   |  profit:  107   |  [1, 0, 0, 1, 0, 0, 0]
P 8   |  profit:  9748  |  [1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0]
P 9   |  profit:  130   |  [1, 1, 1, 1, 0]
P 10  |  profit:  1019  |  [1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1]
```

Tablo 4.2. Python ile kodlanan 0-1 sırt çantası optimizasyon yönteminin herbir problem için ürettiği çözümler

Tablo 4.2'de yer alan sonuçlar incelendiğinde, 2, 4, 8 ve 10 no.lu problemlerin çözümü optimum çözümlere ulaşamamıştır. Sonuçlar MBO ile kıyaslandığında, MBO'nun daha etkin sonuçlar ürettiği görülmektedir.

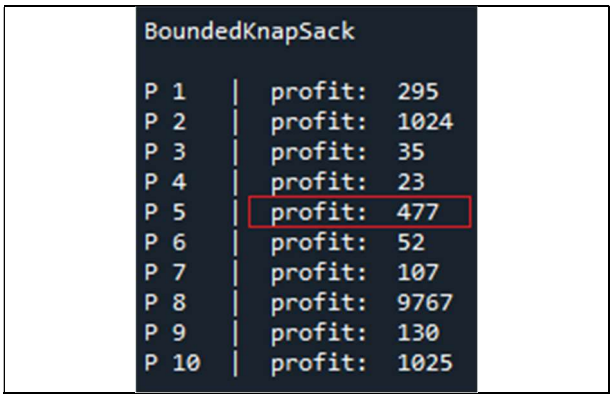
Bir başka sırt çantası optimizasyon yöntemi olan kesirli sırt çantası (fractional knapsack) optimizasyon yönteminin integer değere dönüştürülen sonuçları ise Tablo 4.3'te sunulmuştur.



Tablo 4.3. Python ile kodlanan kesirli sırt çantası optimizasyonun ürettiği çözümler

Kesirli sırt çantası yöntemi, nesnelerin parçalanabildiği esasına dayandığından, MBO algoritmasından, hatta optimal çözümlerden daha iyi sonuçlar üretmiştir. Ancak MBO ayrık problemler, kesirli sırt çantası yöntemi ise sürekli problemlerin çözümünde kullanıldığından ikisinin kıyaslanması doğru bir yaklaşım olmayabilir.

Sınırlandırılmış sırt çantası (bounded knapsack) optimizasyon yönteminin ürettiği sonuçlar ise Tablo 4.4'te gösterilmiştir.



Tablo 4.4. Python ile kodlanan sınırlandırılmış sırt çantası optimizasyonunun ürettiği çözümler

Sınırlandırılmış sırt çantası probleminde, kapasiteye ilave olarak seçilecek nesne adedi de bir kısıt olarak bulunur. Ancak seçilecek nesne adedi, problemin boyutuna eşit olacak şekilde belirlendiğinde, problem 0-1 sırt çantası problemine dönüşür. Tablo 4.4'te seçilecek nesne adedi, problemin boyutuna eşit olarak belirlenmiştir. Bu çerçevede sınırlandırılmış sırt çantası yöntemi 5 no.lu problemin çözümü dışında diğer tüm çözümleri optimal seviyede bulmuştur. Optimal çözümler kapsamında sınırlandırılmış sırt çantası ile MBO ile kıyaslandığında, MBO algoritmasının çözüm üretme performansının bir miktar önde olduğu ifade edilebilir.

Sonuç olarak karesel atama problemleri için geliştirilen MBO algoritmasının, sırt çantası problemlerinin çözümünde yukarıda kıyaslanan diğer algoritmalardan daha başarılı olduğu görülmektedir. Bu sonuçlara göre MBO algoritması, 0-1 sırt çantası problemlerinin çözümü için başvurulabilecek önemli bir yöntem olarak öne çıkmaktadır.