DOKUZ EYLÜL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BULANIK ATAMA PROBLEMİNDE OWA YIĞIŞIM OPERATÖRÜNÜN KULLANIMI VE UYGULAMASI

Çağla Efsun COŞKUN

Ekim, 2019
İZMİR

BULANIK ATAMA PROBLEMİNDE OWA YIĞIŞIM OPERATÖRÜNÜN KULLANIMI VE UYGULAMASI

Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü
Yüksek Lisans Tezi
Bilgisayar Bilimleri Anabilim Dalı

Çağla Efsun COŞKUN

Ekim, 2019
İZMİR

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

ÇAĞLA EFSUN COŞKUN, tarafından **DR.ÖĞR.ÜYESİ RESMİYE NASİBOĞLU** yönetiminde hazırlanan **"BULANIK ATAMA PROBLEMİNDE OWA YIĞIŞIM OPERATÖRÜNÜN KULLANIMI VE UYGULAMASI"** başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

.....

Dr.Öğr.Üyesi Resmiye NASİBOĞLU

Danışman

.....

Doç.Dr. Fidan NURİYEVA

Jüri Üyesi

.....

Dr.Öğr.Üy. Sinem PEKER

Jüri Üyesi

Prof.Dr. Kadriye ERTEKİN
Müdür
Fen Bilimleri Enstitüsü

TEŞEKKÜR

Bana bu yüksek lisans tezi konusunda çalışma olanağı tanıyan tez danışmanım Dr. Resmiye Nasiboğlu'na çok teşekkür ederim.

Tez çalışmalarıma bilgisi ile ışık olan ve sabırla beni yönlendiren Prof. Dr. Efendi Nasiboğlu'na ve yardımlarını esirgemeyen tüm bölüm hocalarıma teşekkürü bir borç bilirim.

Lisansüstü eğitim hayatımda desteklerini bolca hissetiğim bölüm arkadaşlarım Ayşe ve Mehmet'e teşekkür ederim.

Tüm öğretim hayatım boyunca desteklerini hissettiğim arkadaşlarıma, beni yüreklendiren sevgili anneme ve babama sonsuz minnetle...

Çağla Efsun COŞKUN

BULANIK ATAMA PROBLEMİNDE OWA YIĞIŞIM OPERATÖRÜNÜN KULLANIMI VE UYGULAMASI

ÖZ

Klasik mantık ikili değerlerden oluşmaktadır. Kesin ve net değerler, bazen ifade edilmek istenen durumu karşılamayabilir. Görece kıyaslama ile ifade edilmek istenen durumlar ikili değerlerin dışında kalabilir. Benzerlik ve belirsizlik içeren bazı kavramlar için bulanık mantık kullanımı, durumun anlaşılması ve kavranması için daha uygun olacaktır. Matematiksel anlamda bulanık olma durumu, çözüm kümesinin sıfır ve bir elemanlarından değil, sıfır ve bir kapalı aralığında kalan elemanlardan oluşabilmesi durumudur. Bu sonsuz çözüm kümesi; problemin sonucunun, ortaya çıkardığı durumun anlaşılmasına esneklik kazandırmaktadır. Bu tez içerisinde OWA yığışım operatörünün belirli bir algoritma ile elde edildikten sonra bulanık atama problemlerindeki kullanımı incelenmiştir. OWA ağırlıklarının dağılımını yansıtan stres fonksiyonu belli kalıplar doğrultusunda parçalı doğrusal fonksiyon aracılığıyla yakınsanılmıştır. Bu amaçla yapay arı kolonisi algoritması kullanılmıştır. Ayrıca parçalı doğrusal fonksiyonun oluşturulmasında kullanılan doğru parçalarının sayısı önceden parametre şeklinde verilmeksizin, algoritma aracılığıyla otomatik olarak bulunabilmektedir. Diğer operatörlere kıyasla bulanık atama problemlerinde OWA yığışım operatörünün daha uygun olduğu görülmüştür. Bu durumun görsel ifadesi için C# programlama dili ile yazılım geliştirilmiştir.

Anahtar kelimeler: Bulanık mantık, bulanık atama, sıralı ağırlıklı ortalama operatörü, OWA, stres fonksiyonu, yapay arı kolonisi algoritması

USING AND APPLICATION OF OWA AGGREGATION OPERATOR IN FUZZY ASSIGNMENT PROBLEM

ABSTRACT

Classical logic consists of binary values. The precise values can not sometimes correspond the conditions required to be stated. Cases that can be expressed by relative comparison go out of view the binary values. For some concepts of similarity and uncertainty, the use of fuzzy logic would be more appropriate for understanding and absorbing the situation. Mathematically, being fuzzy is has solution set which includes one and zero bounded interval not only one and zero. This infinity solution set gives the problem a flexibility. In this thesis, the use of the OWA aggregation operator in fuzzy assignment problems after a certain algorithm has been studied. The stress function, which reflects the distribution of the OWA weights, is converged through the segmented linear function in the direction of certain patterns. For this purpose, artificial bee colony algorithm was used. In addition, the number of straight segments used in the creation of the segmented linear function can be automatically detected by the algorithm without prior parameterization. It has been found that OWA aggregation operator is more suitable for fuzzy assignment problems compared to other operators. For the visual expression of this problem, the software has developed with C# programming language.

Keywords: Fuzzy logic, fuzzy assignment, ordered weighted averaging operator, OWA, stress function, artificial bee colony algorithm

İÇİNDEKİLER

Sayfa

ŞEKİLLER LİSTESİ

Sayfa

BÖLÜM BİR
GİRİŞ

Belirsizlik konusu, yıllardır egemen olan geleneksel görüş karşısında çok ilgilenilen, merak edilen ve çaba harcanan bir konu olamamıştır. Klasik mantığa uymayan durumlar, olaylar incelenememiş ve üzerinde çalışılamamıştır. Bu durum 19. yüzyılın sonlarına doğru fizikçiler tarafından fark edilerek istatiksel mekanik ile ilişkili yeni metotlar geliştirilmesini sağlamıştır (Ross, 2010). Belirsizliğe yer vermeyen Newton mekaniği ve altındaki hesaplamalar olasılık teorisi ile tanımlanan istatiksel mekanik ile yer değiştirmiş ve kesin olmayan problemler üzerine son yıllarda yapılan çalışmalar ile modeller daha güvenilir ve sağlam hale getirilmeye çalışılmıştır (Ross, 2010).

Olasılık teorisi, 19. yüzyılın sonlarından 20. yüzyılın başlarına kadar belirsizliğin nicel hale getirilmesinde kullanılmıştır. 1937 yılında Max Black tarafından olasılık teorisi kullanılarak, 1965 yılında ise Lütfi Aliasker Zadeh'nin bulanık kümeler ile ilgili çalışmalarıyla belirsizliğin aşamalı şekilde evrimi gerçekleşmiştir. Zadeh'nin çalışması olasılık teorisinin dayandığını ikili mantık temeline meydan okuyarak belirsizlik üzerinde güçlü ve derin bir etki bırakmış bulanık mantık teorisinin temellerini atmıştır. Zadeh, çalışmasında Lukasiewicz'in dereceli üyelik ile tanımlanmış kümelerini ve klasik mantığı harmanlayarak matematiğin, dil ve insan zekasını bağlayıcı özellikte olabileceği sezmiştir. Bazı konularda matematikten ziyade kelimelerin daha iyi tanımlayıcı olması nedeniyle gerçeğin bir modeli bulanık mantık ile daha iyi oluşturulabilmektedir (McNeill ve Thro, 1994).

1.1 Problemin Tanımı

Belirsizlik konusunun yanısıra gerek günlük yaşamımızda gerek bilim dünyasında ya da iş dünyasında karar verme ve bununla ilgili olarak atama problemleri önemli bir yere sahiptir. Atama problemleri belirli bir sayıda nesnenin (örneğin iş, öğrenci) belirli bir sayıda nesneye (örneğin işçi, görev) nasıl atanacağı (Amaç; minimum

maliyet, maksimum kazanç olarak belirlenebilir) sorusu ve cevabıyla ilgilenir. Matematiksel olarak sonlu sayıda eleman içeren kümeler arasında fonksiyon tanımlanarak atama problemleri ele alınabilir. Gerçek yaşamın hassasiyet ve belirsizliği, ilgilenilen problemin kısıtlarının, tanımlarının ve parametrelerinin tam sayı olmaması durumuna yol açabildiğinden, karar sisteminin bulanık küme üzerinde olması problemin çözümünde etkili bir durumdur (Zadeh, 2008).

1.2 Probleme Yaklaşım ve Amaç

Bu tez kapsamında bulanık mantık ile ele alınan atama problemlerinde karar vermeyi sağlayan yapı olarak sıralı ağırlıklı ortalama operatörü kullanılacaktır. Bunun sebebi ise diğer yığışım operatörlerine göre sıralı ağırlıklı ortalama operatörü daha kullanışlı, esnek ve etkilidir. Klasik mantık sınırları ve arasında kalan tüm durumlar için bir ortalama değeri sağlayarak karar vermede ideal olanı elde etmek için en iyi operatör sıralı ağırlıklı ortalama operatörüdür (McNeill ve Thro, 1994). Stres fonksiyonunun ağırlıkları OWA yığışım operatörü ile belirlenecek, bu ağırlıkların bulunması için ise yapay arı koloni algoritmasından yararlanılacaktır. Böylelikle kullanıcının çok değerli girdileri ifade edilelebilecektir.

BÖLÜM İKİ
KLASİK MANTIK VE BULANIK MANTIK

Yunanca logike (λογική) kelimesinin arapça tercümesi olan ‘mantık ’, kelime anlamı ile söze ve akla (ya da akıl yürütmeye) ait demektir. Mantık kelimesi, hem bir bilim olarak hem de düşünme tarzını belirtmek için kullanılmaktadır. Mantık bilimi, doğru düşünmenin düzenli tespitinden ibarettir yani bilgi yapısını inceler ve doğru ile yanlış arasındaki akıl yürütmenin ayrımını yapar (Öner, 1974).

Mantık biliminin kurucusu Aristoteles (mö 4. yy)’dir. Aristo’nun çalışmaları uzun yıllar mantık biliminde hüküm sürmüştür. Daha sonra Farabî, İbni Sina mantık alanında bir çok eser vermiştir. Ortaçağdan rönesansa kadar Aristo’nun mantık düşüncesi hakimdir. Ortaçağın mantık alanında öncü isimleri Albert Magnus, Thomas Aquinas ve XXI. İoannes’dir. Aristo mantığı ortaçağda bir metot olarak yeterli iken tabiat bilimlerinin gelişme gösterdiği rönenans döneminde Aristo mantığı metot olarak yetersiz kalmıştır. Bacon ve Descartes Aristo mantığının temelini oluşturan kıyasa karşı koyarak yetersiz bulmuşlar ve yeni metotlar aramışlardır. Daha sonra sembolik mantık, mantığı formel hale getirme ile ilgilenilmiştir. Klasik mantık felsefenin bir kolu olmaktan çıkarak daha çok matematikçilerin ve fizikçilerin ilgilendiği alan olmuştur.

2.1 Klasik Mantık

Tarihsel düzende matematik ve mantık birbirinden farklı disiplinler olarak ortaya çıkmış olsa da modern çağlarda birbirleri ile kopmaz bir bütün olmuşlardır. Zamanla matematik daha mantıksal ve mantık daha matematiksel hale gelmiştir. Klasik mantık ikili değer sistemi ile kesin durumlar ortaya koyar.

Nesnelerin ya da elemanların oluşturduğu (birbirlerinden farklı ve iyi tanımlı olmak koşulu ile) bir koleksiyona ya da gruba *küme* denir. Bir kümenin objeleri o kümenin elemanı ya da üyesi olarak adlandırılır yani objeler kümeye aittir. Objeler kümeye ait

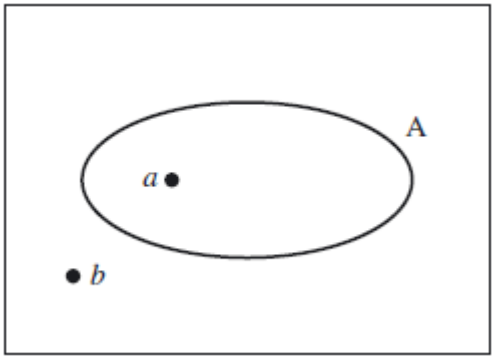
değilse kümenin üyesi değildir, küme dışındadır.

Klasik kümeler kesin, net ve kuşkuya yer vermeyen sınırlamalar ile ifade edilir; tanımında ya da küme sınırlarında bir belirsizlik söz konusu değildir. E kümesi evrensel bir küme ve A kümesi de E evrensel kümesinin bir alt kümesi olmak üzere $A \subset E$ şeklinde ifade edilir. A kümesine ait ve ait olmayan elemanlar sırasıyla $a_1 \in A$ ve $a_2 \notin A$ şeklinde ifade edilir ($a_1 \in E, a_2 \in E$).

Kümenin elemanı ile kümenin elemanı olmayan arasında belirgin bir ayrım vardır. Diğer bir ifade ile a bir nesne ve $\mu(a)$ üyelik derecesini ifade eden bir fonksiyon olmak üzere küme tanımı matematiksel olarak şu şekilde ifade edilir.

$$\mu_X(a) = \begin{cases} 0 & a \notin X \\ 1 & a \in X \end{cases} \tag{2.1}$$

$\mu_X(a)$ fonksiyonu $\{0, 1\}$ değerlerinden birini alabilmektedir. Evet/Hayır, Siyah/Beyaz, 0/1 kadar net bir sonucu vardır. Şekil 2.1'de görüldüğü üzere A kümesinin sınırları kesin ve nettir; a nesnesi A kümesinin elemanı iken b nesnesi küme sınırları dışındadır. Matematiksel olarak $a \in A$ ve $b \notin A$ olarak ifade edilir.



Şekil 2.1 Klasik küme

Klasik kümelere örnek olarak tam sayılar kümesi $\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$, doğal sayılar kümesi $\mathbb{N} = \{1, 2, 3, ...\}$, negatif olmayan tamsayılar kümesi $\mathbb{N}_0 = \{0, 1, 2, 3, ...\}$ ve reel sayılar ($\mathbb{R}$) verilebilir.

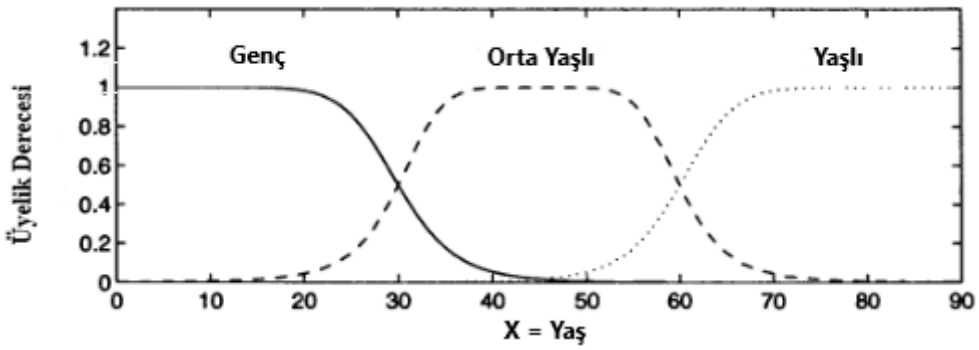
2.2 Bulanık Mantık

Uzun yıllar belirsizlik ve bununla ilgili problemler üzerinde düşünülmüş, klasik mantığın bu tür problemlere ve kavramlara getirebildiği çözümler, yorumlar felsefeciler tarafından tartışılmıştır. Bulanık Mantık (İspanyolca'dan 'Borrosa Mantığı'(Lógica Borrosa), yayılan mantık (Lógica Difusa) ya da sezgisel mantık (Lógica Heurística) olarak çevrilebilir) baypas değerli (çok-değerli mantık: Multi Valued Logic, MVL) mantık olarak düşünülebilir (Garrido, 2012). Bulanık Mantık, Bulanık Kümeler teorisi ile yakından alakalıdır. Bulanık Mantık yakın zamanda sahneye çıkmış gibi dursa da aslında Platon (Eflatun) zamanına kadar uzanan bir geçmişi bulunmaktadır.

2.2.1 Bulanık Mantık Teorisi

Bulanık Mantık hakkında en önemli gelişim noktası 1965 yılında Zadeh tarafından ifade edilmiştir. Aslında yaklaşık 30 yıl öncesinde Max Black adlı felsefeci belirsizlik konusu ile ilgili temeller atmıştır. Zadeh'in çalışması bu alanda bir kilometre taşıdır. Zadeh, bu çalışmasında nesnelerin sürekli üyelik derecelerinin sınıfını bulanık küme olarak tanımlamıştır (Zadeh, 1965). Zadeh, çalışmasında kümenin bir üyelik fonksiyonuna ve kümenin her bir nesnesinin 0 ve 1 arasında değer alan üyelik derecesi olduğunu öne sürmüştür.

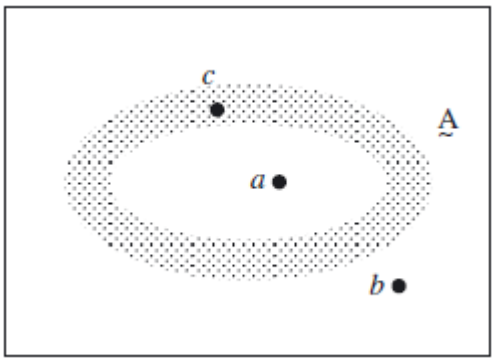
Günlük olarak dilimizde kullandığımız uzun, orta, kısa, güzel, çirkin, sıcak ve soğuk gibi sıfatların aslında kişiden kişiye değişen birer dereceleri vardır. Örneğin aşağıda Şekil 2.2 grafiğinde yaş aralıklarına göre yaşlı, orta yaşlı ve genç tanımlaması yapılmıştır.



Şekil 2.2 Dilsel değerlere ait üyelik fonksiyonu

2.2.2 Bulanık Küme

Bulanık kümelerde sınırlar net değildir, belirsizlik söz konusudur. Belirsiz özellikler ve tanımlar ile ifade edildiğinden sınırlar bulanıktır. Üyelik fonksiyonu, $\mu_x(a)$, klasik kümelerdeki gibi kesin olarak 0 ya da 1 değerini almaz; $[0, 1]$ aralığında bir değer alır. Herhangi bir elemanın, üyelik fonksiyonun değeri 0'a yakın ise kümeye düşük bir dereceyle; 1'e yakın ise kümeye yüksek bir dereceyle aitliği söz konusudur. Şekil 2.3'de a, b, c elemanlarının A kümesine üyelik dereceleri, tanımlanan üyelik fonksiyonuna göre sırasıyla $\{1, 0, 0.5\}$ değerlerini alabilir.



Şekil 2.3 Bulanık küme

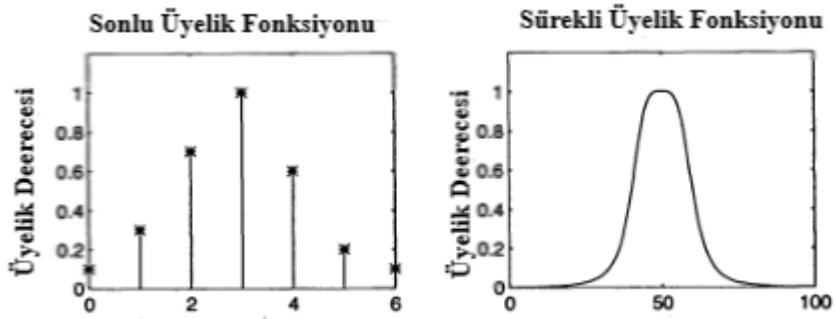
Zadeh'in, bulanık kümeler için tanımladığı üyelik derecesinin değeri reel, sürekli, birim aralıktatır ($\mu_A(x) \in [0, 1]$). μ değerinin 0 ve 1 olması sırasıyla üyeliğin olmadığı ve tam bir üyeliğin olduğu sonucunu verir. Birim aralıkta sonsuz değer bulunmaktadır bu sebeple evrensel kümedeki bir eleman için bir çok üyelik derecesi değeri ve tanımı

olabilir.

E evrensel kümesi içerisinde A bulanık kümesinin tanımı, E kümesine bağlı olarak aşağıdaki gibi yapılabilir:

$$A = \begin{cases} \sum_{x_i \in E} \mu_A(x_i)/x_i, & \text{E kümesi sonlu sayıda elemana sahip ise,} \\ \int_E \mu_A(x)/x, & \text{E evrensel kümesi sürekli ise.} \end{cases} \tag{2.2}$$

Şekil 2.4'deki sonlu ve sürekli kümeler üzerinde üyelik fonksiyonları tanımlanmıştır.



Şekil 2.4 Sonlu ve sürekli üyelik fonksiyonu

2.2.3 Tanımlar ve Terminoloji

2.2.3.1 Üyelik Fonksiyonu (Membership Function)

X, x ile ifade edilen nesnelerin bir koleksiyonu olmak üzere, bulanık küme A aşağıdaki sıralı ikililer olarak ifade edilebilir.

$$A = \{(x, \mu_A(x)) | x \in X\} \tag{2.3}$$

$\mu_A(x)$ A bulanık kümesisinin üyelik fonksiyonu olarak ifade edilir. Her X kümesine ait elemanların üyelik derereceleri, üyelik fonksiyonu ile ifade edilir ve $[0, 1]$ aralığında değer alabilir (Jang ve diğer., 1997). Matematiksel olarak Şekil 2.3'deki A kümesi, 2.3

ile ifade edilen bulanık küme tanımı ile aşağıdaki şekilde gösterilebilir.

$$A = \left\{ \frac{a}{1}, \frac{b}{0}, \frac{c}{0,5} \right\}$$

2.2.3.2 Dayanak (Support) ve Çekirdek (Core)

Dayanak (Support): Bulanık bir A kümesinin dayanağı, üyelik fonksiyon değeri 0'dan büyük olan tüm noktaların oluşturduğu kümedir.

$$s(A) = \{x | \mu_A(x) > 0\} \tag{2.4}$$

Çekirdek (Core): Bulanık bir A kümesinin çekirdeği, üyelik fonksiyon değeri 1'e eşit olan tüm noktaların oluşturduğu kümedir.

$$c(A) = \{x | \mu_A(x) = 1\} \tag{2.5}$$

2.2.3.3 Normallik (Normality)

Bulanık bir A kümesinin çekirdek (core) kümesi boş değil ise bu bulanık küme normaldir. Bulanık bir küme normal ise $x \in A$ olmak üzere $\mu_A(x) = 1$ eşitliğini sağlayacak bir x noktası her zaman bulunur.

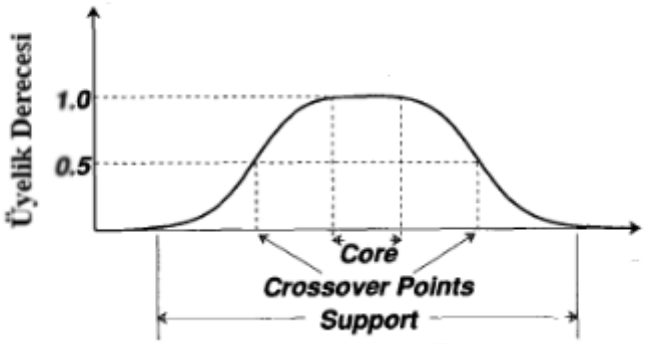
2.2.3.4 Köprü Noktaları (Crossover Points)

A bulanık kümesinde $x \in A$ olmak üzere $\mu_A(x) = 0,5$ eşitliğini sağlayacak x noktası köprü noktasıdır.

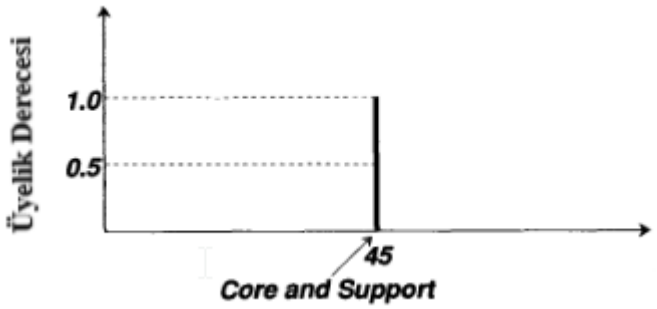
$$cp(A) = \{x | \mu_A(x) = 0,5\} \tag{2.6}$$

2.2.3.5 Tekil Bulanık (Fuzzy Singleton)

Bir A bulanık kümesinin tek bir dayanak noktası ($\mu_A(x) = 1$) var ise bu küme tekil bulanıktır.



Şekil 2.5 Çekirdek, köprü ve dayanak gösterimi



Şekil 2.6 Çekirdek ve dayanak gösterimi (tekil bulanık)

2.2.3.6 α-Kesim (α-Cut)

Bulanık bir A kümesinin α kesim ya da α seviye kesim kümesi kesin bir küme olup aşağıdaki şekilde tanımlanır.

$$A_\alpha = \{x|\mu_A(x) \geq \alpha\} \tag{2.7}$$

Kesin α-kesim (Strong α-cut) kümesi ise aşağıdaki gibi tanımlanabilir.

$$A_\alpha = \{x|\mu_A(x) > \alpha\} \tag{2.8}$$

Yukarıdaki tanım ve notasyonlar yardımı ile bulanık bir A kümesinin dayanak (support) ve çekirdeği (core) aşağıdaki şekilde ifade edilebilir.

$$s(A) = A_0 \tag{2.9}$$

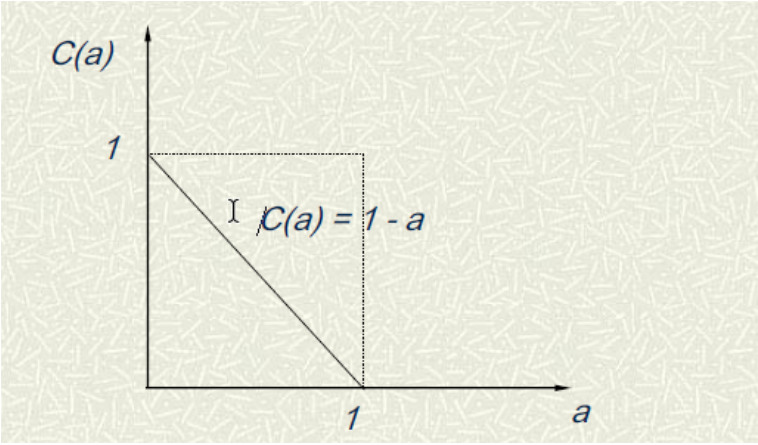
ve

$$c(A) = A_1 \tag{2.10}$$

2.2.3.7 Bulanık Değili

C Bulanık değil operatörü sürekli bir fonksiyon $C : [0,1] \rightarrow [0,1]$ olmak üzere $C(0) = 1$ ve $C(1) = 0$ ve $C(a) \geqslant C(b), \quad a \leqslant b$ koşulları ile aşağıdaki şekilde ifade edilir.

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \tag{2.11}$$



Şekil 2.7 Bulanık değil fonksiyon gösterimi

2.2.3.8 T-Norm ve T-Conorm

A ve B iki bulanık küme olmak üzere kesişimleri $T : [0,1]$ x $[0,1] \rightarrow [0,1]$ fonksiyonu ile ifade edilirse üyelik dereceleri aşağıdaki şekilde ifade edilebilir.

$$\mu_{A\cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \quad \overline{*} \quad \mu_B(x) \tag{2.12}$$

$\overline{*}$, T fonksiyonu için kullanılan operatördür. Bu fonksiyon genel olarak T-norm (Triangular norm) operatörü olarak adlandırılır.

T-norm operatörü sırasıyla kapalılık, monotonluk, değişme ve dağılma özelliklerine sahiptir.

$$T(0,0) = 0, \qquad T(a,1) = T(1,a) = a \tag{2.13}$$

$$T(a,b) \leq T(c,d), \qquad a \leq c \quad ve \quad c \leq d \tag{2.14}$$

$$T(a,b) = T(b,a) \tag{2.15}$$

$$T(a,T(b,c)) = T(T(a,b),c) \tag{2.16}$$

2.2.4 Bulanık Küme Operasyonları

A ve B iki bulanık küme olmak üzere;

A ve B kümelerinin birleşim kümesi ve A ve B kümelerinin kesişim kümesi aşağıdaki şekilde ifade edilmektedir.

$$\mu_{A\cup B} = Max[\mu_A(x), \mu_B(x)] \tag{2.17}$$

$$\mu_{A\cap B} = Min[\mu_A(x), \mu_B(x)] \tag{2.18}$$

Birleşim ve kesişime ait bazı özellikler aşağıdaki gibidir.

- Bulanık kümenin değilin değili yine kümenin kendisini ifade eder.

$$\overline{(\overline{A})} = A \tag{2.19}$$

- Bulanık kümelerdeki işlemlerde değişme özelliği vardır.

$$A \cup B = B \cup A \tag{2.20}$$

$$A \cap B = B \cap A \tag{2.21}$$

- Bulanık kümelerdeki işlemlerde birleşme özelliği vardır.

$$(A \cup B) \cup C = A \cup (B \cup C) \tag{2.22}$$

$$(A \cap B) \cap C = A \cup (B \cap C) \tag{2.23}$$

- Bulanık kümelerdeki işlemlerde dağılma özelliği vardır.

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \tag{2.24}$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C) \tag{2.25}$$

Diğer küme özellikleri de aşağıdaki gibidir.

$$A \cup A = A \tag{2.26}$$

$$A \cap A = A \tag{2.27}$$

$$A \cup (A \cap A) = A \tag{2.28}$$

$$A \cap (A \cup A) = A \tag{2.29}$$

$$A \cup X = X \tag{2.30}$$

$$A \cap \emptyset = \emptyset \tag{2.31}$$

$$A \cup \emptyset = A \tag{2.32}$$

$$A \cap X = A \tag{2.33}$$

$$\overline{A \cap B} = \bar{A} \cup \bar{B} \tag{2.34}$$

$$\overline{A \cup B} = \bar{A} \cap \bar{B} \tag{2.35}$$

$$(\bar{A} \cup B) \cap (A \cup \bar{B}) = (\bar{A} \cap \bar{A}) \cup (A \cap B) \tag{2.36}$$

$$(\bar{A} \cap B) \cup (A \cap \bar{B}) = (\bar{A} \cup \bar{A}) \cap (A \cup B) \tag{2.37}$$

BÖLÜM ÜÇ
ATAMA PROBLEMLERİ

3.1 Klasik Atama Problemleri ve Algoritmaları

Bir sistemdeki kaynakların en iyi şekilde kullanılması, belirli bir amaca en iyi şekilde ulaşmak bir optimizasyondur. Atama problemi ulaşılmak istenen hedefe ait kullanılan yolu optimize etmek için çözüm bulmaya çalışan doğrusal bir programlama problemidir. Genel anlamda toplam maliyeti en aza indirmek ya da toplam karı en üst seviyeye çıkarmak için eşit sayıdaki kaynakların eşit sayıdaki hedeflere atandığı bir problem türüdür.

Bir görevlendirme işlemi (belirli bir sayıdaki işin işçilere dağıtımı), kaynakların hedeflere aktarılması gibi sunulan problemler atama problemleridir. Bu problemlerde kaynak (işçi) ve hedef (iş) sayıları birebir eşleşmelidir. Birebir eşleme olmadığı durumlarda yani kaynak hedef sayılarının uyuşmadığı durumlarda problemin çözümü için sanal bir kaynaktan ya da hedeften yararlanılabilir.

Kaynaklar aşağıdaki gibi olabilir.

- İşgücü
- Anapara
- Zaman
- Hammadde
- Kapasite

Atama probleminin amacına göre ilgili yöntem belirlenir. Amaçlar problem içerisindeki istenen değerin minimize yada maksimize edilmesi şeklindedir. Amaçlar; maliyet minimizasyonu, kar maksimizasyonu, kapasite kullanımı, verimliliğin maksimizasyonu şeklinde olabilir.

c_{ij} , i. kaynak j. hedefe atanmasındaki toplam maliyeti ifade etmek üzere amaç fonksiyonu 3.1 ile edilebilir.

$$min \sum_{i=1}^{m} \sum_{j=1}^{m} c_{ij} x_{ij} \tag{3.1}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \quad \forall j = 1, 2, \ldots, m$$

$$\sum_{j=1}^{m} x_{ij} = 1 \quad \forall i = 1, 2, \ldots, m$$

$$x_{ij} = \begin{cases} 1 & i. \text{ kaynak } j. \text{ hedefe atandı ise,} \\ 0 & \text{diğer durumlar.} \end{cases}$$

$\forall i, j = 1, 2, \ldots, m$

mxm matris i. kaynağın j. hedefe atanmasında x_{ij} atama değerini ifade edecek olursa maliyet matrisi mxm olmak üzere aşağıdaki gibi ifade edilebilir.

$$C = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \ldots & x_{1m} \\ x_{11} & x_{12} & x_{13} & \ldots & x_{1m} \\ & & \vdots & & \\ x_{m1} & x_{m2} & x_{m3} & \ldots & x_{mm} \end{bmatrix} \tag{3.2}$$

Atama problemlerine çözüm bulabilmek için bir çok algoritma mevcuttur.

- Macar algoritması
- Simpleks algoritması
- Genetik algoritma

3.2 ile gösterilen matriste değerler sadece 0 veya 1 olabilir. Fakat bu değerlerin $[0, 1]$

aralığında sürekli olabilmesi durumları bulanık atama problemine uygundur.

3.2 Bulanık Atama Problemleri ve Algoritmaları

Klasik atama problemlerinde de bahsedildiği gibi işçilerin işlere atanması, işlerin makinelere paylaştırılması, nesnelerin kutulara yerleştirilmesi gibi problemler birer atama problemidir. Son zamanlarda ise klasik atama problemlerinden çok bulanık atama problemleri önem kazanmıştır.

Atama problemlerinin bulanık mantık yaklaşımıyla ele alınmış bir çok çalışması mevcuttur. İşlerin kaliteli bir biçimde işçilere paylaştırılması problemi (Nasibov ve Kınay, 2006), maliyetin kesin olmadığı durumlarda atamanın etiketleme algoritması (labeling algorithm) ile bulunması (Lin ve Wen, 2004) gibi çalışmalar yapılmıştır.

3.2.1 Bulanık Atama Problemleri

3.2.1.1 Kaliteli İş Paylaşım Problemi

İşlerin işçilere atanması problemini ele alalım. Klasik atama problemlerinde olduğu gibi birebir eşleme olmaksızın (kaynak sayısı, hedef sayısına eşit olmayabilir) n sayıdaki işin $(w_1, w_2, \ldots, w_n)$ m sayıdaki işçilere $(s_1, s_2, \ldots, s_m)$ atanması ile ilgili kısıtlar aşağıdaki gibidir. s_{m+1} ise kiralık işçi grubunu oluştursun.

Her işin bir işçi tarafından yapılabilmesi için $a_j > 0$ olmak üzere bir a_j zamanı gerekmektedir. İşçi ve işler arasındaki bulanık ilişkileri temsil edebilmek için bulanık ilişki matrisi kullanılmaktadır ($C = ||c_{ij}||$). Bu matris $i.$ işçinin $j.$ işi yapabilme yeteneğini gösterir.

$$c_{ij} = \begin{cases} 1 & \text{işçi, iş konusunda yetenekli} \\ 0 & \text{işçi, iş konusunda yeteneksiz} \\ 0 < c_{ij} < 1 & \text{diğer durumlar} \end{cases} \tag{3.3}$$

Her işçinin iş yapabilme kapasitesi, işçinin yetenekli olduğu işi yapabilmesi gibi koşullar ile atama kalitesinin maksimum seviyeye getirilmesi gerekmektedir.

İşlerin işçilere atama matrisi $X = \left|X_{ij}\right|$, $i = 1, 2, \ldots, m+1$ ve $j = 1, 2, \ldots, n$ olmak üzere matris değerleri sadece 1 ve 0 değerlerini alabilsin.

$$x_{ij} = \begin{cases} 1 & j. \text{ iş } i. \text{ işçiye atandı ise,} \\ 0 & \text{diğer durumlar.} \end{cases} \tag{3.4}$$

3.2.1.2 Kutu Paketleme Problemi

Nesnelerin kutulara (konteynerlere), kutularda en az boşluk kalacak şekilde yerleştirilmesi problemidir. $X = x_1, x_2, \ldots x_n$ nesneler, $S = S_1, S_2, \ldots S_m$ ise konteynerler ve S_{m+1} kiralık konteyner olsun.

Nesneler ve konteynerler arasındaki bulanık ilişkiler aşağıdaki gibi ifade edilebilir.

x_i ve x_j nesnelerinin bir arada taşınması; gerekliliği R_1 matrisi, uyumluluğu R_2 matrisi ile ifade edilsin;

$$R_1 = ||R_1(x_i, x_j)|| \tag{3.5}$$

$$R_2 = ||R_2(x_i, x_j)|| \tag{3.6}$$

x_i nesnesinin S_j konteynerında taşınması; gerekliliği R_3 matrisi, uyumluluğu ise R_4 matrisi ile ifade edilsin.

$$R_3 = ||R_3(x_i, S_j)|| \tag{3.7}$$

$$R_4 = ||R_4(x_i, S_j)|| \tag{3.8}$$

Taşımanın kalite derecesi belirtilen kısıtlar ile ($j = 1, \ldots, m$) aşağıdaki gibi ifade edilebilir.

$$A = min_{j=1,2,\ldots m} min\ K_1(S_j), K_2(S_j), K_3(S_j), K_4(S_j) \tag{3.9}$$

$$K_1(S_j) = 1 - \{max\ R_1(x^1, x^2) | x^1 \in S_j, x^2 \notin S_j\} \tag{3.10}$$

$$K_2(S_j) = min\{\ R_2(x^1, x^2) | x^1 \in S_j, x^2 \in S_j\} \tag{3.11}$$

$$K_3(S_j) = 1 - \{max\ R_3(x, S_j) | x \notin S_j\} \tag{3.12}$$

$$K_4(S_j) = min\{\ R_4(x, S_j) | x \in S_j\} \tag{3.13}$$

Yukarıda $K_i(S_j), \quad i = 1, \ldots, m \quad j = 1, \ldots, m$ değerlerinin hesaplanmasında basit minimum ve maksimum yığışım operatörleri kullanılmıştır. Fakat bu durum daha genel olan OWA yığışım operatörü kullanılarak $j = 1, \ldots, m$ olmak üzere aşağıdaki gibi ifade edilebilir.

$$K_1(S_j) = 1 - OWA\{R_1(x^1, x^2) | x^1 \in S_j, x^2 \notin S_j\} \tag{3.14}$$

$$K_2(S_j) = OWA\{R_2(x^1, x^2) | x^1 \in S_j, x^2 \in S_j\} \tag{3.15}$$

$$K_3(S_j) = 1 - OWA\{R_3(x, S_j) | x \notin S_j\} \tag{3.16}$$

$$K_4(S_j) = OWA\{R_4(x, S_j) | x \in S_j\} \tag{3.17}$$

3.2.2 Bulanık Atama Algoritmaları

Kutu paketleme probleminin çözümü için Nasibov (2004) çalışmasında aşağıdaki algoritmayı önermiştir.

Nesnelerin konteynerlera dağıtılması algoritması

Adım 1.

R_1 komşuluğundaki değerlerin tekrarlamalı olarak bulunması.

${R_1}^1 = R_1\ , \quad {R_1}^k = {R_1}^{k-1} \circ R_1, \quad k = 2, 3, \ldots, n-1.$

$\circ$ matrislerin minmax çarpımını ifade eder.

Adım 2.

g eşik değeri aşağıdaki koşullar ile sağlanması.

$g \leq min\{g_1, g_2\}$

$g_1 = min_{i,j=1,2,...,n} max\{1 - R_1(x_i, x_j), R_2(x_i, x_j)\}$

$g_2 = min_{\{} i = 1, 2, ..., n \quad j = i = 1, 2, ..., m\} max\{1 - R_3(x_i, S_j), R_4(x_i, S_j)\}$

Adım 3.

g-gruplarının oluşturulması.

$\hat{R_1}$ ilişkisi, g-grupları olarak adlandırılan X kümesinin ayrımlarında kesişmeyen denklik sınıfları ile ifade edilir.

$y_1, y_2, \ldots, y_l \quad l \leq k.$

Her denklik sınıfı aşağıdaki koşulu da sağlamalıdır.

$y_i, i = 1, 2, \ldots, l$ ve

$x^1, x^2 \in y_i, \quad x^3 \notin y_i$ olduğunda $\hat{R}_1(x^1, x^2) \geq g, \quad \hat{R}_1(x^1, x^3) \leq g$ durumu da sağlanmaktadır.

Adım 4.

$R_2{}', R_3{}'$ve$R_4{}'$ilişki matrisleri aşağıdaki gibi ifade edilir ise;

$R_2{}'(y_i, x_j) = min\{R_2(x^1, x^2) || x^1 \in y_i, x^2 \in y_i\};$

$R_3{}'(y_i, S_j) = max\{R_3(x^1, x^2) || x^1 \in y_i, x^2 \in S_j\};$

$R_4{}'(y_i, S_j) = min\{R_4(x^1, x^2) || x^1 \in y_i, x^2 \in S_j\};$ her grup toplam içerdiği nesneyi ifade edebilir.

Adım 5.

$\hat{R_3}$ ilişkisinin koşullarına göre grupların belirlenmesi ile; $i \leq l$ *olduğu sürece aşağıdaki adımlar tekrarlanır*

Adım 6.

$j \leq m$ *olduğu sürece aşağıdaki adımlar tekrarlanır*

Adım 7.

if $\hat{R}_3(y_i, S_j) \geq 1 - g$ **then**

else if $(\hat{R}_4(y_i, S_j) < g)$ or $(minR_2(y_i, y)|y \in S_j < g)$ or (y_i, S_j içerisinde yer almıyorsa) **then**

problemin g değeri için bir çözümü yoktur. g azaltılması ile *Adım 2*'ye git.

else

y_i, S_j içeisinde yer alsın

end if

Adım 8.

i = 1'den başlasın.

Adım 9.

if y_i henüz yerleştirilmedi ise **then**

$j = 1$

end if

Adım 10.

if $(R_4\prime(y_i, S_j) \geq g)$ and $(min\{R_2(y_i, y)|y \in S_j \geq g)$ and (y_i), S_j içerisinde yer alabiliyor **then**

(y_i) nesnesi S_j içerisinde yer alsın

end if

if $j < m$ **then**

$j = j + 1$ ve *Adım 10*' a git.

end if

Adım 11.

$i = i + 1$

if $i < l$ **then**

Adım 9' a git.

else

Algoritmayı sonlandır.

end if

Burada kullanılan OWA operatörlerinin ağırlıkları karar vericinin stratejisine uygun

olarak ayarlanabilir.

Tezin sonraki bölümlerinde OWA operatörü hakkında bilgi verilecektir ve ağırlıkların belirlenmesinde kullanılan stress fonksiyonları ele alınacaktır. Daha sonra stres fonksiyonun hesaplanması için yapay arı kolonisi algoritması kullanılacaktır.

BÖLÜM DÖRT
SIRALI AĞIRLIKLI ORTALAMA OPERATÖRÜ

Bir konu, bir durum, bir problem hakkında elde edilen veriler işlenerek bir bilgi birikimi elde edilemiyorsa verilerin bir anlamı olmamaktadır. Bir bilgi birikimi elde edildiğinde ise bu bilgi birikiminin de işlenmesi gerekiyor. Aşağıda yer alan bazı sistem alanları için bilgi birikiminin işlenmesi oldukça önem kazanmıştır.

- Sinir Ağları (Neural Network)
- Bulanık Mantık Kontrolleri (Fuzzy Logic Controllers)
- Görüntü Sistemleri (Vision System)
- Uzman Sistemler (Expert System)
- Çok Kriterli Karar Yapıları (Multi Criteria Decision Aids)

R.R. Yager 1988 yılındaki çalışmasında sıralı ağırlıklı ortalama operatörlerine bağlı olarak yeni bir yığışım (aggregation) tekniği tanıtmıştır (Yager, 1993b). Tanıttığı bu modelin sinir ağlarında, çok kriterli karar verme alanlarındaki kullanımına dikkat çekmiş ve veritabanı sistemlerindeki kullanımı tartışılmıştır. Çoğu uygulamaları kısa zaman içerisinde karar verme, akıllı sistemler, sinir ağları, bulanık sistemler, kontrol ve iletişim ağları gibi alanlara sunulmuş ve Kacprzyk, öğrenme sistemlerinde; O'Hagan akıllı sistemlerde; Yager ve Filev bulanık akıllı sistemlerde esnek modeller geliştirmede; Rubinson ise iletişim ağlarını geliştirmede; Eklund ve Klawon ise sinirsel bulanık mantık programlarında kullanmıştır.

Yager'in bu operatörü ortaya sürmesindeki amaç, çok kriterli yığışım problemleri için bir karar fonksiyonu bulmaktı. Ortalama operatörlerinin ailesinden olan OWA operatörü her karar vericinin farklı bakış açılarına sahip olmalarından dolayı çeşitli stratejiler için oldukça verimli ve esnek çalışmaktadır.

OWA operatörü yığışımın mantıksal olarak **"ve"** veya **"veya"** arasında kalmasını sağlar. Çok kriterli karar yapılarında bir karar fonksiyonu oluşturmak üzere kullanılan OWA operatörü esnek bir yığışım operatörüdür. Bu operatörün ağırlıklarından elde edilen değer ile karar vericinin değerlendirmesinin ne kadar iyimser ya da kötümser bir tutumla gerçekleştiği saptanabilir. Aynı zamanda ağırlıklar ile karar vericinin nasıl bir tutum sergilediği görsel olarak belirlenebilir (Coşkun ve diğer., 2017).

Bir çok disiplinde karar fonksiyonu oluşturmak için kriter fonksiyonlarının yığışımı önemli bir yere sahiptir. Genel durumlardan biri tüm kriterlerin sağlanması ve diğeri ise istenilen kriterlerin sağlanmasıdır. Bu iki durum, kriter fonksiyonlarını oluşturmak için **'ve'**(and) yada **'veya'**(or) ve operatörlerini birlikte kullanmıştır. Böylelikle her karar vericinin farklı bakış açılarına sahip olmalarından dolayı çeşitli stratejiler için oldukça verimli ve esnek çalışan çok kriterli yığışım problemleri için bir karar fonksiyonu bulunmuştur (Coşkun ve diğer., 2017).

4.1 Çok Kriterli Karar Verme Yapıları

OWA operatörü *n* boyutlu bir fonksiyon olmak üzere OWA operatörünün matematiksel modeli şu şekilde ifade edilebilir;

$$f : R^n \to R$$

Bu fonksiyon *n* boyutlu, ağırlık değerlerini tanımlayacak bir vektör *W* ile ilişkilendirilirse

$$W = [w_1, w_2, ..., w_n]^T$$

öyle ki;

$$w_i \in [0, 1] \quad \sum_i w_i = 1$$

kısıtları ile fonksiyon şeklinde tanımlanır.

$$f(a_1, a_2, ..., a_n) = \sum_j w_j b_j$$

Toplam fonksiyonunda yer alan b_j değerleri $a_1, a_2, ..., a_n$ koleksiyonunda yer alan j. en büyük a_i leri temsil eder. Böylelikle fonksiyonun değeri ($f(a_1, a_2, ..., a_n)$) argümanlarının $a_1, a_2, ..., a_n$ yığışım değerine karar vermiş olur.

OWA operatörünün en önemli ve temel kısmı "sıralama" dır. a_i. argümanı belirli bir w_i ağırlığı ile ilişkili değildir. w_i ağırlığı sıralanmış argümanın *i.*nci pozisyonu ile ilgilidir.

OWA ağırlıklarını kolon vektörü olarak göstermekle düşük indekslileri üst ağırlıklarda yüksek indislileri ise aşağıda belirtmiş oluruz. Farklı OWA operatörleri farklı ağırlık fonksiyonları ile birbirinden ayrılır. Yager 3 önemli OWA yığışımına dikkat çekmiştir.

1. **F**$^*durumunda$

$$W = W^* = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix}^T$$

$F^*(a_1, a_2, \dots, a_n) = Max_i(a_i)$

2. **F**$_*$ durumunda

$$W = W_* = \begin{bmatrix} 0 & 0 & \dots & 1 \end{bmatrix}^T$$

$F_*(a_1, a_2, \dots, a_n) = Min_i(a_i)$

3. **F**$_A$ durumunda (A: Average [ortalama])

$$W = W_A = \begin{bmatrix} 1/n & 1/n & \dots & 1/n \end{bmatrix}^T$$

$F_A(a_1, a_2, \dots, a_n) = \frac{1}{n} \sum_i (a_i)$

Birçok önemli özellik OWA operatörü ile ilişkilendirilebilinir. Yager, OWA operatörü ile ilgili 2 önemli ölçü tanımlamıştır.

İlki *dağılım yada düzensizlik* (entropy / dispersion) denilen ölçüdür.

$$d(W) = \sum_i w_i \ln w_i$$

Dağılım değeri; ağırlık vektörüne bağlı yığışım değeri hesaplanırken argümanlardaki bilginin ne kadarının kullanıldığı hakkında bilgi verir. Düzensizlik değeri $[0, \ln n]$ arasında bir değer alabilir. OWA operatörünü ağırlıklı ortalama olarak kullandığımızda dağılım, yığınları eşit olarak kullandığımızın ölçüsünü verir.

İkinci ölçü ise fonksiyonun *davranışsal karakteri* (orness) ile ilgilidir.

$$o(W) = \frac{1}{n-1} \sum_{i=1}^{n} ((n-i).w_i)$$

$o(W^*) = 1, \quad o(W_*) = 0, \quad o(W_A) = 0,5.$

Davranışsal karakter değeri $[0, 1]$ aralığında değer almakta olup; $o([0, 0, \ldots 1]) = 0$ olması durumunda mantıksal **"ve"** operatörü kullanılmış ve karar verici kötümser bir yaklaşım sergilemiştir; $o([1, 0, \ldots 0]) = 1$ olması durumunda mantıksal **"veya"** operatörü kullanılmış ve karar verici iyimser bir yaklaşım sergilemiştir, $o([1/n, 1/n, \ldots 1/n]) = 1/2$ olması durumunda ortalama operatörü kullanılmış ve karar verici ortalama bir yaklaşım sergilemiştir. Ağırlık vektörünün değerine göre yığışım tipini belirleyebilmek için Orness değeri kullanılmaktadır.

Genel olarak OWA operatörü tepedeki değerleri kullanırsa *veya* operatörü gibi davranır ($o(W) \geq 0,5$). Diğer bir durum ise OWA operatörü dipteki 0 (sıfır) olmayan değerleri kullanırsa *ve* operatörü gibi davranır ($o(W) \leq 0,5$).

4.2 OWA Operatör Ailesi

OWA operatorlerinin en büyük avantajlarından biri, operatörler ile modellenebilen yığışım kurallarının seçiminde esnek bir yapıya sahip olmasıdır. Bu esnekliğin bir

bedeli vardır. Belli bir uygulamada uygun ağırlıkların seçilmesi gerekir. Bir önceki bölümde görüldüğü gibi ağırlıkların ne olması gerektiğine verilen verilerden direkt olarak karar verebiliriz.

Bu bölümde ise OWA operatörü ile alakalı ağırlıkları hesaplarken farklı bir yaklaşımdan bahsedeceğiz. Bir çok durum için bu yaklaşım oldukça basittir. OWA operatörlerinin ailesi ağırlıkları oluşturmak için bazı parametreleri kullanır.
Daha önce bahsedilen 3 OWA operatörünün W^*, W_* ve W_A sıralanması gerektiğini biliyoruz. W^* ve W_* operatörleri sırasıyla maksimum ya da minimum elemanı bulabilirken W_A operatörü herhangi bir sıralamaya gerek duymaz. Parametrelere bağlı OWA operatörü ilk olarak O'Hagan tarafından **ME-OWA** operatörü olarak tanıtılmıştır. ME'nin açılımı *Maximum Entropy*'dir.
O'Hagan OWA'nın yığışımında kullanılacak ağırlıkların şu şekilde hesaplanmasını önermiştir:

Adım 1. İstenilen orness değeri, α, seçilsin.
Adım 2. Seçilen orness değerinin maksimum düzensiz (maximum entropy) ağırlıkları hesaplansın.

Matematiksel olarak aşağıdaki problemin çözülmesi istenmektedir:

$$Max : -\sum_i w_i \ln w_i$$

öyle ki

$$\alpha = \frac{1}{n-1}\sum_i^n (n-i)w_i,$$

$$\sum_{i=1}^n w_i = 1,$$

$$w_i \in [0,1].$$

Sadece α parametresini özelleştirerek tüm ağırlıkları elde edebiliriz.
Maksimum düzensiz tekniklerin bel kemiği bu yaklaşımdır. Yager'in son

çalışmalarının ışığında ME-OWA operatörünün daha geniş hali şu şekilde ifade edilebilir:
Düzensizliğin ölçüsü $1 - Max_i[w_i]$ ifade edilirse amaç fonskiyonu şu şekilde yeniden düzenlenir:

$$Min : Max_i w_i$$

OWA ağırlıklarının başka bir ailesi Yager ve Filev tarafından tanıtılmıştır. Bu operatörler S-OWA olarak adlandırılmıştır. Yager ve Filev aslında S-OWA ailesinden iki operatör tanıtmıştır. İlki 'orlike' özelliğinde, ikincisi ise'andlike' özelliğinde olan S-OWA operatörü olarak ifade edilmiştir. 'orlike' S-OWA operatörü F_{SO} olarak gösterilip $\alpha \in [0, 1]$ paremetresi ile ağırlıklar aşağıdaki gibi ifade edilir.

$$w_i = \begin{cases} \frac{1}{n}(1-\alpha) + \alpha, & i = 1, \\ \frac{1}{n}(1-\alpha), & i = 2, \ldots, n. \end{cases}$$

4.3 Sıralı Ağırlıklı Operatörün Ağırlıkları

OWA operatörünü böylesine etkin ve önemli kılan ise ilgili ağrılıkların belirlenebilmesidir (Filev ve Yager, 1998). Yager bir metot olarak dilsel ifadeleri kullanmayı önermiştir (Yager, 1993b). O'Hagan ise OWA ağırlıklarının entropisini maksimum alarak ve önceden tanımlanmış bir davranışsal karakter değeri ile ağrılıkların bulunabileceğini söylemiştir (O'Hagan, 1987).

Bazı uygulamalarda, gözlemlere dayanarak OWA operatörüne bağlı ağırlıkları öğrenmemiz gerekir. F; n boyutlu, W ağırlık vektörlü bir OWA operatörü olsun. m parça veri topluluğumuz olduğunu ve bu toplulukların her birinin n+1 gruplamış verisi(tuple) olduğunu varsayalım.

Amaç, modelin işlenme sürecinde OWA ağırlıklarını bulmaktır.

Yukarıda ağırlıkları yaklaşık olarak öğrenebildiğimiz sinirsel yapının şekli vardır.

g fonksiyonun basit lineer sıkıştırma fonksiyonu olduğu varsayılırsa *g(x)=x* olur. Sıralayıcı denilen yere gelen girdiler en büyüğü ilk dala, ikinci en büyük ikinci dala şeklinde atanırlar.

Verilen bir girdi $(a_1, \ldots, a_n)$ için sistemin çıktısı $t = \sum w_j b_j$ şeklinde olur. b_j değeri j. en büyük a_i değerini verir.

Bu durumda hata, e=y-t olur. y değeri istenilen çıktıdır. **Widrow-Hoff** yaklaşımı kullanılarak ağırlıklardaki değişim hesaplanabilinir.

$$\Delta w_j = \eta e b_j$$

Her j için Δw_j değerinde küçük değişimler oluncaya kadar bu işlem tekrarlanır (η : katsayı)

4.4 OWA Ağırlıklarının Bulunması

4.4.1 Parametrik Çok Tipli Lineer Stres Fonksiyonu

OWA ağırlıklarının stres fonksiyonları ile bulunması Yager tarafından öne sürülmüştür.

Stres fonksiyonlarının en önemli özelliği karar vericinin düşüncesine göre istenilen sayıda elemanın değerinin belirlenebilmesidir. F, birim aralıktan birim aralığa tanımlı bir fonksiyon olmak üzere, $F : [0,1] \rightarrow [0,1]$, $F(0) = 0$ ve $F(1) = 1$ eşitlikleri ve $F(x) \geq F(y)$ eşitsizliği her $x > y$ için sağlanmak koşulu ile F fonksiyonu basit birim aralıkta monoton fonksiyon (basic unit interval monoton, BUM) olarak adlandırılmıştır. BUM fonksiyonu kullanılarak OWA ağırlıklıkları şu şekilde üretilir:

$$w_j = F\left(\frac{j}{n}\right) - F\left(\frac{j-1}{n}\right) \tag{4.1}$$

$j = 1, \ldots, n$

Açıkça görülmektedir ki w_j ağırlıkları birim aralıkta kalmakta ve toplamları 1

etmektedir. F fonksiyonu negatif olmayan, birim aralıkta tanımlı bir $h(x)$ fonksiyonu $h(x)\geq 0$ $h : [0,1] \rightarrow R^{+}$ ile aşağıdaki gibi tanımlanırsa BUM fonksiyon yapısı bozulmaz.

$$F(x) = \frac{1}{K} \int_0^x h(y)dy \tag{4.2}$$

ve

$$K = \int_0^1 h(y) \tag{4.3}$$

Eşitlikteki $h(x)$ fonksiyonu stres fonksiyonu olarak adlandırılıp F'in x-ekseni boyunca nerede konumlandığını belirtmektedir. Diğer bir deyişle$h(x)$ fonksiyonu, F fonksiyonu ile elde edilen OWA ağırlıklarının nerede konumlandığını ifade eder. Verilen bir stres fonksiyonu $h(x)$ ve BUM fonksiyonu $F(x)$ ile her zaman OWA ağırlıkları elde edilebilir (Yager, 1996). Ağırlıklar;

$$w_j = F\left(\frac{j}{n}\right) - F\left(\frac{j-1}{n}\right) \tag{4.4}$$

şeklinde ifade edilmekteydi. Bu eşitlik aşağıdaki gibi düzenlenirse;

$$w_j = F\left(\frac{j}{n}\right) - F\left(\frac{j}{n} - \frac{1}{n}\right) \tag{4.5}$$

$\frac{1}{n}$ ifadesine Δ denilirse;

$$w_j = F\left(\frac{j}{n}\right) - F\left(\frac{j}{n} - \Delta\right) \tag{4.6}$$

ve eşitliğin daha anlaşılır bir hale gelmesi için;

$$w_j = \left(\frac{F\left(\frac{j}{n}\right) - F\left(\frac{j}{n} - \Delta\right)}{\Delta}\right))\Delta \tag{4.7}$$

$\tilde{w}$ ifadesini orijinal metotta elde edilen w_j ağırlıklarının yaklaşığı olarak belirtebiliriz. Stres fonksiyonundan elde edilen yaklaşık ağırlıkların toplamının 1'e eşit olmasını garantilemek için ağırlıları normalize etmek gerekir.

$$w_j = \frac{\frac{1}{K}h\left(\frac{j}{n}\right)\frac{1}{n}}{\sum_{j=1}^{n}\frac{1}{K}h\left(\frac{j}{n}\right)\frac{1}{n}} = \frac{h\left(\frac{j}{n}\right)}{\sum_{j=1}^{n}h\left(\frac{j}{n}\right)} \tag{4.8}$$

OWA ağırlıklarının stres fonksiyonu ile elde edilmesinin bir çok kullanışlı özelliği vardır. Bunlardan biri herhangi bir n değeri için aynı fonksiyon kullanılabilmektedir. Böylelikle farklı argüman sayılarına göre tutarlı bir ağırlık vektörü oluşturulabilmektedir. Diğer özelliği ise kullanıcının kolaylıkla OWA yığışım operatörünün sonuçlarının doğasını anlamlandırabilmesini sağlar.

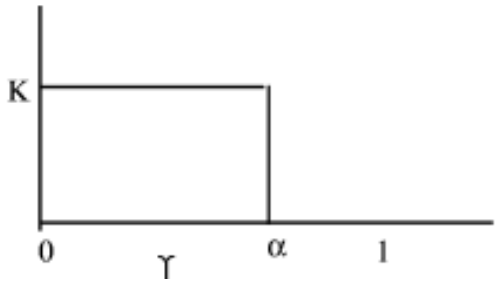
Stres fonksiyonları tipleri aşağıdaki gibi belirtilmiştir.

- Şekil 4.1'deki sabit seviye stres fonksiyonu tüm verilere aynı derece ile yaklaşır. Tüm seçimlerin üyelik değerleri aynıdır. Davranışsal karakteri bu sebeple 0,5'dir. K parametresi yükseklik değeridir.

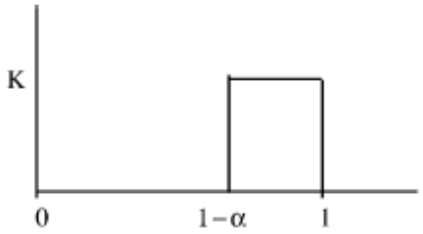
Şekil 4.1 Sabit seviye stres fonksiyonu

- Şekil 4.2'deki stres fonksiyonu küçük değerli seçimlerin ağırlıkları ile ilgilenir. En az ağırlık en az değerli seçime aittir. K parametresi yükseklik değeri ve α değeri ise ne kadar küçük değerli verilerin seçileceğini belirler.

- Şekil 4.3'deki stres fonksiyonu ise Şekil Şekil 4.2'deki stres fonksiyonun tersi mantığa sahiptir. 1-α kısmındaki verilerin değerleri ile ilgilenir.

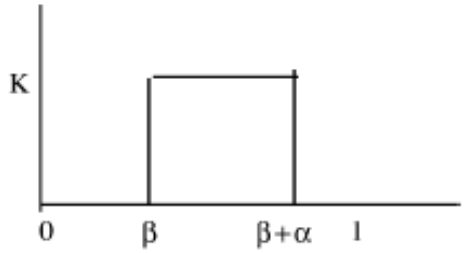


Şekil 4.2 Sabit seviye aralıklı stres fonksiyonu



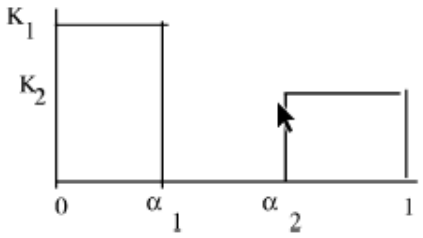
Şekil 4.3 Sabit seviye üst uç stres fonksiyonu

- Şekil 4.4'deki stres fonksiyonu ile ara kısımdaki seçimlerin değerlendirmesi yapılır. Bu ara değerlerin seçimleri için α ve β değerleri kullanılır.

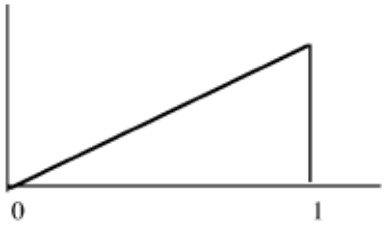


Şekil 4.4 Sabit seviye aralıklı stres fonksiyonu

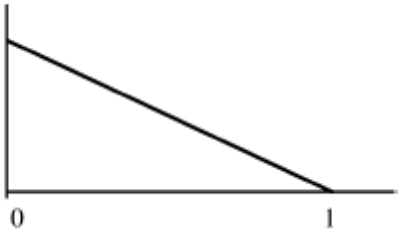
- Şekil 4.5'deki stres fonksiyonu hem küçük hem büyük değerleri belirli bir oran da seçebilmektedir. Oranları yükseklikleri ise K_1 ve K_2 değerleri ile belirlenir.
- Şekil 4.6'deki stres fonksiyonu lineer bir stres fonksiyonudur. Davranışsal karakteri $1/3$ değerindedir.
- Şekil 4.7'deki stres fonksiyonu ise azalan lineer yapıdadır. Şekil 4.6'daki stres fonksiyonuna göre davranışsal karakter değeri $2/3$'tür.



Şekil 4.5 İki uçlu stres fonksiyonu

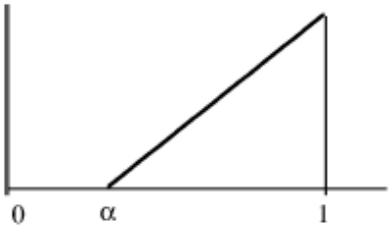


Şekil 4.6 Lineer stres fonksiyonu



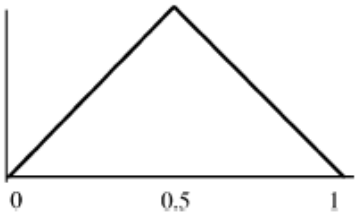
Şekil 4.7 Azalan lineer stres fonksiyonu

- Şekil 4.8'deki stres fonksiyonu adlığı α değeri ile belirli bir orandan sonra verilerle ilgilenir.



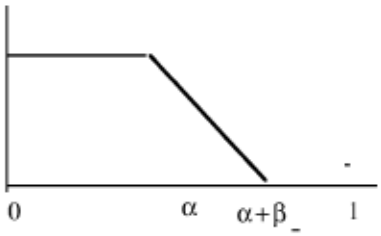
Şekil 4.8 Kısmi lineer stres fonksiyonu

- Şekil 4.9'deki stres fonksiyonu ise merkez tipli stres fonksiyonu olup toplam veri sayısının tek ya da çift sayı da olmasına göre farklı sonuçlar verebilir.



Şekil 4.9 Merkez tipli lineer stres fonksiyonu

- Şekil 4.10'deki stres fonksiyonu ise hem lineer hem de sabit fonksiyon içermektedir. Belirli bir α değerine kadar sabit, belirli bir $\beta - \alpha$ değeri kadar azalan lineer fonksiyon yapısı içermektedir.



Şekil 4.10 Çok tipli stres fonksiyonu (sabit ve lineer)

4.4.2 Yapay Arı Kolonisi Algoritması (Artificial Bee Colony Algorithm)

Optimizasyon için 2005 yılında D.Karaboğa tarafından tasarlanmış yapay arı kolonisi algoritması doğadaki arıların yiyecek arama davranışlarının modellenmesi ile bulunmuştur. Yapay arı kolonisi algoritmasının amaçladığı problem; arıların yüksek nektarlı yiyecek kaynağını elde etmek için bir yiyecek uzayına gitme ve bırakma hareketlerini inceler. Yüksek nektarlı yiyecek kaynağını elde etmek için gidilen kaynakların belirli özellikleri vardır. Bu özellikler kaynağın değeri için zenginliği, yuvaya yakın olup olmaması, çeşidi ve nektar miktarı olabilir. Kovanda yer alan arılar yaptıkları işlere göre sınıflandırılırlar. Görevli işçi arı belirlenmiş yiyecek kaynağına giderek nektarı kovana getirmekten sorumlu iken görevsiz arılar ise yeni kaynak arayışı içindedir. Görevsiz arılar ise iki tür olarak ele alınabilir; rastgele kaynak arayan kaşif arılar ve kovana gelen bilgi yardımıyla yeni kaynağa yönelen

gözcü arılar. Arıların görev paylaşımı ve bu görevler ile elde edilen ortak bilgi çok önemlidir.

- Bir arı bir kaynaktan sorumlu olmalı
- Toplam yiyecek sayısı görevli arı sayısına eşit olmalı
- Kovanda işçi ve gözcü arılar eşit sayıda olmalı

yukarıda ifade edilen maddeler sağlandığından işleyeyecek olan algoritma aşağıdaki şekildedir:

Adım 1. Başlangıç yiyecek kaynağını üret
Adım 2. Durdurma kriteri sağlanmadıkça Adım 2'ye dön
Adım 3. Görevli arıları yiyecek kaynaklarına gönder ve nektar miktarını belirle
Adım 4. Kaynakların seçilme olasılıklarını nektarın miktarına göre hesapla
Adım 5. Gözcü arıları olasılık değerlerine göre yiyecek kaynaklarına gönder
Adım 6. Tükenmiş kaynakları bırak
Adım 7. Kaşif arıları yeni kaynaklar için gönder
Adım 8. En iyi nektarlı kaynağı hafızaya al
Adım 9. Adım 2'ye dön
Adım 10. Elde edilen en iyi çözümü göster.

Başlangıç yiyecek kaynağının üretilmesi her parametrenin alt ve üst sınırları arasında rastgele seçimler ile sağlanır (Şekil 4.11).

$$x_{ij} = x_j^{min} + rand(0,1)(x_j^{max} - x_j^{min}) \tag{4.9}$$

Yiyecek kaynağı $i = 1, \ldots, SN$ ile optimize edilecek parametre sayısı ise $j = 1, \ldots, D$ ile ifade edilmektedir. j.parametrenin alt sınırı x_j^{min} ile ifade edilirken, üst sınırı x_j^{max} ile ifade edilir. Bir hata değeri ya da maksimum çevrim sayısı aramayı durdurma kriteri olarak belirlenebilir.

	x_1	x_2	x_3	x_4	,,,	i	...	x_{n-2}	x_{n-1}	x_n
1	0,2	0,7	0,24	0,14	0,36	0,88	0,9	0,4	0,5	0,62
2	0,7	0,5	0,28	0,4	0,56	0,03	0,07	0,7	0,2	0,5
⋮										
i	0,37	0,65	0,8	0,43	0,25	0,3	0,67	0,45	0,3	0,09
⋮										
⋮										
k	0,88	0,2	0,36	0,10	0,3	0,9	0,7	0,45	0,36	0,19
⋮										
SN	0,9	0,8	0,08	0,14	0,36	0,06	0,07	0,1	0,2	0,7

Şekil 4.11 Başlangıç yiyecek kaynaklarının atanması

Görevli arıların en önemli sorumluluğu gittikleri yiyecek kaynağının belirli komşuluğunda yeni kaynaklar bulması ve bu yeni kaynakları mevcut kaynaklar ile kıyaslayarak en iyi olanı hafızada tutmasıdır.

$$v_{ij} = x_{ij} + \Omega(x_{ij} - x_{kj}) \tag{4.10}$$

4.10 eşitliği ile mevcut kaynak komşuluğunda yeni kaynak belirlenir. k ve Ω sırasıyla $1, \ldots, SN$ ve $[-1, 1]$ aralığında rastgele değer alan sayılardır. Yeni kaynakların parametrelere ait sınırlar içinde yer alıyor olması 4.11 denklemi ile kontrol edilebilir.

$$A = \begin{cases} x_j^{min} & v_{ij} < x_j^{min}, \\ v_{ij} & x_j^{min} \leq v_{ij} \leq x_j^{max}, \\ x_j^{max} & v_{ij} > x_j^{max}. \end{cases} \tag{4.11}$$

uygunluk değer hesaplaması (fitness değeri) aşağdaki 4.12 denklemi ile yapılmadan önce sınır değerlerine öteleme yapılmalıdır.

$$F_i = \begin{cases} \frac{1}{1+f_i} & f_i \geq 0, \\ 1 + abs(f_i) & f_i < 0. \end{cases} \tag{4.12}$$

Eski kaynağın silinip yeni kaynak hafızaya alınması üretilen yeni kaynak uygunluk değerinin daha iyi olmasına bağlıdır. Yeni kaynak daha iyi durumda değil ise mevcut kaynak için geliştirilememe sayacı (failure) bir arttırılır, geliştirdiği zaman ise sıfırlanır.

Rulet tekniği, turnuva yöntemi ya da sıralamaya dayalı herhangi bir seleksiyon şeması kullanılarak yeni kaynakların seçilme olasılıkları uygunluk değerleri göz önünde bulundurularak hesaplanabilir.

Yeni bir kaynak seçiminde, rulet tekniği kullanılacak ise denklem 4.13 yardımı ile seçilme oalsılığı hesaplanabilir.

$$p_i = \frac{F_i}{\sum_{j=1}^{SN} fitness_j} \tag{4.13}$$

Yeni kaynağın nektar miktarının artması hangi seçim tekniğin kullanıldığına bakmaksızın seçilme olasılığını arttırmaktadır.

Her bir kaynak için $[0, 1]$ aralığında rastgele sayı üretilerek hesaplanan p_i değerleri ile karşılaştırılır. Gözcü arı hangi kaynağın p_i değeri daha büyükse yeni bir kaynak üretmeye orada başlar.

Tüm arıların çözüm geliştirme sayaçları bir çevrim sonunda kontrol edildikten sonra nektarların tükenip tükenmediği kaynaklardan fayda elde edilip edilmediği bilgisine ulaşılır. Geliştirememe sayacı belirlenmiş eşik değerinin üzerinde ise görevli arı yeni bir kaynak arayışına girer. Bu işlemin belirlenen kısıtlarla sürdürülmesi sonucu en çok nektara sahip kaynak elde edilmiş olunur.

Yapay arı koloni algoritması ile bilinen tüm stres fonksiyon tiplerinin genelleştirilmiş hali elde edilebilmektedir. Algoritma, başlangıçta belirlenen yiyecek

kaynak sayısı kadar rastgele çözümler üreterek fonksiyonların uygunluk değerleri ve başarısızlık sayısına göre en optimum ağırlıkları ve fonksiyonun parametre değerlerini bulmaktadır. Yapay arı koloni algoritması sonucu elde edilen stres fonksiyonunun parametreleri başlangıçta aşağıdaki gibi belirlenmiştir.

x_1	x_2	y_1	y_2

Başlangıçta belirlenen yiyecek kaynakları (çözümleri) için 4 parametreden oluşan bir vektör kullanılmıştır. Buradaki amaç istenen fonksiyonun tek doğru ile çizilip çizilemeyeceği denenerek doğru sayısı artırılıp en ideal kaç doğruda çizimin gerçekleştirilebileceğini saptamaktır. Yapay arı koloni algoritması 4 parametreden oluşan vektör için belirlen iterasyon sayısı kadar çalıştıktan sonra doğru sayısı 1 arttırılır, diğer bir deyişle çözüm vektörü için parametre sayısı 2 artar.

x_1	x_2	x_3	y_1	y_2	y_3

Yapay arı koloni algoritması 6 parametreli çözüm vektörü için yine belirlenen iterasyon sayısı kadar çalışır ve iterasyondaki en iyi ağırlıkları ve çözümü saklar. Bir sonraki adımda çözüm vektörüne ait parametre sayısı 2 arttırılarak verilen fonksiyonun 3 doğru ile çizimindeki en iyi çözüm saklanır.

x_1	x_2	x_3	x_4	y_1	y_2	y_3

Doğru sayıları birer birer artarak (çözüm vektörü parametreleri için 2'şer artarak) sonraki çözüm ile önceki çözümün hatalarının belirlenmiş eşik değerine göre kıyası yapılarak optimum doğru sayısı verilen fonksiyon için bulunmuş olur.

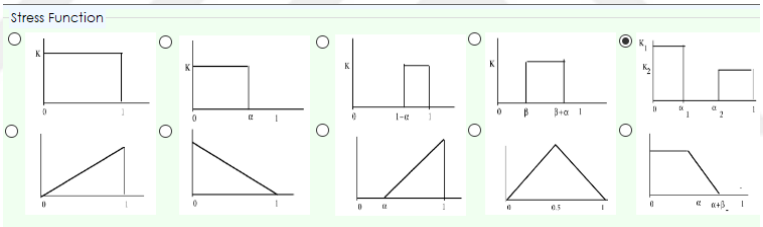
BÖLÜM BEŞ
PROGRAM VE SONUÇLAR

5.1 Program

Bu çalışmada stres fonksiyonları bilinen OWA ağırlıkları ile elde edilmeye çalışılmıştır. OWA ağırlıklarının elde edilmesi için yapay arı koloni algoritmasından yaralanılmıştır.

5.1.1 Program Çalışma Prensibi

Kullanıcı ilk adımda belirlenmiş tipteki stres fonksiyonlarından birini seçer. Belirli tipteki stres fonskiyonlarının seçimi Şekil 5.1'de gösterilen seçim ekranından yapılır.



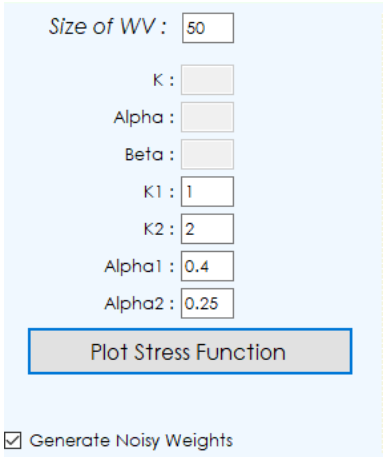
Şekil 5.1 Stress fonksiyon seçimi

Stres fonksiyon tipinin seçiminden sonra, ilgili fonksiyona ait parametrelerin kullanıcı tarafından girişi Şekil 5.2 ile bir stres fonksiyonun grafiği elde edilir.

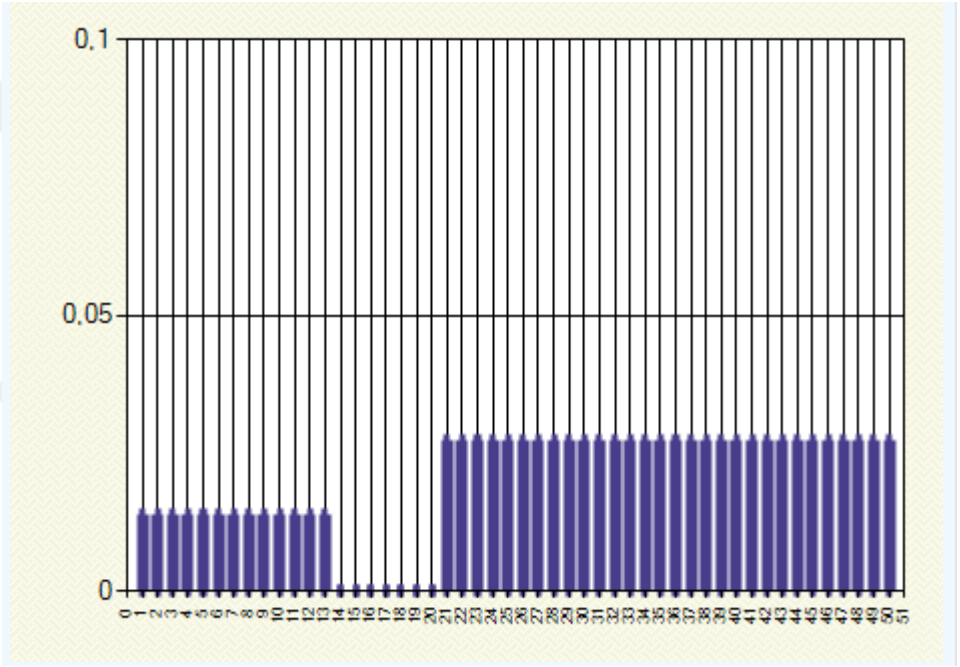
Örnek olarak iki uçlu stres fonksiyon tipi seçilsin. Seçim kriterlerine göre elde edilen fonksiyon grafiği Şekil 5.3'de olduğu gibi görüntülenecektir.

5.1.2 Program Çıktıları

Kullanılan algoritma ve OWA Ağırlıklarına ait grafik Şekil 5.4 ve Şekil 5.5'de gösterilmiştir.



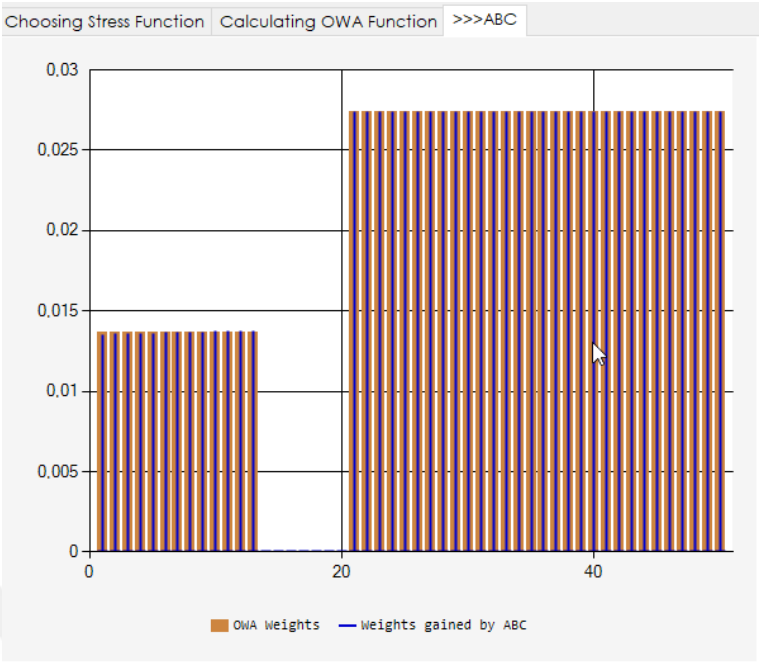
Şekil 5.2 Stress fonksiyon seçim parametreleri



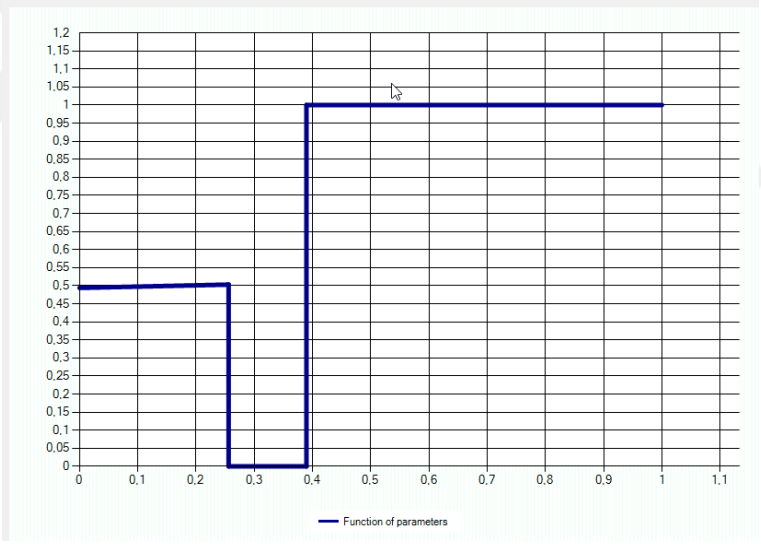
Şekil 5.3 Stress fonksiyon

5.2 Sonuçlar

OWA operatörü, karar problemlerinde karar verme stratejisini belirleme imkanı sunan genel bir yığışım operatörüdür. OWA operatöründe kullanılan stres fonksiyonları, karar verme stratejisinin görsel olarak yansıtılmasına kolaylık sağlamaktadır. Ayrıca stres fonksiyonu yığışımda yer alan değerlerin sayısına uygun olarak ağırlıklar üretilmesini sağlayabilmektedir. Bu çalışmada, ters problem, yani OWA operatörünün ağırlıkları verildiğinde onun stres fonksiyonunun oluşturulması



Şekil 5.4 Sonuç 1



Şekil 5.5 Sonuç 2

problemi ele alınmıştır. Stres fonksiyonu belli kalıplar doğrultusunda parçalı doğrusal fonksiyon aracılığıyla yakınsanılmıştır. Bu amaçla Yapay Arı Kolonisi (ABC) algoritması kullanılmıştır. Ayrıca parçalı doğrusal fonksiyonun oluşturulmasında kullanılan doğru parçalarının sayısı önceden parametre şeklinde verilmeksizin, algoritma aracılığıyla otomatik olarak bulunabilmektedir. Çeşitli ağırlıklar içeren test veri setlerinde algoritma çalıştırılmış ve sonuçların kalıplara uygunluğu görsel olarak da doğrulanmıştır. Önerilen yaklaşım, çeşitli karar problemlerinde kişisel

değerlendirme örnekleri temel alınarak daha genel karar verme stratejilerinin oluşturulması amacıyla kullanıla bilecektir.

KAYNAKLAR

Burkard, R., Dell'Amico, M. ve Martello, S. (2009). *Assignment Problems*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Chen, G. ve Pham, T. T. (2000). *Introduction to fuzzy sets, fuzzy logic, and fuzzy control systems*. Boca Raton: CRC press.

Coşkun, E., Nasiboglu, R. ve Tezel, B. T. (2017). An approach to obtain the generalized mixed linear stress function for known owa weights with artificial bee colony algorithm. *Global Journal of Information Technology: Emerging Technologies*, *6*(3), 150–157.

Filev, D. ve Yager, R. R. (1998). On the issue of obtaining owa operator weights. *Fuzzy Sets and Systems*, *94*(2), 157–169.

Garrido, A. (2012). A brief history of fuzzy logic. *BRAIN. Broad Research in Artificial Intelligence and Neuroscience*, *3*(1), 71–77.

Jang, J.-S., Sun, C.-T. ve Mizutani, E. (1997). Neuro-fuzzy and soft computing-a computational approach to learning and machine intelligence [book review]. *Automatic Control, IEEE Transactions on*, *42*, 1482 - 1484.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical report, *Technical report-tr06,* Erciyes University, Engineering faculty.

Klir, G. ve Yuan, B. (1995). *Fuzzy sets and fuzzy logic* (1 ed.)., volume 4. London: Prentice hall.

Lin, C. J. ve Wen, U. P. (2004). A labeling algorithm for the fuzzy assignment problem. *Fuzzy Sets and Systems*, *142*(3), 373–391.

McNeill, F. M. ve Thro, E. (1994). *Fuzzy logic: A practical approach* (3 ed.). San Diego, CA, USA: Academic Press Professional, Inc.

Nasibov, E. (2004). An algorithm for constructing an admissible solution to the bin packing problem with fuzzy constraints. *Journal of Computer and Systems Sciences International*, *43*(1), 205-212.

Nasibov, E. ve Kınay, A. (2006). Kaliteli iş paylaşımı problemi için bulanık mantık yaklaşımı. *İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi*, *18*(1), 13–22.

Nasibov, E. ve Nasiboglu, R. (2005). Problem of information aggregation in the fuzzy packing problem. *Automatic Control and Computer Sciences*, *39*(1), 29-36.

Öner, N. (1974). *Klasik mantık*. Ankara Üniversitesi Basımevi.

O'Hagan, M. (1987). Fuzzy decision aids. *Proc. 21st Ann. Asilomar Conf. on Signals, Systems, and Computer*, *2*, 624–628.

Ross, T. J. (2010). *Fuzzy logic with engineering applications* (3 ed.). United States: John Wiley and Sons.

Yager, R. R. (1993a). Families of owa operators. *Fuzzy Sets and Systems*, *59*(2), 125–148.

Yager, R. R. (1993b). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, *18*(1), 80–87.

Yager, R. R. (1996). Quantifier guided aggregation using owa operators. *International Journal of Intelligent Systems*, *11*(1), 49–73.

Yager, R. R. (2007). Using stress functions to obtain owa operators. *IEEE Transactions on Fuzzy Systems*, *15*(6), 1122–1129.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353.

Zadeh, L. A. (2008). Is there a need for fuzzy logic? *Information Sciences*, *178*(13), 2751–2779.

EKLER

EK 1: Terimler listesi

OWA	Ordered Weighted Averaging
Membership Function	Üyelik Fonksiyonu
Support	Dayanak
Core	Çekirdek
Normality	Normallik
Ccrossover Points	Köprü Noktaları
Fuzzy Singleton	Tekil Bulanık
α-cut	α-kesim
ABC	Artificial Bee Colony
Artificial Bee Colony Algorithm	Yapay Arı Kolonisi Algoritması
Fitness	Uygunluk

EK 2: OWA ağırlıklarının elde edilmesi ile ilgili C# kaynak kod

```
void WriteNormalWeights(double[] wei) {
string foldername = Path.Combine(Directory.GetCurrentDirectory(),
"Weights");
Directory.CreateDirectory(foldername);
try {
string filename = string.Format("{0}\\Weights {1}.txt", foldername,
DateTime.Now.ToString("G").Replace(':', '-'));
if (File.Exists(filename))
File.Delete(filename);
using (StreamWriter sw = new StreamWriter(File.Open(filename,
FileMode.CreateNew),
Encoding.GetEncoding("ibm857"))) {
for (int i = 0; i < wei.Length - 1; i++) {
sw.Write(wei[i] + ";");}
sw.Write(wei[wei.Length - 1]);}}
catch (Exception error) {
MessageBox.Show(error.ToString());}
}

private void WriteNoisyWeights(double[] n_wei) {
double NoisyWeights = 0;
Random rand = new Random();
for (int i = 0; i < weights.Length; i++) {
noisy_weights[i] = weights[i] *
(1 + (rand.NextDouble() * 2 - 1) / (double)20);
NoisyWeights += noisy_weights[i];}
for (int j = 0; j < weights.Length; j++) {
noisy_weights[j] = noisy_weights[j] / NoisyWeights;}
string noisyfoldername = Path.Combine(Directory.GetCurrentDirectory(),
"NoisyWeights");
Directory.CreateDirectory(noisyfoldername);
try {
string filename = string.Format("{0}\\NoisyWeights {1}.txt",
noisyfoldername,
DateTime.Now.ToString("G").Replace(':', '-'));
```

```
if (File.Exists(filename)) File.Delete(filename);
using (StreamWriter sw = new StreamWriter(File.Open(filename,
FileMode.CreateNew),
Encoding.GetEncoding("ibm857"))) {
for (int i = 0; i < weights.Length – 1; i++) {
sw.Write(noisy_weights[i] + ";");}
sw.Write(noisy_weights[weights.Length –1]);}}
catch (Exception error) {
MessageBox.Show(error.HelpLink);}
}
```

EK 3: Yapay arı kolonisi algoritması ile ilgili C# kaynak kod

```
class ABC {
int VariableCount;
int LimitControl;
int SourceNumber;
double[][] Variable_Limits;
double[][] sources;
double[] fitness;
double[] probs;
double[] Best_Solution;
double BestFit;
int[] limits;
Func<int, double[], double> EvaluateFunction;
public ABC(int variableCount,
double[][] Variable_Limits, int LimitControl,
Func<int, double[], double> evaluateFunction) {

this.LimitControl = LimitControl;
this.VariableCount = variableCount;
this.Variable_Limits = new double[variableCount][];
this.EvaluateFunction = evaluateFunction;

for (int i = 0; i < variableCount; i++) {
this.Variable_Limits[i] = new double[2];
this.Variable_Limits[i][0] = Variable_Limits[i][0];
this.Variable_Limits[i][1] = Variable_Limits[i][1];}}

public void GenerateInitialFoodSource(int sn) {
this.SourceNumber = sn;
Random rand = new Random();
sources = new double[sn][];
limits = new int[sn];
fitness = new double[sn];
probs = new double[sn];

for (int i = 0; i < sn; i++) {
```

```
sources[i] = new double[VariableCount];
for (int j = 0; j < VariableCount; j++) {
sources[i][j] = Variable_Limits[j][0] + rand.NextDouble() *
(Variable_Limits[j][1] – Variable_Limits[j][0]);}}
}

private void ResetFoodSource(int index) {
Random rand = new Random();
sources[index] = new double[VariableCount];
for (int j = 0; j < VariableCount; j++) {
sources[index][j] = Variable_Limits[j][0] + rand.NextDouble() *
(Variable_Limits[j][1] – Variable_Limits[j][0]);}
fitness[index] = EvaluateFunction(VariableCount, sources[index]);
}

private void EvaluateFitness() {
for (int i = 0; i < SourceNumber; i++) {
fitness[i] = EvaluateFunction(VariableCount, sources[i]);}
}

private void EmployedBee() {
Random rand = new Random();
for (int i = 0; i < SourceNumber; i++) {
double[] New_Source = new double[VariableCount];
Array.Copy(sources[i], New_Source, sources[i].Length);
int j_ind = rand.Next(0, VariableCount);
int k_ind = rand.Next(0, SourceNumber);
double temp_var = New_Source[j_ind] + (rand.NextDouble() * 2 – 1) *
(New_Source[j_ind] – sources[k_ind][j_ind]);
if (temp_var < Variable_Limits[j_ind][0])
temp_var = Variable_Limits[j_ind][0];
else if (temp_var > Variable_Limits[j_ind][1])
temp_var = Variable_Limits[j_ind][1];
New_Source[j_ind] = temp_var;
double temp_fit = EvaluateFunction(VariableCount, New_Source);
if (fitness[i] < temp_fit) {
Array.Copy(New_Source, sources[i], New_Source.Length);
```

```
fitness[i] = temp_fit;
limits[i] = 0;}
else {limits[i]++;}}
}

private void EvaluateProbs() {
double top = fitness.Sum();
for (int i = 0; i < SourceNumber; i++) {
probs[i] = fitness[i] / top;}
}

private void OnLookerBee() {
Random rand = new Random();
for (int i = 0; i < SourceNumber; i++) {
double wheel = rand.NextDouble();
double CumulativeProbs = 0.0;
int indis = 0;
for (int j = 0; j < SourceNumber; j++) {
CumulativeProbs += probs[j];
if (wheel <= CumulativeProbs) {
indis = j;
break;}}
double[] New_Source = new double[VariableCount];
Array.Copy(sources[indis], New_Source, sources[indis].Length);
int j_ind = rand.Next(0, VariableCount);
int k_ind = rand.Next(0, SourceNumber);
double temp_var = New_Source[j_ind] + (rand.NextDouble() * 2 - 1) *
(New_Source[j_ind] - sources[k_ind][j_ind]);
if (temp_var < Variable_Limits[j_ind][0])
temp_var = Variable_Limits[j_ind][0];
else if (temp_var > Variable_Limits[j_ind][1])
temp_var = Variable_Limits[j_ind][1];
New_Source[j_ind] = temp_var;
double temp_fit = EvaluateFunction(VariableCount, New_Source);
if (fitness[indis] < temp_fit) {
Array.Copy(New_Source, sources[indis], New_Source.Length);
fitness[indis] = temp_fit;
```

```
// şgelitirememe ıkatsays
limits[indis] = 0;}
else{limits[indis]++;}}
}

private void GetBestSolution() {
int BestIndex = 0;
for (int i = 1; i < SourceNumber; i++) {
if (fitness[BestIndex] < fitness[i]) {
BestIndex = i;}}
BestFit = fitness[BestIndex];
Array.Copy(sources[BestIndex], Best_Solution, VariableCount);
}

private void checkFLimits() {
for (int i = 0; i < SourceNumber; i++) {
if (limits[i] > LimitControl) {
ResetFoodSource(i);}}
}

void SendBee() {
int MaxTrialIndex = 0, i;
for (i = 1; i < SourceNumber; i++) {
if (limits[i] > limits[MaxTrialIndex])
MaxTrialIndex = i;}
if (limits[MaxTrialIndex] >= LimitControl) {
ResetFoodSource(MaxTrialIndex);}
}

public double[] run(int it) {
Best_Solution = new double[VariableCount];
EvaluateFitness();
for (int i = 0; i < it; i++) {
EmployedBee();
EvaluateProbs();
OnLookerBee();
GetBestSolution();
```

```
SendBee();
//checkFLimits();      }
return Best_Solution;}
}
```