PARÇACIK SÜRÜ OPTİMİZASYONU YÖNTEMLERİNİN UYGULAMALARLA KARŞILAŞTIRILMASI

2011
YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

Yasin ORTAKCI

PARÇACIK SÜRÜ OPTİMİZASYONU YÖNTEMLERİNİN UYGULAMALARLA KARŞILAŞTIRILMASI

Yasin ORTAKCI

Karabük Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalında
Yüksek Lisans Tezi
Olarak Hazırlanmıştır

KARABÜK
Haziran 2011

Yasin ORTAKCI tarafından hazırlanan "PARÇACIK SÜRÜ OPTİMİZASYONU YÖNTEMLERİNİN UYGULAMALARLA KARŞILAŞTIRILMASI" başlıklı bu tezin Yüksek Lisans Tezi olarak uygun olduğunu onaylarım.

Doç. Dr. Cevdet GÖLOĞLU .....

Tez Danışmanı, Makine Mühendisliği Anabilim Dalı

Bu çalışma, jürimiz tarafından oy birliği ile Bilgisayar Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir. 23 / 06 / 2011

Ünvanı, Adı SOYADI (Kurumu) İmzası

Başkan : Prof. Dr. Abdullah ÇAVUŞOĞLU (KBÜ) .....

Üye : Doç. Dr. Cevdet GÖLOĞLU (KBÜ) .....

Üyе : Yrd. Doç. Dr. Baha ŞEN (KBÜ) .....

...../...../2011

KBÜ Fen Bilimleri Enstitüsü Yönetim Kurulu, bu tez ile, Yüksek Lisans derecesini onamıştır.

Doç. Dr. Nizamettin KAHRAMAN .....

Fen Bilimleri Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Yasin ORTAKCI

ÖZET

Yüksek Lisans Tezi

PARÇACIK SÜRÜ OPTİMİZASYONU YÖNTEMLERİNİN UYGULAMALARLA KARŞILAŞTIRILMASI

Yasin ORTAKCI

Karabük Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı:
Doç. Dr. Cevdet GÖLOĞLU
Haziran, 2011, 63 sayfa

Optimizasyon problemlerinin çözümünde birçok algoritma kullanılmaktadır. Popülasyon tabanlı evrimsel arama algoritması olan Parçacık Sürü Optimizasyonu (PSO), uygulama kolaylığı ve hızlı çözüm bulma yeteneği ile diğer optimizasyon algoritmaları arasında dikkat çekmektedir.

Tez kapsamında, mühendislik optimizasyon problemlerinden, sürekli, tamsayı ve ayrık değişken tipleri içeren kısıtlara sahip bir problemin, uygunluk fonksiyonu çerçevesinde optimize edilmesi hedeflenmiştir. PSO'da kısıtlar, Uygunluk Tabanlı Kurallar (UTK) yöntemi ile ele alınmıştır. PSO'nun yapısı gereği zamanla çözüm uzayının dışına çıkabilen değişken değerlerini arama uzayında sınırlandırmak için Emme, Yansıtma, Sönümleme, Görünmez, Görünmez Yansıtma ve Görünmez Sönümleme teknikleri kullanılarak, bu tekniklerin başarım karşılaştırmaları yapılmıştır.

Yine PSO kümeleme algoritması olarak kullanılmış ve zambak çiçeği verilerinin kümelenmesi gerçekleştirilmiştir. Kümeleme başarımı, Kümeleme Doğruluk İndeksi (KDİ) olarak adlandırılan üç farklı uygunluk fonksiyonu tarafından değerlendirilmiş ve sonuçlar karşılaştırılmıştır. Öncelikle yöneticisiz kümeleme yöntemleri kullanılarak küme sayısı bulunmaya çalışılmış, daha sonra yöneticili kümeleme yöntemi kullanılarak kümeleme başarımı ölçülmüştür.

Ayrıca, Fuzzy C-Means (FCM) kümeleme algoritması ile PSO hibritleştirilmiş ve iki farklı yöntem, Küme Merkezine Dayalı FPSO (C-FPSO) ve Üyeliğe Dayalı FPSO (M-FPSO) ile kümeleme başarımları ölçülmüştür.

Anahtar Sözcükler : Optimizasyon, parçacık sürü optimizasyonu, kısıtlar, karma değişkenler, kümeleme.

Bilim Kodu : 902.1.014

ABSTRACT

M.Sc. Thesis

COMPARISION OF PARTICLE SWARM OPTIMIZATION METHODS IN APPLICATIONS

Yasin ORTAKCI

Karabük University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Thesis Supervisor:
Assoc. Prof. Dr. Cevdet GÖLOĞLU
June 2011, 63 pages

Many algorithms in solving optimization problems are used. Particle Swarm Optimization (PSO), which is a population based evolutionary algorithm, is distinguished by the easiness in implementation and the speed in comparison to other optimization algorithms.

In this thesis, it is aimed to optimize an engineering optimization problem that possesses different constraints including continuous, integer and discrete variable types by using fitness functions. The constraints of the problem are handled with Feasibility Based Rules (FBR) method. The techniques of Absorbing, Reflecting, Damping, Invisible, Invisible Reflecting, and Invisible Damping are used in order to limit the values of variables that over flow to the outside of the searching space. The performances of the aforementioned methods are compared to one another.

Besides, iris flower data set is clustered by using PSO as a clustering algorithm. The clustering performance is evaluated by three fitness functions called as Clustering Validity Indexes (CVI) and the results are compared. For this purpose, firstly, unsupervised clustering method is used in order to find the number of cluster, then, the supervised clustering method is employed for clustering performance measure.

In addition, Fuzzy C-Means clustering algorithm (FCM) and PSO are hybridized and two different methods, Center Based FPSO (C-FPSO) and Membership Based FPSO (M-FPSO), are utilized for the performance measure of the hybrid clustering algorithm.

Key Words : Optimization, particle swarm optimization, constraints, mixed variable, clustering.

Science Code : 902.1.014

TEŞEKKÜR

Tez çalışmalarım boyunca danışmanlığımı yapan; bilgi birikimi ile bana yol gösteren ve tecrübelerini benden esirgemeyen Sayın Hocam Doç. Dr. Cevdet GÖLOĞLU'na sonsuz teşekkür ederim.

Tez çalışmalarım esnasında tez konumla ilgili araştırma yapmak üzere yurt dışına gitmeme imkan sağlayan Prof. Dr. Abdullah ÇAVUŞOĞLU'na ve Sayın Rektörümüz Prof. Dr. Burhanettin UYSAL'a teşekkür ederim.

Amerika Birleşik Devletleri Auburn Üniversitesi'nde yaptığım çalışmalar esnasında bana danışmanlık yaparak çalışmalarıma olan katkılarından dolayı Endüstri Mühendisliği Bölüm Başkanı Prof. Dr. Alice Smith'e teşekkür ederim.

Beni bugünlere getiren, eğitim hayatım boyunca benden her türlü desteklerini esirgemeyen anneme ve babama ve yine yüksek lisans eğitimim boyunca bana sonsuz destek veren eşime yürekten teşekkür ederim.

İÇİNDEKİLER

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

KISALTMALAR

ACO : Ant Colony Optimization
BCO : Bee Colony Optimization
C-FPSO : Center Based FPSO
CVI : Clustering Validity Index
ES : Evolutionary Programming
FBR : Feasibilty Based Rules
FPSO : Fuzzy Dekstekli PSO Kümeleme Algoritması
GA : Genetik Algoritmalar Evrimsel Stratejiler
GP : Genetic Programming
KDİ : Kümeleme Doğruluk İndeksi
M-FPSO : Membership Based FPSO
PSO : Parçacık Sürü Optimizasyonu
SA : Simulated Annealing
UTK : Uygunluk Tabanlı Kurallar

BÖLÜM 1

GİRİŞ

Optimizasyon, muhtemel çözümlerden oluşan bir kümedeki en uygun çözümü sorunla ilgili kısıtları da göz önünde bulundurarak bulmaya çalışan matematiksel bir yöntemdir. Optimizasyon yöntemleri sistematik bir şekilde belirtilen aralıktaki parametrik değerleri kullanarak gerçek bir fonksiyonun maksimum yada minimum değerini ararlar. Bu yöntemler matematik, bilgisayar bilimleri, ekonomi, mühendislik, endüstri ve tıp gibi birçok alanda yaygın olarak kullanılmaktadır.

Optimizasyon konusu kullanım alanlarına ve yöntemlerin uygulanış şekillerine göre birçok alt alana sahiptir. Bunlardan bazıları Olasılıklı (Stochastic) Programlama, Sezgisel (Heuristic) Yöntemler, Evrimsel Algoritmalar ve Popülasyon Tabanlı Algoritmalardır.

Olasılıklı Programlama rastgele değerler kullanarak en iyiyi bulmaya çalışırlar. Rastgele oluşturulan muhtemel çözümler değerlendirilir. Bu değerlendirmeler sonucu bulunan en iyi rastgele çözüm önerisi sistemin ürettiği cevap olur.

Sezgisel Yöntemler bir döngü içersinde tekrarlı olarak çözüm önerilerini iyileştirerek en iyi çözümü bulmaya çalışırlar. Her bir döngüde muhtemel çözüm ya da çözümler üzerinde işlemler yapılır. Döngüsel arama işlemi sırasında çözüm uzayındaki hiç denenmemiş çözümleri denemek ile önceki adımlarda edinilmiş bilgileri kullanmak arasındaki dengeyi de gözetirler. Çözülecek optimizasyon problemi hakkında çok az varsayım yaparak veya hiç yapmayarak geniş bir çözüm uzayını tararlar. Her zaman en iyi sonucu bulamayabilirler.

Evrimsel Algoritmalar [1], biyolojik evrimi taklit ederler. Biyolojideki doğal seleksiyon ve güçlü olanın doğaya ayak uydurması ve hayatta kalabilmesi

konusundan esinlenerek üretilen algoritmalardır. Belirli sayıda eleman içeren bir popülasyon kullanılarak en iyi çözümü bulmaya çalışırlar. Bu popülasyonda iyi çözümler seçilirken kötü çözümler popülasyondan silinir. Seçilen çözümler arasında veri alış verişi yapılıp yeni çözüm kombinasyonları üretilirken, bazı çözümler üzerinde manipülasyona gidilerek çözüm çeşitliliği sağlanmış olur. Genetik Algoritmalar (GA) [2], Evrimsel Stratejiler (ES) [3], Evrimsel Programlama (EP) [4], ve Genetik Programlama (GP) [5,6] evrimsel algoritmalara örnek olarak gösterilebilir.

Popülasyon Tabanlı Algoritmalar, doğada var olan ilginç ve bilimsel alt yapı oluşturabilecek varlıklardan ve sistemlerden esinlenerek geliştirilen algoritmalardır. Popülasyon tabanlı algoritmalarda çoklu çözüm önerisi kullanılır. Yani her bir iterasyonda bir çözüm önerisi yerine birden fazla çözüm önerisi sunulur. Bu çözüm önerileri aralarında bilgi paylaşımı yaparak veya birbirlerini etkileyerek en uygun çözümü bulmaya çalışırlar. Popülasyon tabanlı algoritmaların kullanımı bazı avantajlar sağlar. Bu avantajlar [7]:

- Çoklu çözümler kullanıldığı için arama uzayındaki en iyi noktanın dışında iyi sonuç veren bölgeler hakkında da bilgi verir,
- Popülasyon ilk iterasyonlarda geniş bir bölgede arama yapar. İterasyonlar ilerledikçe popülasyon üyeleri bilgi paylaşımı yaparak optimum sonuca daha çok yaklaşırlar. Böylece başlangıçta geniş olan arama bölgesi optimum sonuca yaklaştıkça daralır. Popülasyon bu optimum sonuca yaklaşma işlemini herhangi bir kılavuza ihtiyaç duymadan kendisi yapabilir,
- Popülasyon üyelerinin her biri arama uzayında farkı bölgelerde paralel bir arama gerçekleştirdiği için yerel optimumlara takılma riski azdır,

şeklinde sıralanabilir. Karınca Kolonisi Optimizasyonu (ACO) [8], Benzetimli Tavlama (SA) [9], Arı Koloni Optimizasyonu (BCO) [10], Parçacık Sürü Optimizasyonu (PSO) popülasyon temelli optimizasyon algoritmalarıdır.

Bu tezde popülasyon tabanlı sezgisel bir optimizasyon yöntemi olan Parçacık Sürü Optimizasyonunu (PSO) iki farklı probleme uygulanması hedeflenmektedir.

Birinci problem tasarım aşamasında bazı kısıtlara sahip baskı yaylarının tasarımıdır. Bu problemde PSO ile sürekli, ayrık, tamsayı değişkenlerinin kullanım şekilleri incelenecektir. Problem ile ilgili kısıtların ele alınış yöntemi ise Uygunluk Tabanlı Kurallar (UTK) yöntemi olacaktır. PSO'da optimize edilen değişkenler zamanla çözüm uzaylarının dışına çıkabilmektedirler. Yapılacak çalışmada, baskı yayı tasarım değişkenlerinin çözüm uzaylarının sınırlandırılması için farklı yöntemler kullanılacak ve bu yöntemlerin performansları değerlendirilecektir.

İkinci problem ise kümeleme aracı olarak PSO ele alınmıştır. Üç farklı kümeleme indeksleri kullanılarak, zambak çiçeği verilerinin PSO ile kümeleme işlemi gerçekleştirilecektir. Öncelikle PSO, yöneticisiz kümeleme algoritması olarak kullanılacak ve zambak çiçeği verilerinin ayrılabileceği küme sayısı üç farklı kümeleme indeksi için ayrı ayrı bulunmaya çalışılacaktır. Sonrasında ise PSO, yöneticili kümeleme algoritması olarak kullanılacak ve kümeleme indekslerinin kümeleme başarımı ölçülecek ve birbirleri ile kıyas edilecektir. Ayrıca Bulanık C-Means kümeleme algoritması ile PSO algoritması birleştirilerek hibrit bir kümeleme algoritması oluşturulacaktır. Bu hibritleştirme işleminde ise üyeliğe dayalı ve küme merkezine dayalı olmak üzere iki farklı kümeleme yöntemi kullanılacaktır. Bu iki yöntemin kümeleme başarımı ise yine zambak çiçeği verileri üzerinde ölçülecektir ve karşılaştırılacaktır.

BÖLÜM 2

OPTİMİZASYON

Matematiksel programlama olarak da adlandırılan optimizasyon, bir uygunluk (değerlendirme) fonksiyonuna göre belirli aralıktaki sayısal değerlerin en uygununu seçerek kompleks problemleri çözmektir. Optimizasyon problemlerinde öncelikle uygunluk fonksiyonu seçilmelidir. Uygunluk fonksiyonu muhtemel çözümlerin kalitesini ve performansını belirler. Değerlendirme fonksiyonunun sonucu kârı, zamanı, herhangi bir özelliği yada özellikler grubunu temsil eden sayısal bir ifade olabilir. Uygunluk fonksiyonu çözülecek problemle ilgili karakteristik özelliklere sahip değişken adı verilen değerlere göre hesaplanır. Amaç en uygun değişkenleri bulmaktır. Uygunluk fonksiyonları probleme göre bir maksimizasyon ya da minimizasyon fonksiyonu olabilir. Probleme göre değerlendirme fonksiyonlarının hesaplanması sırasında bazı kısıtlara dikkat edilmesi gerekir. Uygunluk fonksiyonlarında çözüm uzayındaki bütün değişkenlerin değil bu kısıtlara uyan değişkenler kullanılır.

Bir optimizasyon probleminde değişkenlerin, kısıtların ve uygunluk fonksiyonlarının tanımlanması modelleme olarak adlandırılabilir. Bu modeldeki değişkenlerin ve kısıtların sayılarının artması modelin kompleks bir hal almasına ve genellikle çözümü için bilgisayar uygulamalarına ihtiyaç duyulmasına neden olur. Bütün optimizasyon problemlerinin çözümü için kullanılabilecek genel bir optimizasyon algoritması yoktur. Probleme dayalı olarak kullanılabilecek çeşitli optimizasyon algoritmaları vardır. Probleme göre bu algoritmanın seçimi kullanıcının sorumluluğunda olan bir işlemdir ve problemi çözme konusunda önemli bir etkendir [11].

2.1. OPTİMİZASYONUNUN MATEMATİKSEL İFADESİ

x, değişkenlerden oluşan bir vektör,

f, uygunluk fonksiyonu,

G_i, eşitsizliklerden ve

H_i, eşitliklerden oluşan kısıt vektörleri olmak üzere uygunluk fonksiyonu minimizasyon şeklindeki bir optimizasyon problemi;

$$\begin{aligned} &min\big(f(\vec{x})\big) \\ &G_i(\vec{x}) \leq 0 \qquad i = 1, 2, \ldots., m; \\ &H_j(\vec{x}) = 0 \qquad j = 1, 2, \ldots., n; \end{aligned} \tag{2.1}$$

şeklinde ifade edilebilir.

2.2. AYRIK VE SÜREKLİ OPTİMİZASYON

Bazı optimizasyon problemlerinde kullanılan değişkenler tamsayı tipinde olması gerekir. Buna göre optimizasyon probleminde, $\forall\ i$ ve j için $x_{ij} \in Z$ olmalıdır. Ayrık optimizasyonun temelinde sonlu sayıda elemen içeren bir çözüm kümesinden en uygun olanlarını seçmek vardır. Sürekli optimizasyonda ise sonsuz sayıda eleman içeren bir kümeden en uygun elemanı seçmek gerekir. Sürekli optimizasyon problemlerinin çözümünde değişkenler üzerinde herhangi bir işlem yapmaya gerek olmadığı için uygulaması kolaydır. Yalnız sonsuz sayıda eleman arasından seçim yapılması gerektiği için sonuç bulmak ayrık optimizasyon problemlerine göre daha zordur. Çünkü ayrık optimizasyon problemlerinde irdelenmesi gereken değişken sayısı sınırlıdır. Bazı optimizasyon problemleri hem ayrık, hem sürekli hem de tam sayı tipinde değişkenler içerebilir. Bu tip optimizasyon problemleri karma değişkenli optimizasyon problemleri olarak adlandırılır.

2.3. KISITLI VE KISITSIZ OPTİMİZASYON

Optimizasyon problemleri kısıtlara sahip olması durumuna göre kısıtlı ve kısıtsız olmak üzere ikiye ayrılırlar. Problemlerin çözümü esnasında değişkenler, tanımlı

oldukları aralıkta başka sınırlamalara tabi tutuluyorsa bu tip optimizasyon yöntemine kısıtlı optimizasyon denir. Değişkenlerin tanımlı olduğu aralıktaki bütün değerleri uygun çözümleri vermezler. Değişkenin tanım aralığındaki kısıtlara uyan değerlerine uygun (feasible) çözümler, uymayanlara ise uygun olmayan (infeasible) çözümler denir.

2.4. OLASILIKLI VE BELİRLEYİCİ OPTİMİZASYON

Optimizasyon problemlerinin bazılarında değişkenlerin özelliklerine bağlı olarak model tam olarak tanımlanamaz. Modeli oluşturanlar bu bilinmeyen özellikleri tahmin etmeye çalışırlar. Bu bilinmeyen özellikler ile ilgili çeşitli senaryolar üretilir ve bu senaryolara olasılık değerleri atanır. Olasılıklı optimizasyon yöntemleri bu senaryolar arasından en uygununu seçerek probleme en uygun sonucu bulmuş olurlar. Belirleyici (determinist) optimizasyon yöntemlerinde ise olasılıklı optimizasyon yöntemlerinden farklı olarak model tam olarak tanımlanmış ve olasılığa yer verilmemiştir.

2.5. DOĞRUSAL VE DOĞRUSAL OLMAYAN OPTİMİZASYON

Doğrusal algoritmalar, uygunluk fonksiyonun doğrusal fonksiyonlardan oluştuğu ve problemle ilgili eşit ve eşitsizliklerin doğrusal olduğu optimizasyon algoritmalarıdır. Doğrusal olmayan algoritmalar ise uygunluk fonksiyonun, problemle ilgili kısıtların karekök ve trigonometrik fonksiyonlar gibi doğrusal olmayan fonksiyonların olduğu optimizasyon algoritmalarıdır [12].

2.6. CEZA FONKSİYONLARI

Gerçek hayatta karşılaştığımız bir çok optimizasyon problemi kısıtlara sahiptir. Bu problemlerin çözümünde bulunan sonuçların bu kısıtlara da uyması gerekmektedir. Kısıtlara uyan değerlere uygun (feasible) değer, uymayan değerlere ise uygun olmayan (infeasible) değerler denir. Optimum değerlerin bulunması sırasında bulunan birçok uygun olmayan değerlere rastlanmaktadır. Bazı problemlerde bu uygun olmayan değerlerin onarılarak uygun hale getirilmesi söz konusu iken bazı

problemlerde bu değerlerin onarılması mümkün olmamaktadır. Bu durumda çözümlere ceza (penalty) değerleri uygulanarak uygun olmayan çözümlerin optimum çözüm olması engellenmeye çalışılmaktadır. Ceza fonksiyonları amaç fonksiyonlarına değişik şekillerde uygulanarak uygun olmayan çözümler cezalandırılabilmektedir. Ceza değerleri amaç fonksiyona sabit bir katsayı olarak uygulanabileceği gibi, belirli parametrelere bağlı olarak dinamik olarak da uygulanabilir. Ceza fonksiyonu parametrelerinin amaç fonksiyonuna iyi bir şekilde adapte edilmesi gerekir. Aksi takdirde optimum sonuç bulunamayabilir [13, 14].

2.7. OPTİMİZASYON ALGORİTMALARI

Optimizasyon algoritmaları genellikle tekrarlamalıdır. Tahmini veya belirli bir başlangıç değeri ile arama işlemine başlarlar ve iterasyonlar boyunca en iyi sonucu bulmaya çalışırlar. İterasyonlar ilerledikçe bulunan çözümlerin kalitesi de artar. Bir iterasyondan diğer iterasyona geçiş her optimizasyon algoritması için farklılık gösterir. Bazı algoritmalar geçmiş iterasyonda elde ettiği verileri kullanarak çözüm üretirken, bazıları ise her iterasyonda bağımsız çözümler üretirler. İyi bir optimizasyon algoritmasında olması gereken başlıca özellikler şunlardır:

- Geniş bir uygulama alanına sahip olmalıdırlar.
- Çok fazla hesaplama zamanı ve hafızası gerektirmemeliler.
- Verilerdeki hatalardan veya matematiksel yuvarlamalardan çok etkilenmeden detaylı çözümler bulabilmelidirler.

Literatürde bir çok optimizasyon algoritması kullanılmıştır. Bunlardan bazıları;

Macdonald 1960 yılında Doğrusal Programlama ile çeşitli oyunlar ilgili çalışmalar yapmıştır [15]. Yine Doğrusal Programlama ile 1960 yılında Tucker matris oyununun çözmeyi başarmıştır [16]. Pearson and Sridhar 1966 yılında ayrık optimizasyon problemlerini Doğrusal Olmayan yöntemlerle optimize etmek için çalışmalar yapmıştır [17]. Huper ve Trumpf Newton-Like yöntemini kullanarak işaret işleme alanında çalışmalar yapmıştır [18]. Lark ile Wheeler 1962 yılında Newton yöntemlerini otomatik kontrol alanında kullanmışlardır [19]. De ve Pantoja

1989 yılında Ardışıl Quadratik Porgramlamayı ayrık kontrol problemlerinde kullanmışlardır [20]. Viera and Lisboa 2010 yılında Elips metodunu elektromanyetik cihazların tasarım ve optimizasyonu çalışmalarında kullanmışlardır [21].

BÖLÜM 3

PARÇACIK SÜRÜ OPTİMİZASYONU (PSO)

Parçacık Sürü Optimizasyonu (PSO) 1995 yılında Russell Eberhart ve James Kennedy tarafından önerilmiş bir optimizasyon algoritmasıdır [22]. Kuş sürülerinin yiyecek arayışları sırasındaki toplu hareketlerinden esinlenerek ortaya atılmış popülasyon tabanlı evrimsel arama algoritmalarından bir tanesidir. PSO senaryosunda sürünün bir bölgede yiyecek arayışı, sürü mensubu kuşların o bölgeye rastgele dağılımı ile gerçekleşir. Kuşlar yiyeceğin nerede olduğunu bilmezler. Sürüdeki bütün kuşlar eş zamanlı olarak ani hareketlerle yön değiştirerek arama bölgesine yayılırlar ve yiyecek ararlar. Daha sonra eş zamanlı olarak tekrar bir araya gelerek yiyeceğin nerede olduğu konusunda bilgi paylaşımı yaparlar. Sürüdeki kuşlar yiyeceğe ne kadar mesafede olduklarını ve sürüdeki yiyeceğe en yakın kuşun pozisyonunu bilirler. Sürüdeki kuşlar bu elde ettikleri pozisyon bilgilerini kullanarak yiyeceğe ulaşırlar [23]. Gerçek hayatta da yiyecek olan bir bölgede bir kuş varken kısa bir süre sonra bu bölgeye birçok kuşun geldiği görülmektedir.

Senaryoda bahsedilen kuşlar, PSO'da parçacık olarak adlandırılır ve her bir parçacığın pozisyon bilgisi bir çözümü temsil eder. Parçacığın pozisyon değiştirme miktarı ise parçacığın hızı olarak adlandırılır. Her bir parçacığın pozisyon değerleri uygunluk fonksiyonuna bağlı olarak değerlendirilir. Bütün parçacıklar kendi keşfettikleri en iyi pozisyon değerini ve sürüde keşfedilen en iyi pozisyon değerine sahip parçacığı referans alarak çözüme ulaşmaya çalışırlar.

PSO diğer optimizasyon teknikleri ile karşılaştırıldığında özellikle bazı yönleriyle diğer optimizasyon tekniklerinden daha üstün olduğu görülmektedir [24]. Örneğin PSO'da ayarlanması gereken parametre sayısının az olması nedeniyle, PSO diğer optimizasyon yöntemlerine göre uygulaması daya kolay olan bir yöntemdir. Bir başka özelliği ise PSO'da parçacıklar hem kendi en iyi pozisyon değerlerini hem

sürüdeki diğer komşularının en iyi pozisyon değerini hatırladıkları için iyi bir hafıza yeteneğine sahiptirler. Ayrıca PSO'da çözüm uzayında arama yapılırken en iyi pozisyona sahip parçacığın değerinden yararlanılır ve arama uzayında herhangi bir değişiklik olmaz. Buna karşın, genetik algoritmada kötü sonuçlar devre dışı bırakılarak arama uzayı daraltılır. Arama uzayında herhangi bir değişikliğe yada kısıtlamaya gitmemek PSO'nun yerel optimumlara takılmasını engeller.

3.1. STANDART PSO

PSO'da parçacıklar belirli bir aralıkta rastgele olarak pozisyon ve hız değerleri alarak arama işlemine başlarlar. Her yeni nesilde (iterasyonda) parçacıkların arama uzayında bulundukları pozisyonların uygunluk değerleri konuyla ilgili uygunluk fonksiyonunda hesaplanır. Sürüdeki en iyi uygunluk değerine sahip parçacık belirlenir. Aynı zamanda her parçacık nesiller boyunca elde ettiği en iyi değeri hafızasında tutar. Sürüdeki en iyi uygunluk değeri ve her bir parçacığın hafızasındaki kendi en iyi değeri kullanılarak bütün parçacıkların pozisyon ve hız değerleri güncellenir. Bu döngü belirlenen sayıda iterasyonu tamamlanıncaya kadar devam eder. Bütün parçacıklar bu iterasyonlar boyunca çözüm uzayında en iyi çözümü bulmaya çalışırlar.

d boyutlu bir problemde sürüdeki $i.$ parçacığın konum vektörü $X_i = (x_{i1} , x_{i2} , \dots , x_{ik} , \dots , x_{id})$ olsun. Sürünün en iyi parçacığı diğer bir değişle sürüde en iyi uygunluğa sahip olan parçacık (global en iyi) g_{best} olarak adlandırılsın. Sürüdeki her bir parçacığın nesiller boyunca elde ettiği en iyi uygunluk değeri de (kişisel en iyi) p_{best} olarak adlandırılsın. Buna göre sürüdeki $i.$ parçacığın p_{best} değerleri $P_i = (P_{i1} , P_{i2} , \dots , P_{ik} , \dots , P_{id})$'dir. $i.$ parçacığın yer değişim vektörü yani hız vektörü ise $V_i = (V_1 , V_{i2} , \dots , V_{ik} , \dots , V_{id})$ olsun. Bu tanımlamalara göre nesiller boyunca güncellenen $i.$ parçacığın hız vektörü ve buna bağlı olan konum vektörünün formülleri aşağıdaki gibidir:

$$V_{ij}^{t+1} = wV_{ij}^{t} + c_1 r_1 \left(P_{ij} - x_{ij}\right) + c_2 r_2 \left(G_{gbest} - x_{ij}\right) \tag{3.1}$$

$$X_{ij}^{t+1} = X_{ij}^{t} + V_{ij}^{t+1} \quad (3.2)$$

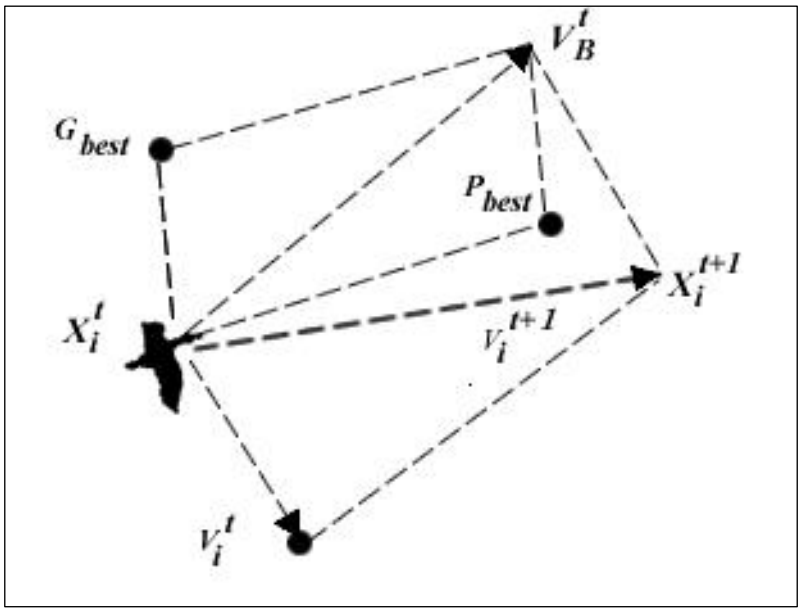
t : iterasyon sayısı; j: 1,2, …. , d;

Formül 3.1 sonucunda t. iterasyonda i. parçacığın $(t+1)$. iterasyondaki hız vektörü bulunmuş olur. Formül 3.2'de ise bulunan hız vektörü (V_{ij}^{t+1}), i. parçacığın t. iterasyondaki pozisyon vektörüne eklenerek $(t+1)$. iterasyondaki pozisyon vektörü (X_{ij}^{t+1}) bulunmuş olur. Bulunan bu pozisyon vektörü probleme yeni bir çözüm önerisi demektir. Buradaki w atalet ağırlığı değeridir. c_1 ve c_2 değerleri ise hızlandırma katsayılarıdır. r_1 ve r_2 ise [0,1] aralığında rastgele değerler alan ve PSO'nun rastgeleliğini sağlayan parametrelerdir.

$$V_{ij}^{t+1} = \underbrace{wV_{ij}^{t}}_{\text{Birinci Kısım}} + \underbrace{c_1 r_1 (P_{ij} - x_{ij})}_{\text{İkinci Kısım}} + \underbrace{c_2 r_2 (G_{gbest} - x_{ij})}_{\text{Üçüncü Kısım}}$$

Yukarıdaki ifadede görüldüğü gibi Formül 3.1 üç kısımdan oluşmaktadır [24]. Birinci kısım hız değerinin atalet ağırlığını göstermektedir. Parçacıkların hız değerlerinde ani değişiklikler olmamalı, parçacıklar bir önceki hızlarına bağlı kalarak hız güncellemesi yapmalıdırlar. Aksi takdirde parçacıklar çözüm uzayında ani yön değişiklikleri yaparak uygun bir arama gerçekleştiremeyeceklerdir. İkinci kısım ise kişisel hafıza kısmı: Bu kısım vasıtasıyla parçacıkların konumlarını geçmişte elde ettikleri en iyi konuma (p_{best}) doğru çekilirler. Bu kısımda elde edilen değer $c_1 r_1$ değeri ile ölçeklenmiştir. Formül 3.1'in üçüncü kısmı ise sosyal hafıza kısmı: Bu kısımda parçacıklar sürünün en iyi konum değerine (g_{best}) doğru çekilirler [25]. Bu kısımda da elde edilen değer $c_2 r_2$ değeri ile ölçeklenmiştir.

Hız güncellemelerine göre elde edilen yeni hız değerine $\pm V_{max}$ gibi bir kısıtlama getirilebilir. Bu V_{max} değeri sabit bir değer seçilebildiği gibi dinamik değişen bir değer de seçilebilir [24, 26]. PSO'nun bileşenleri takip eden alt bölümlerde daha ayrıntılı olarak incelenecektir. Şekil 3.1'de bir parçacığın konum değişimi görselleştirilmiştir.



Şekil 3.1. Parçacığın pozisyon değiştirmesi.

X_i^t : i. parçacığın t. iterasyondaki konumu,

V_i^t : i. parçacığın t. iterasyondaki hızı,

G_{best}: Sürüdeki en iyi konuma sahip parçacığın konumu,

P_{best} : i. parçacığın kişisel en iyi konumu,

V_B^t : t. iterasyonda G_{best} ve P_{best}'ün bileşkesi,

X_i^{t+1}: i. parçacığın $(t+1)$. iterasyondaki konumu,

V_i^{t+1}: i. parçacığın $(t+1)$. iterasyondaki hızı'dır.

3.1.1. Başlangıç Değerleri

Sürü içinde toplam p tane parçacık olduğunu kabul edelim. Bu p tane parçacığın konum ve hız değerleri aşağıdaki formüllere göre hesaplanır.

$$X_i = X_{min} + (X_{max} - X_{min}) * random( ) \tag{3.3}$$

$$V_i = V_{max} * random( ) \tag{3.4}$$

$V_{\max}$: Hız değişkeninin alabileceği en büyük değer ; $i = 1, 2 \ldots, p$

3.1.2. Konum Değeri

PSO'da her bir parçacığın pozisyon değeri ilgili probleme yeni bir çözüm önerisi demektir. $[X_{min}, X_{max}]$ çözüm aralığında değerler alan bir parçacığın konum değeri X_i belirtilen aralığın dışına çıkarsa parçacığa çeşitli kısıtlamalar getirilebilir. Bu kısıtlamalar aşağıdaki gibi tanımlanır [27].

Eğer $X_i > X_{max}$ ise $X_i = X_{max}$ ve $V_i = 0$;
Eğer $X_i < X_{min}$ ise $X_i = X_{min}$ ve $V_i = 0$;
Eğer $X_i > X_{max}$ veya $X_i < X_{min}$ ise $V_i = -V_i$
Eğer $X_i > X_{max}$ veya $X_i < X_{min}$ ise X_i'nin uygunluk değerini en kötü uygunluk değeri yap.

X_{max}: Arama uzayının üst sınırı,
X_{min}: Arama uzayının alt sınırı.

3.1.3. Hız Değeri

Hız değeri bir parçacığın çözüm uzayında arama yapmasını sağlayan en önemli etkendir. Hız değerleri pozitif ve negatif değerler alıp parçacıkların çözüm uzayında çok yönlü hareket ederek arama yapmasını sağlarlar. Hız değeri kontrol edilmediği takdirde parçacıklar çözüm alanının dışına çıkabilir ve uygun olmayan değerler bulabilir. Bunu engellemek için hız değerine V_{max} gibi kısıtlayıcı bir limit konulmuştur. V_{max} değerinin belirlenmesi probleme göre faklılık gösterebilir. Bazı uygulamalarda $V_{max} = X_{max}$ olarak kullanılmıştır [28]. Bazı uygulamalarda ise V_{max} arama uzayındaki parçacığın konum vektörünün her bir boyutunun alabileceği en büyük değer ile en küçük değerin farkının %10-20'si aralığında bir değer almıştır [27]. Yani;

$$V_{max} = (X_{max} - X_{min}) * \%R \tag{3.5}$$

R ϵ [10-20]

Eğer bir parçacığın hız değeri V_{max} limitini aşarsa aşağıdaki gibi bir kısıtlama getirilir.

Eğer $V_i > V_{max}$ ise $V_i = V_{max}$;
Eğer $V_i < -V_{min}$ ise $V_i = -V_{min}$;

3.1.4. Atalet Ağırlığı

Kennedy ve Eberhart tarafından önerilen PSO'nun ilk halinde atalet ağırlığı bulunmamaktaydı [22]. Eberhart ve Shi'nin daha sonra yaptığı bir çalışmada ise hız güncelleme formülünde birinci kısma atalet ağırlığı (inertia weight) bir çarpan olarak eklenmiştir [28]. Böylelikle, atalet ağırlığı kullanılarak parçacığın bir önceki hızının yeni hızına etkisi kontrol altına alınmış olur. Atalet ağırlığının büyük değerler alması parçacığın çözüm uzayında daha genel (global) aramalar yapmasını, p_{best}, g_{best} değerlerinden çok etkilenmeden daha araştırmacı bir yapıda çalışmasını sağlar. Atalet ağırlığının küçük değerler alması ise parçacığın çözüm uzayında daha bölgesel aramalar yapmasını, p_{best} ve g_{best} değerlerinden daha fazla faydalanarak en uygun çözüme yakınsama yapmasını sağlar.

Atalet ağırlığı, bütün arama işlemi boyunca sabit bir değer olarak kullanılabildiği gibi başlangıçta büyük değer alarak iterasyonlar ilerledikçe azaltılacak şekilde dinamik olarak kullanılabilir [28]. Dinamik kullanımda ilk iterasyonlarda parçacıklar daha genel aramalar yaparak çözüm uzayını tararken, ilerleyen iterasyonlarda arama işlemi daha ayrıntılı ve bölgesel bir hal alır. Genel olarak uygulamalarda atalet ağırlığının (w) en büyük değeri $w_{max} = 0{,}9$ en küçük değeri ise $w_{min} = 0{,}4$ olarak alınır. Aşağıdaki formülde de görüldüğü gibi iterasyon sayısı atalet ağırlığın dinamik değerini belirlemede rol alır.

$$w = w_{max} - \frac{(w_{max} - w_{min})}{iterasyon\ sayısı} * t \tag{3.6}$$

t: halihazır iterasyon

Formül 3.6'da da görüldüğü gibi başlangıçta büyük değer alan atalet ağırlığı iterasyonlar ilerledikçe daha küçük değerler almaya başlayacaktır.

3.1.5. Hızlandırma Katsayıları

Hızlandırma katsayılarından c_1 parçacıkların p_{best} değerine, c_2 ise g_{best} değerlerine doğru çekilmesini kontrol eden katsayılardır. Hızlandırma katsayılarının büyük değerler alması parçacıkların birbirinden uzaklaşıp ayrılmalarına sebep olurken, küçük değerlerler alması parçacıkların hareketlerinin kısıtlanmasına ve çözüm uzayının yeterince taranamamasına sebep olur. Hızlandırma katsayıları problemin türüne göre değişik değerler alabilir. $c_1 = c_2 = 2{,}0$ genel olarak önerilen değerdir. $c = c_1 + c_2$ iken c ne kadar büyük değer alırsa arama algoritmasının optimum değer etrafında yaptığı salınım miktarı artar. Ayrıca c_1 ve c_2 katsayıları birbirine eşit olmak zorunda değildirler, farklı değerler de alabilirler [24].

3.1.6. Uygunluk Fonksiyonu

Uygunluk fonksiyonu parçacığın pozisyon vektörünü kullanarak ve varsa problemle ilgili kısıtlamaları da göz önünde bulundurarak uygunluk değeri üreten bir fonksiyondur. Uygunluk değeri parçacığın aynı zamanda çözümün kalitesini belirler. p_{best} ve g_{best} değerleri de uygunluğu en iyi olan parçacıklardan seçilir.

3.1.7. Kişisel En İyi Değeri

Parçacıklar iterasyonlar boyunca en iyi konum değerini ararlar. Her bir iterasyonda parçacık, geçmişte bulduğu en iyi konum değeri ile iterasyonda elde ettiği yeni konum değerini kıyas eder. Eğer yeni konum değeri o ana kadarki en iyi kişisel konum değerinden daha uygun bir değere sahipse yeni kişisel en iyi konum olur. İşte parçacığın iterasyonlar boyunca, sürüdeki diğer parçacıklarla kıyas etmeksizin sadece kendi geçmişindeki değerlere bakarak bulduğu kendi en iyi değerine kişisel en iyi değer (p_{best}) denir. Aynı zamanda p_{best} değeri parçacığın geçmiş tecrübelerini gösteren bir değerdir. Formül 3.1'deki hız güncelleme denkleminde p_{best} değeri parçacığın konumunu p_{best} konumuna doğru çeker.

3.1.8. Global En İyi Değer

Sürüde iterasyonlar boyunca parçacıkların konum değerleri hesaba katıldığında bulunan en iyi uygunluğa sahip parçacığın konum değerine global en iyi değer (g_{best}) denir. Yeni iterasyonda bulunan en iyi değer g_{best}'den daha iyi uygunluğa sahipse yeni g_{best}değeri olur. Formül 3.1'deki hız güncelleme denkleminde g_{best} değeri sürüdeki bütün parçacıkların konumunu g_{best} konumuna doğru çeker.

3.1.9. Sonlandırma Kriteri

PSO'da sonlandırma kriteri kullanıcı tarafından belirtilmiş sabit bir iterasyon sayısı ya da belirlenmiş bir CPU çalışma zamanı veya son iki iterasyonun g_{best}değerleri arasındaki farkın belirli bir değerin altına düşmesi durumu olabilir.

3.2. PSO ALGORİTMASI

Standart olarak kullanılacak PSO algoritmasının sözde kodu (pseudo code);

BEGIN
p Tane Parçacığa Rastgele Hız ve Konum Değerleri Ata
REPEAT
    FOR i=1 TO p
        Uygunluk Değerini Hesapla
        P_{best}Değerini Güncelle
        g_{best} Değerinin Güncelle
        Hız ve Pozisyon Değerlerini Güncelle
    END FOR
UNTIL Sonlandırma Kriteri
END

şeklindedir. Yalın şekilde ifade edilen bu algoritmaya uygulanacak alana göre eklemeler ve uyarlamalar yapılabilir.

BÖLÜM 4

KARMA DEĞİŞKEN TİP VE KISITA SAHİP TASARIM PARAMETRELERİNİN OPTİMİZASYONU

Sun, Zeng ve Pan'ın 2009 yılında yaptıkları çalışmada kısıtlara sahip, karma tipi değişkenler içeren optimizasyon problemlerini PSO ile çözmeye çalışmışlardır [29]. Herhangi bir ceza fonksiyonu kullanmadan Uygunluk Tabanlı Kurallar (UTK) yöntemini kullanarak problem ile ilgili kısıtları sağlayan çözümler bulmuşlardır. Böylece ayrı ve hassas parametreler gerektiren ceza fonksiyonunun uygunluk fonksiyonuna entegrasyonu işlemine gerek kalmamıştır.

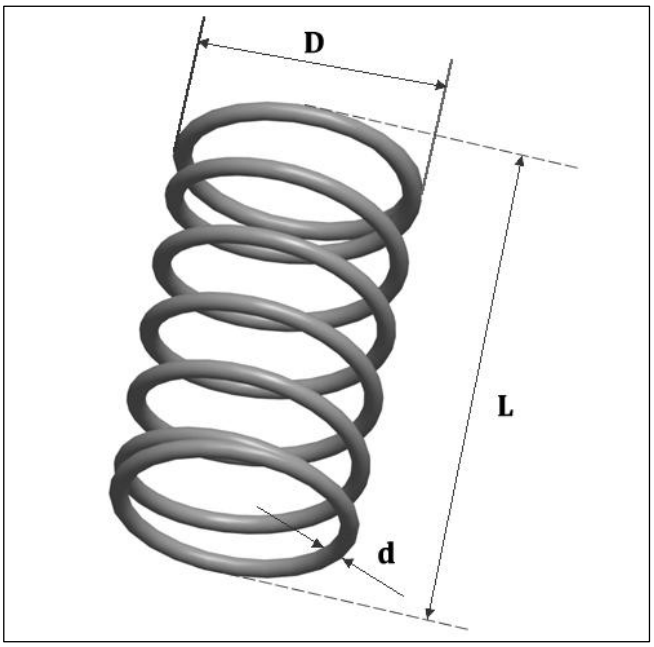
Tez çalışması kapsamında yine UTK yöntemi kullanılarak karma değişken tipleri ve kısıtlar içeren baskı yayı probleminin PSO ile çözülmesi amaçlanmaktadır. Buna karşın, ilgili literatürdeki problem formülasyonundan farklı olarak, parçacıkların her bir iterasyonda buldukları yeni çözümlerin, tanımlı oldukları aralıkların dışına çıkması durumunda tekrar sınırlar içerisine çekilmesi için önerilen, Emme (Absorbing), Yansıtma (Reflecting), Sönümleme (Damping), Görünmez (Invisible), Görünmez Yansıtma (Invisible Reflecting) ve Görünmez Sönümleme (Invisible Damping) yöntemlerinin optimizasyon problemine uygulanması ve başarım karşılaştırmalarının yapılması hedeflenmektedir.

4.1. PROBLEMİN TANIMLANMASI

İmalat ve konstrüksiyon endüstrisinde kullanılan baskı yaylarının minimum hacimde malzeme kullanılarak üretilmesini problemi çalışılacaktır. Baskı yaylarının tasarım problemi sürekli, ayrık ve tamsayı olmak üzere üç farklı değişken tipine sahip bir problemdir. Şekil 4.1'deki yayda tel çapı $d = x_1$ ayrık, telin dış çapı $D = x_2$ sürekli ve telin sarım sayısı $L = x_3$ tamsayı tipindeki değişkenlerdir. x_1 inç biriminde 42 ayrık değer alabilir.

$x_1 \in \{0{,}009\ 0{,}0095\ 0{,}0104, 0{,}0118\ 0{,}0128\ 0{,}0132\ 0{,}014\ 0{,}015\ 0{,}0162\ 0{,}0173\ 0{,}018\ 0{,}020\ 0{,}023\ 0{,}025\ 0{,}028\ 0{,}032\ 0{,}035\ 0{,}041\ 0{,}047\ 0{,}054\ 0{,}063\ 0{,}072\ 0{,}080\ 0{,}092\ 0{,}105\ 0{,}120\ 0{,}135\ 0{,}148\ 0{,}162\ 0{,}177\ 0{,}192\ 0{,}207\ 0{,}225\ 0{,}244\ 0{,}263\ 0{,}283\ 0{,}307\ 0{,}331\ 0{,}362\ 0{,}394\ 0{,}4375\ 0{,}500\}$

Bunun yanı sıra x_2, $0.6 \leq x_2 \leq 3.0$ aralığında ve x_3, $1 \leq x_3 \leq 70$ aralığında sırasıyla sürekli ve tamsayı değerler alırlar.



Şekil 4.1. Baskı yayı.

Baskı yayı tasarımında değişkenlerin değerlendirilmesi için kullanılan matematiksel modelin uygunluk fonksiyonu;

$$f(x) = \frac{\pi^2 x_2\, x_1^2 (x_3 + 2)}{4} \tag{4.1}$$

şeklindedir. Bu fonksiyon aynı zamanda bir minimizasyon fonksiyonudur.

4.2. KISITLAR

Baskı yayının tasarımı sırasında dikkat edilmesi gereken 11 ayrı kısıt vardır. Problemin çözümünde bu kısıtları sağlayan değişkenler dikkate alınır. Baskı yaylarının tasarım kısıtları:

$$g_1(x) = \frac{8\, C_f\, F_{max}\, x_2}{\pi\, x_1^3} - S \leq 0 \tag{4.2}$$

$$g_2(x) = l_f - l_{max} \leq 0 \tag{4.3}$$

$$g_3(x) = d_{min} - x_1 \leq 0 \tag{4.4}$$

$$g_4(x) = x_2 - D_{max} \leq 0 \tag{4.5}$$

$$g_5(x) = 3{,}0 - \frac{x_2}{x_1} \leq 0 \tag{4.6}$$

$$g_6(x) = \sigma_p - \sigma_{pm} \leq 0 \tag{4.7}$$

$$g_7(x) = \sigma_p + \frac{(F_{max} - F_p)}{K} + 1{,}05(x_3 + 2)x_1 - l_f \leq 0 \tag{4.8}$$

$$g_8(x) = \sigma_p + \frac{(F_{max} - F_p)}{K} \leq 0 \tag{4.9}$$

ve

$$C_f = \frac{4(x_2/x_1) - 1}{4(x_2/x_1) - 4} + \frac{0{,}615\, x_1}{x_2} \tag{4.10}$$

$$K = \frac{G x_1^4}{8 x_3 x_2^3} \tag{4.11}$$

$$\sigma_p = \frac{F_p}{K} \tag{4.12}$$

$$l_f = \frac{F_{max}}{K} + 1{,}05(x_3 + 2)x_1 \tag{4.13}$$

Yay tasarımı ile ilgili diğer tanımlamalar ise, minimum tel çapı $d_{min} = 5{,}08$ cm (0,2 inç), izin verilebilir kesme gerilimi $S = 13031{,}06$ bar (189000,0 psi), yayın azami dış çapı $D_{max} = 7{,}62$ cm (3,0 inç), malzeme kayma modülü $G = 84{,}04$ bar (1219 psi), ön yükleme altındaki izin verilebilir azami gerilme $\sigma_{pm} = 15{,}24$ cm (6,0 inç), azami serbest boy $l_{max} = 35{,}56$ cm (14,0 inç), azami iş yükü $F_{max} = 453.59$ kg (1000,0 libre), ön yükleme basma kuvveti $F_p = 136{,}07$ kg (300,0 lp), ön yükleme pozisyonundan azami yükleme pozisyonuna kadar olan gerilme $\sigma_w = 3{,}17$ cm (1,25 inç).

4.3. PSO'DA KISITLARIN ELE ALINMASI

Son yıllarda yapılan çalışmalarda özelikle mühendislik tasarımı, yerleştirme problemleri, ekonomi, kontrol vb. gibi alanlarda kısıtlamalara sahip birçok optimizasyon probleminde de PSO kullanılmaya başlanmış ve başarılı sonuçlar elde edilmiştir.

Genel olarak, kısıtlara sahip bir minimizasyon problemi şu şekilde ifade edilir [30]:

$$\begin{aligned} &min\big(f(\vec{x})\big) \\ &G_i(\vec{x}) \leq 0 \qquad i = 1, 2, \ldots, m; \\ &H_j(\vec{x}) = 0 \qquad j = 1, 2, \ldots, n; \\ &x_{d_{min}} \leq x_d \leq x_{d_{max}} \\ &d = 1, 2, \ldots, D; \end{aligned} \tag{4.14}$$

$\vec{x} = \{x_1, x_2, \ldots, x_d\}$ vektörü d boyutlu bir çözüm vektörüdür. Burada $x_d \in S \subset R$ 'dir. S, $[x_{d_{min}}, x_{d_{max}}]$ aralığındaki değerleri kapsayan bir çözüm uzaydır. R gerçel sayılar kümesidir. $x_{d_{min}}$, x_d 'nin alabileceği en küçük değer, $x_{d_{max}}$, x_d 'nin

alabileceği en büyük değerdir. Bu tanımlamada m tane $G(\vec{x})$ gibi eşitsizlik ve n tane $H(\vec{x})$ gibi eşitlik içeren kısıtlar mevcuttur.

$H(\vec{x})$ eşitlik kısıtlamaları $|H(\vec{x})| < \delta$ gibi bir eşitsizlik kısıtlamalarına dönüştürülebilir. δ çok küçük ihmal edilebilecek bir pozitif sayı seçilir. Kısıtlara uyan $\vec{x}$ vektörlerine uygun çözümler denir. Bütün uygun çözümleri içine alan kümeye F denir ve $F \subset S$ 'dir.

PSO 'da her bir parçacığın uygunluğu hesaplanırken bu kısıtlar da hesaba katılır. Üretilen bir çözümün bu kısıtlara uyup uymadığı veya ne kadar bu kısıtları aştığı şu formüle göre hesaplanabilir:

$$c(\vec{x}) = \sum_{i=1}^{m} \max\left(0, G_i(\vec{x})\right) + \sum_{j=1}^{n} abs\left(H_j(\vec{x})\right) \tag{4.15}$$

Burada $c(\vec{x})$=0 ise üretilen çözüm yani $\vec{x}$ vektörü uygun bir çözümdür. Eğer $c(\vec{x})$>0 ise üretilen çözüm kısıtları $c(\vec{x})$ değeri kadar aşmış demektir.

PSO'da kısıtlar birçok çalışmada değişik yöntemlerle [31] ele alınmıştır. En yaygın olarak kullanılan yöntemlerden bir tanesi ceza fonksiyonu uygulamasıdır. Formül 4.15'de hesaplanan $c(\vec{x})$ ceza miktarı çözümün uygunluk değerine çeşitli katsayılar kullanarak eklenebilir. Bu şekilde kısıtlar ortadan kaldırılmış olur. Yalnız ceza fonksiyonu uygunluk fonksiyonuna nasıl dahil edileceği, uygulanacak parametrik değerlerin çok hassas olması ve uygulamadan uygulamaya farklılık göstermesi ceza fonksiyonunun uygulanabilirliğini zorlaştıran etkenlerdir. Uygulanan yöntemlerden bir tanesi de tamamen kısıtlara uyan çözümlerden oluşan bir nesil oluşturup, bütün iterasyonlar boyunca bu uygunluğu devam ettirmektir. Bu yöntemde de iterasyonlar boyunca oluşan yeni nesillerde kısıtlara uymayan yeni çözümler oluşabilir. Bu durumda kısıtlara uymayan parçacıkların uygunluk değerlerine p_{best} değerleri atanarak veya bir önceki iterasyondaki uygun çözüm değeri atanarak, parçacıkların uygun bölgede kalmaları sağlanabilir. Bu yöntemin en büyük olumsuzluğu sürüyü başlatırken uygun bir başlangıç nesli oluşturamama ihtimalidir. Bu işlem çok uzun

zaman alabilmekte hatta bazı problemlerde ise başlangıç nesli oluşturmak mümkün olmamaktadır.

PSO'da kısıtlamaların ele alınışı ile ilgili diğer bir yöntem ise Uygunluk Tabanlı Kurallar (Feasibilty Based Rules) yöntemidir [29]. Bu tez çalışmasında da kullanılacak bu yöntem üç temel kuraldan oluşmaktadır.

- Kısıtlara uyan çözümler, kısıtlara uymayanlara tercih edilir.
- Kısıtlara uyan iki çözüm arasından uygunluğu daha iyi olan çözüm tercih edilir.
- Kısıtlara uymayan iki çözüm arasından kısıtları aşma miktarı daha az olan tercih edilir.

Bu üç kural p_{best} değerinin güncellemesinde kullanılacaktır. Bu yöntemi daha detaylandırılacak olursa:

$P_i, t.$ iterasyondaki $i.$ parçacığın p_{best} değerini ifade etsin. $\vec{x}_i(t+1)$ ise i. parçacığın $(t+1).$ iterasyondaki pozisyon vektörünü ifade etsin. $f(\vec{x}_i)$ problemin uygunluk fonksiyonu olmak üzere aşağıda belirtilen şu üç durumda:

- Eğer $\vec{P}_i$ ilgili problemin kısıtlarına uymayan bir değer iken $\vec{x}_i(t+1)$ kısıtlara uyan bir değer ise,
- Eğer $\vec{P}_i$, $\vec{x}_i(t+1)$'den her ikisi de kısıtlara uygun değerler iken $f\big(\vec{x}_i(t+1)\big) < f(\vec{P}_i)$ ise,
- Eğer $\vec{P}$i , $\vec{x}_i(t+1)$'den her ikisi de kısıtlara uymayan değerler iken $c(\overrightarrow{x_i}(t+1)) < c\big(\overrightarrow{P_i}\big)$ ise,

$\vec{P}$i' ye $\vec{x}_i(t+1)$ değeri atanır.

İterasyonlar boyunca devam eden bu işlemler sonucunda kısıtlara uygun en iyi çözüm bulunmuş olacaktır.

4.4. PSO'DA KARMA DEĞİŞKEN TİPLERİNİN ELE ALINMASI

PSO öncelikle sürekli (continous) değişkenler üzerinde sonuç veren bir optimizasyon tekniği olarak ortaya çıkmıştır. Fakat gerçek hayatta optimize edilmesi gereken değerler her zaman sürekli değişkenler olmamaktadır. Bazen bu değişkenler karşımıza tam sayı değişkenleri, ayrık (discrete) değişkenler ya da {0,1} değerleri alan ikilik değişkenler olarak çıkmaktadır. PSO'da farklı tipteki değişkenleri ele almak için değişik yöntemler denenmiştir. Yapılan çalışmada Karma Değişkenler yöntemi kullanılacaktır [31].

Karma Değişkenler yöntemine göre sürekli değişkenler, başlangıçta belirtilen sınırlar içersinde üretilir ve iterasyonlar boyunca bu sınırları aşmadığı sürece herhangi bir özel işleme tabi tutulmazlar. Tam sayı değişkenleri ise, başlangıçta belirtilen sınırlar içersinde reel sayı olarak yani sürekli sayı olarak üretilir. İterasyonlar boyunca Formül 3.1 ve Formül 3.2'deki hız ve pozisyon güncellemelerinde reel sayı olarak işlem görürler. Sadece uygunluk fonksiyonunda uygunluk değeri hesaplanırken bu reel sayılar kendisinden küçük en yakın tamsayıya yuvarlanarak işlem görürler. Bu yöntem ikilik sayı değişkenleri içinde kullanılabilir. Ayrık değişkenler için ise, değişkenin değerleri değil indisleri ele alınarak işlem yapılır. Ayrık değişken dizisindeki her bir eleman bir indise sahiptir. İterasyonlar boyunca bu indis Formül 3.1 ve Formül 3.2'deki hız ve pozisyon formüllerine göre güncellenir. Bu yöntemde dizinin indisleri reel sayı olarak tanımlanmalıdır. Bu indisler uygunluk değeri hesaplanırken kendisinden küçük en yakın tam sayıya yuvarlanarak ilgili ayrık değerin uygunluk fonksiyonunda kullanılmasını sağlarlar. Şöyle bir tanımlama yapılırsa:

$$\overrightarrow{x_i} = \begin{bmatrix} \overrightarrow{x_i^C} \\ \overrightarrow{x_i^D} \\ \overrightarrow{x_i^I} \\ \overrightarrow{x_i^B} \end{bmatrix} \text{ ve } \begin{matrix} x_{ij}^{C_{altlimit}} < x_{ij}^{C} < x_{ij}^{C_{\text{üstlimit}}} & j = 1, 2, \ldots\ldots, n_C \\ x_{ij}^{D_{altlimit}} < x_{ij}^{D} < x_{ij}^{D_{\text{üstlimit}}} & j = 1, 2, \ldots\ldots, n_D \\ x_{ij}^{I_{altlimit}} < x_{ij}^{I} < x_{ij}^{I_{\text{üstlimit}}} & j = 1, 2, \ldots\ldots, n_I \\ x_{ij}^{B_{altlimit}} < x_{ij}^{B} < x_{ij}^{B_{\text{üstlimit}}} & j = 1, 2, \ldots\ldots, n_B \end{matrix} \tag{4.16}$$

$\overrightarrow{x_i}$ sürekli, ayrık, tam sayı ve ikilik değişkenlerden oluşan bir vektördür. x_{ij} sürüdeki

i. parçacığın j. boyuttaki değerini ifade etmektedir. x_{ij}^{C} sürekli tipteki değişkenleri, x_{ij}^{D} ayrık tipteki değişkenleri, x_{ij}^{I} tam sayı tipindeki değişkenleri, x_{ij}^{B} ise ikilik sayı tipindeki değişkenleri temsil etmektedir. $x_{ij}^{altlimit}$ ve $x_{ij}^{üstlimit}$ ilgili değişkenlerin çözüm uzayında alt ve üst limitleri gösteren değerlerdir. X^{C} sürekli değişkenlerin aldığı değerler kümesi, X^{D} ayrık değişkenlerin aldığı değerler kümesi, X^{I} tam sayı olan değişkenlerin aldığı değerler kümesi, X^{B} ikilik değişkenlerin aldığı değerler kümesidir.

Buna göre değişik tipteki değişkenlere uygulanacak PSO'da :

$$x_i = \begin{cases} x_{ij}: x_{ij} \in X^{C}; & j = 1, 2, \dots, n_c \\ int(x_{ij}): x_{ij} \in X^{I}; & j = 1, 2, \dots, n_I \\ x_{i,int(j)}: x_{i,int(j)} \in X^{D}; & j \in [1, n_d + 1) \\ int(x_{ij}): x_{ij} \in X^{\mathrm{B}}; & j = 1, 2, \dots, n_B \end{cases} \tag{4.17}$$

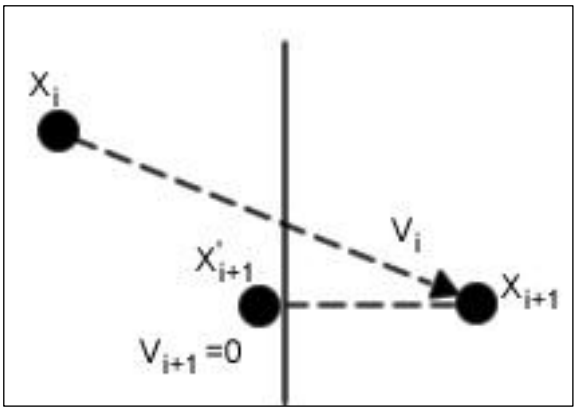
şeklinde bir yöntem ile değişkelerin değerleri hesaplanacaktır.

4.5. PSO'DA ÇÖZÜM UZAYININ SINIRLANDIRILMASI

Optimizasyon problemlerinde genellikle değişkenlerin çözüm uzayı sınırlara sahiptir. Parçacıklar bu sınırlar içerisinde kalarak en uygun çözümü aramak zorundadırlar. Çözüm uzayının alt sınır X_{min}, üst sınır X_{max}, hız değişkeninin alabileceği en büyük değer V_{max} olmak üzere başlangıçta her bir parçacığa Formül 3.3 ve Formül 3.4' e göre pozisyon ve hız değerleri atanır. Parçacıkların her bir iterasyonda buldukların yeni çözümlerin bu sınırlar içerisinde olması gerekir. Formül 3.1 ve Formül 3.2' ye göre hesaplanan yeni konum değerleri bu sınırların dışına taşmış olabilir. Bu durumda parçacığı tekrar sınırlar içerisine çekmek için değişik yöntemler kullanılmaktadır [27, 32]. Bu yöntemlerden bazıları Emme (Absorbing), Yansıtma (Reflecting), Sönümleme (Damping), Görünmez (Invisible), Görünmez Yansıtma (Invisible Reflecting) ve Görünmez Sönümleme (Invisible Damping) yöntemleridir.

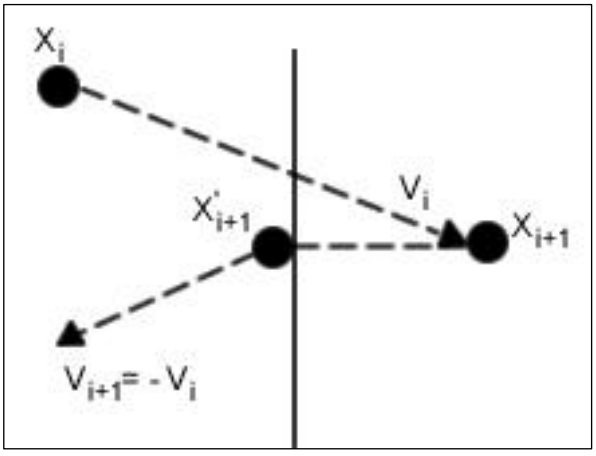
Emme Yöntemi: Bu yöntemde parçacık belirlenen sınırların dışına çıktığında konum

değeri sınıra getirilir. Yine parçacığın hız değeri sıfıra eşitlenir. Emme yöntemi Şekil 4.2'de ifade edilmiştir.



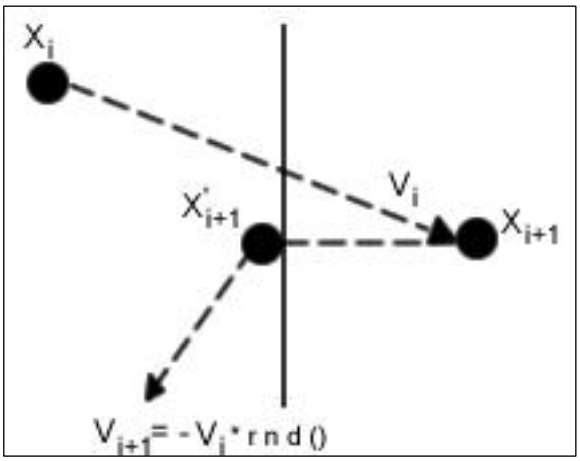
Şekil 4.2. Emme yöntemi.

Yansıtma Yöntemi: Parçacığın sınırları aşan konum değeri Emme yöntemindeki gibi sınıra çekilir, farklı olarak parçacığın hız değerinin işareti değiştirilir. Yansıtma yöntemi Şekil 4.3'de ifade edilmiştir.



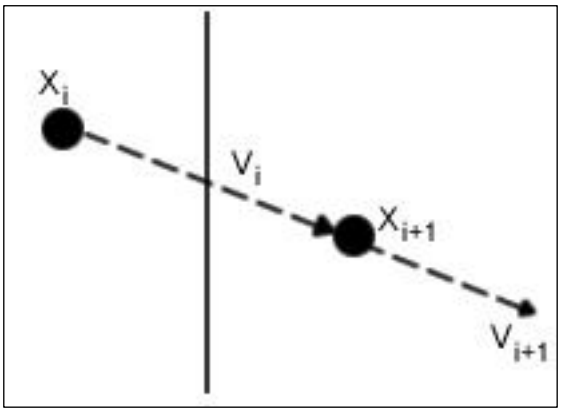
Şekil 4.3. Yansıtma yöntemi.

Sönümleme Yöntemi: Parçacığın sınırları aşan konum değeri Emme ve Yansıtma yönteminde olduğu gibi sınıra çekilir, hız değeri ise işareti değiştirilerek [0,1] aralığındaki rastgele bir sayı ile çarpılır. Sönümleme yöntemi Şekil 4.4.'de ifade edilmiştir.



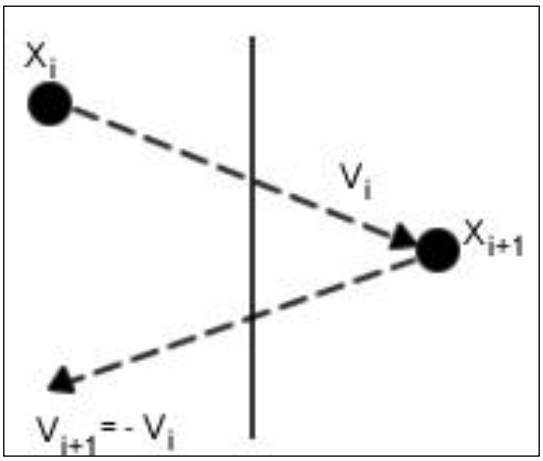
Şekil 4.4. Sönümleme yöntemi.

Görünmez: Parçacığın konum değerlerinin sınırları aşmasına izin verilirken hız değeri üzerinde de herhangi bir değişiklik yapılmaz. Yalnız sınırı aşan parçacığa kötü bir uygunluk değeri atanarak p_{best} ve g_{best} olması engellenir. Bu parçacığın hız güncellemesinde p_{best} ve g_{best}değerlerinin etkisinde kalarak tekrar çözüm uzayının sınırları içersine getirilmesi sağlanır. Görünmez yöntemi Şekil 4.5.'de ifade edilmiştir.



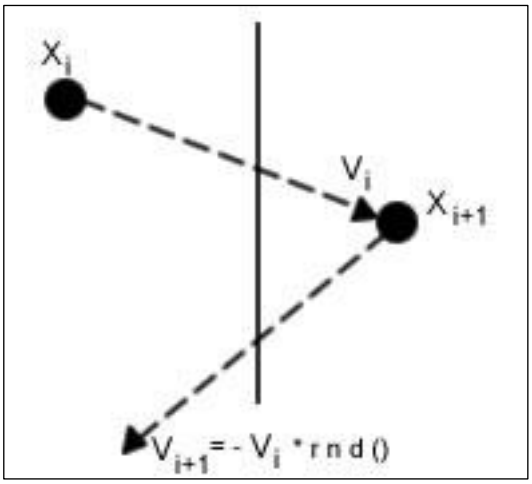
Şekil 4.5. Görünmez yöntemi.

Görünmez Yansıtıma: Bu yöntemde de parçacığın konum değerlerinin sınırları aşmasına izin verilirken parçacığa kötü bir uygunluk değeri atanarak p_{best} ve g_{best}olması engellenir. Yanlız parçacığın hız değerinin işareti değiştirilir. Görünmez Yansıtıma yöntemi Şekil 4.6.'da ifade edilmiştir.



Şekil 4.6. Görünmez Yansıtma yöntemi.

Görünmez Sönümleme: Bu yöntemde de Görünmez Yansıtma yönteminde olduğu gibi konum değerlerinin sınırları aşmasına izin verilirken parçacığa kötü bir uygunluk değeri atanarak p_{best}ve g_{best}olması engellenir. Sadece parçacığın hız değerinin işareti değiştirilerek [0,1] aralığındaki rastgele bir sayı ile çarpılır. Görünmez Sönümleme yöntemi Şekil 4.7'de ifade edilmiştir.



Şekil 4.7. Görünmez Sönümleme yöntemi.

Bu yöntemlerden Emme, Yansıtma, Sönümleme yöntemleri genel olarak sınırlayıcı yöntemler, Görünmez, Görünmez Yansıtma ve Görünmez Sönümleme yöntemleri ise sınırlayıcı olmayan yöntemler olarak adlandırılmaktadır.

X_i: Parçacığın i. iterasyondaki konumu,

V_i: Parçacığın i. iterasyondaki hızı,

X_{i+1}: Parçacığın $(i+1)$. iterasyondaki konumu,

V_{i+1}: Parçacığın $(i+1$. iterasyondaki hızı,

X'_{i+1}: Parçacığın $(i+1)$. iterasyonda sınırlandırıcı yöntemler ile güncellenmiş konumu göstermektedirler.

4.6. DEĞİŞTİRİLEN PSO ALGORİTMASI

Tez çalışmasında kullanılacak PSO'da standart PSO'dan farklı olarak hem kısıtlar hem de farklı tipte değişkenler olacaktır. Hem kısıtlar içeren hem farklı tipte değişkenlere sahip uygulamalar için geliştirilen PSO'nun sözde kodu şöyle olacaktır:

BEGIN
FOR i=1 TO p
    Pozisyon Değerlerini Rastgele Ata (Konu 4.4);
    Hız Değerlerini Rastgele Olarak Ata
    Değişkenlerin Kısıtları Aşma Miktarını $C(\vec{x})$ Hesapla
    Uygunluk Değerini Hesapla
    p_{best} Olarak Parçacığın İlk Konum Değerini Ata
END FOR
REPEAT
    Kısıtlara Uygun Olan En İyi p_{best} Değerini g_{best} Olarak Ata
    FOR i=1 TO p
        Hız ve Pozisyon Değerlerini Güncelle
        Farklı Değişken Tipleri İçin Formül 4.17.'yi Uygula
        Değişkenlerin Kısıtları Aşma Miktarını $C(\vec{x})$ Hesapla
        Uygunluk Değerini Hesapla
        p_{best} Güncelle (Konu 4.3)
    END FOR
UNTIL Sonlandırma Kriteri
END

4.7. PROBLEM UYGULAMASI

Tez kapsamında ilgili problem için uygulanacak PSO'da tanımlamalar ise şu şekildedir: Sürünün eleman sayısı (populasyon) $p = 50$, iterasyon sayısı $t_{max} = 500$, V_{max} ilgili değişkenin tanım aralığının %50'si, $c_1 = 2{,}0$ ve $c_2 = 2{,}0$ iken w ilk iterasyonda $w_{max} = 0{,}9$ değerine sahipken son iterasyonda $w_{min} = 0{,}4$ değerine kadar dinamik olarak düşürülmektedir. w'nin her iterasyondaki değeri Formül 2.6'a göre hesaplanmaktadır. Baskı yayının tasarımı problemini çözmek için geliştirilen kod Microsoft Visual Studio 2008 C# ortamında geliştirilmiştir. Geliştirilen bu kod Window 7 Professional 32 bit işletim sistemin sahip Intel Core2 Duo 2,63 GHz işlemcili ve 4 GB RAM'li bir dizüstü bilgisayar üzerinde çalıştırılmıştır.

Program çözüm uzayını sınırlandırıcı her bir yöntem için 20'şer defa çalıştırılmış ve sonuçların ortalamaları alınmıştır. Her bir yöntem için elde edilen sonuçlar Çizelge 4.1' de gösterilmiştir.

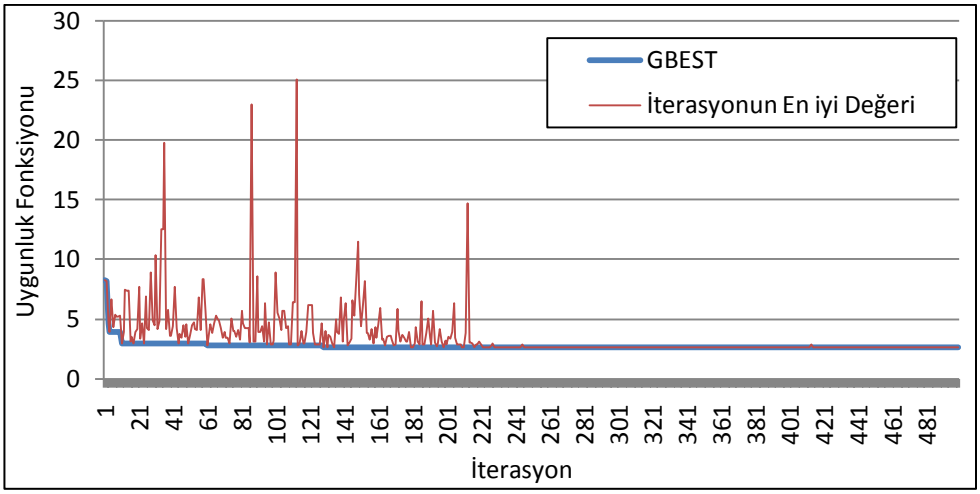
Çizelge 4.1. Yöntemlerin uygunluk değerlerinin kıyaslanması.

Yöntem	Referans [30]	En İyi	En Kötü	Ort.	Ort. Sonuç Bulma Süresi(sn)	$\frac{\text{Ort.} - \text{En İyi}}{\text{En İyi}}$	$\frac{\text{Ort} - \text{En Kötü}}{\text{En Kötü}}$
Emme	2,658	2,658	2,810	2,686	0,872	0,010	-0,043
Yansıtma	2,658	2,658	2,938	2,693	1,236	0,012	-0,083
Sönümleme	2,658	2,658	2,699	2,668	0,611	0,003	-0,011
Görünmez	2,658	2,658	2,909	2,681	0,869	0,008	-0,078
Görünmez Yansıtma	2,658	2,997	5,856	4,066	3,237	0,356	-0,305
Görünmez Sönümleme	2,658	2,658	2,905	2,755	1,346	0,036	-0,051

Çizelge 4.1' de görüldüğü gibi Referans değeri 2,658 olarak alınmıştır [30].

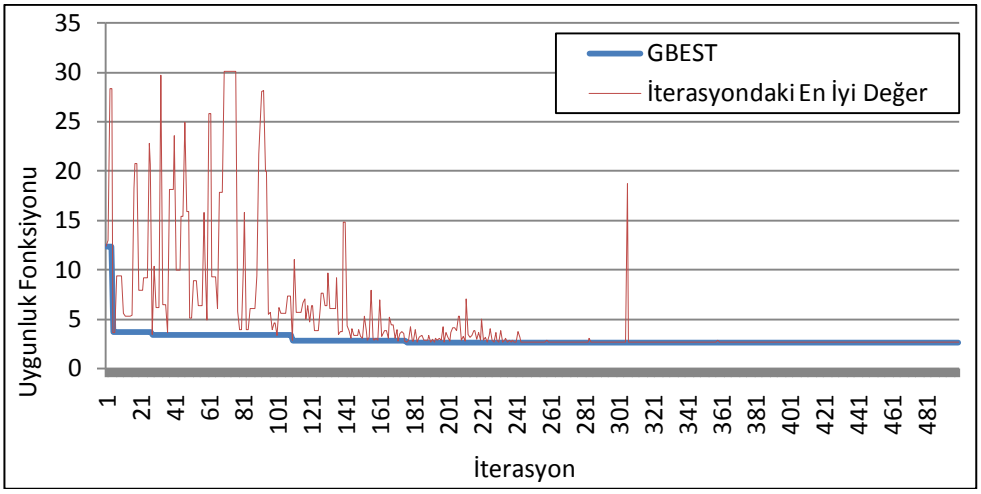
Emme Yöntemi: 20 defa çalıştırılan program optimum değer olan 2,658 değerini 14 kez bulurken, 20 sonucun ortalaması olarak 2,686 sonucunu bulmuştur. Bu yöntem ile yaklaşık 130. iterasyon sonucunda ortalama 0,872 saniyede optimum sonuç olan

2,658 değerine ulaşılmıştır. Emme yönteminde genel olarak hızlı bir yakınsama işlemi gerçekleşmiş ve bu problem için kabul edilebilir bir yöntem olduğu ortaya çıkmıştır. Bu yöntem için iterasyonlar boyunca elde edilen uygunluk değerleri Şekil 4.8'de ifade edilmiştir.



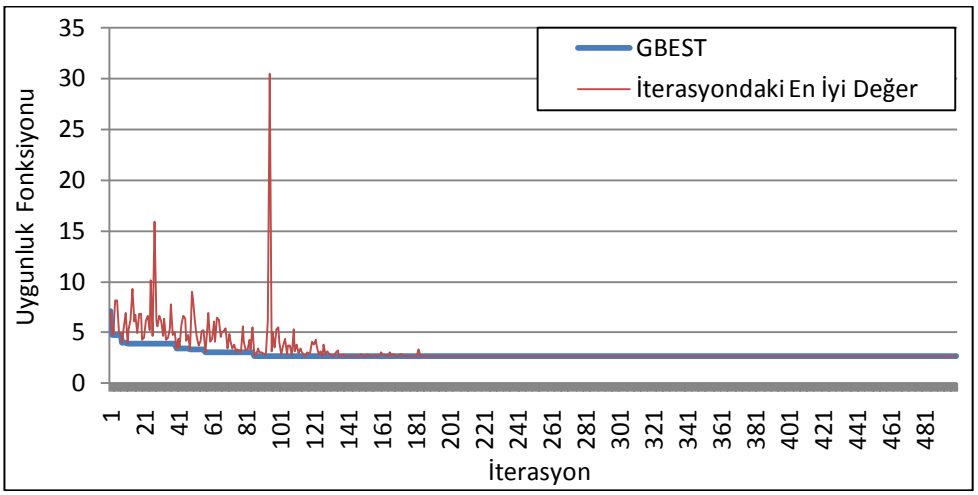
Şekil 4.8. Emme yöntemi uygunluk değerleri.

Yansıtma Yöntemi: 20 defa çalıştırılan program optimum değer olan 2,658 değerini 9 kez bulurken, 20 sonucun ortalaması olarak 2,693 sonucunu bulmuştur. Bu yöntem ile yaklaşık 180. iterasyon sonucunda ortalama 1,236 saniyede optimum sonuç olan 2,658 değerine ulaşılmıştır. Bu yöntem ilk 100 iterasyonda gerekli yakınsamayı yapamamış ve kısıtlara uymayan sonuçlara takılmıştır. 100. iterasyonda sonra ise gerekli yakınsama gerçekleşmiş ve optimum sonuç bulunmuştur. Bu olumsuz duruma rağmen bu yöntemin bu problem için kullanılabilir olduğu ortaya çıkmıştır. Bu yöntem için iterasyonlar boyunca elde edilen uygunluk değerleri Şekil 4.9'da ifade edilmiştir.



Şekil 4.9. Yansıtma yöntemi uygunluk değerleri.

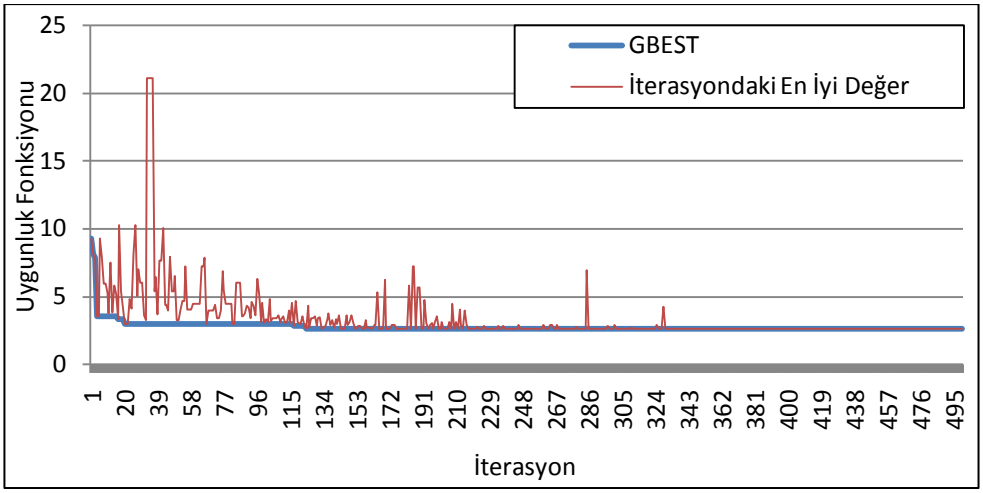
Sönümleme: 20 defa çalıştırılan program optimum değer olan 2,658 değerini 14 kez bulurken, 20 sonucun ortalaması olarak 2,668 sonucunu bulmuştur. Bu yöntem ile yaklaşık 80. iterasyon sonucunda ortalama 0,611 saniyede optimum sonuç olan 2,658 değerine ulaşılmıştır. Sönümleme yönteminin diğer yöntemlere göre daha hızlı yakınsama yaptığı ve bu problem için kabul edilebilir en uygun ve en hızlı sonuç veren yöntem olduğu ortaya çıkmıştır. Bu yöntem için iterasyonlar boyunca elde edilen uygunluk değerleri Şekil 4.10'da ifade edilmiştir.



Şekil 4.10. Sönümleme yöntemi uygunluk değerleri.

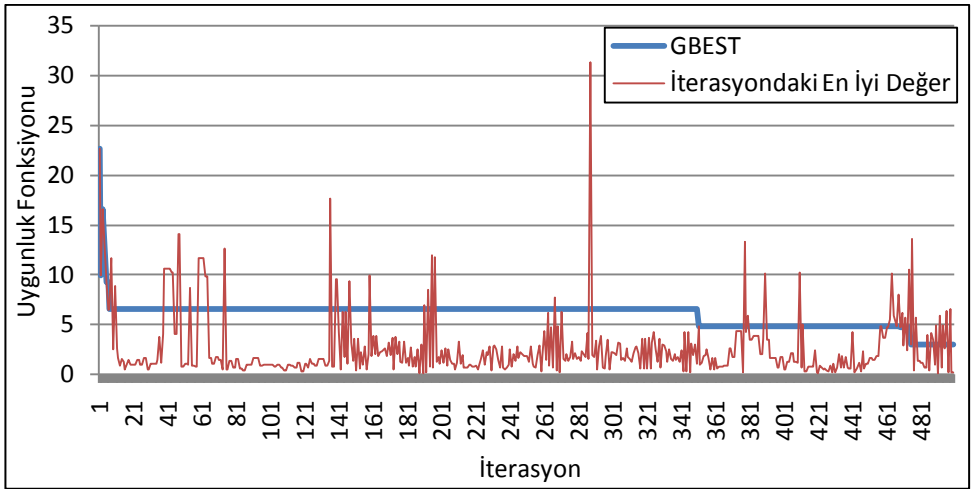
Görünmez: 20 defa çalıştırılan program optimum değer olan 2,658 değerini 14 kez bulurken, 20 sonucun ortalaması olarak 2,681 sonucunu bulmuştur. Bu yöntem ile

yaklaşık 130. iterasyon sonucunda ortalama 0,869 saniyede optimum sonuç olan 2,658 değerine ulaşılmıştır. Görünmez yönteminde genel olarak hızlı bir yakınsama işlemi gerçekleşmiş ve bu problem için kabul edilebilir bir yöntem olduğu ortaya çıkmıştır. Bu yöntem için iterasyonlar boyunca elde edilen uygunluk değerleri Şekil 4.11'de ifade edilmiştir.



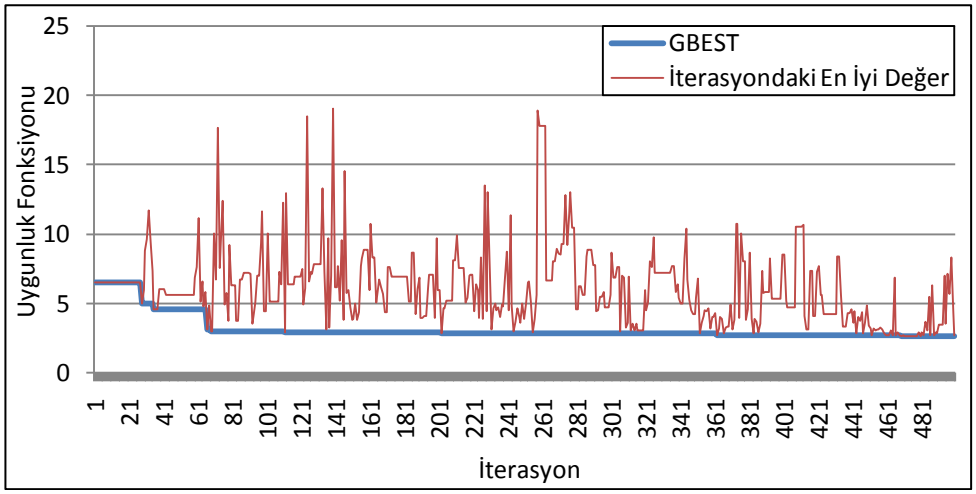
Şekil 4.11. Görünmez yöntemi uygunluk değerleri.

Görünmez Yansıtma: 20 defa çalıştırılan program optimum değer olan 2,658 değerini hiç bulamamış ve 20 sonucun ortalaması olarak 4,066 sonucunu bulmuştur. Bu yöntem ile yaklaşık 470. iterasyon sonucunda ortalama 3,237 saniyede en iyi sonuç olarak optimuma uzak olan 4,066 değeri bulunmuştur. Bu problem için bu yöntemin PSO'ya uygun olmadığı ve gerekli yakınsamayı yapamadığı görülmüştür. Grafiğe bakıldığı zaman iterasyondaki en iyi sonuçlarının global en iyi değerden daha düşük olduğu görülmektedir. Bu durumun oluşması, sürüdeki bütün parçacıkların kısıtlara uymayan sonuçlara sahip olması sebebiyledir. Bu durumda iterasyonun en iyi değeri g_{best} değerin altında bir değer olabilmektedir. Yalnız program bu kısıtlara uymayan sonuçların g_{best} olmasını engellemektedir. Şekil 4.12'nin farklığının en büyük sebebi bu yöntem uygulandığında sürüdeki parçacıkların kısıtlara uyan bölgeye çekilememesi ve kısıtlara uymayan bölgede kalmasıdır.. Bu yöntem için iterasyonlar boyunca elde edilen uygunluk değerleri Şekil 4.12'de ifade edilmiştir.



Şekil 4.12. Görünmez Yansıtma yöntemi uygunluk değerleri.

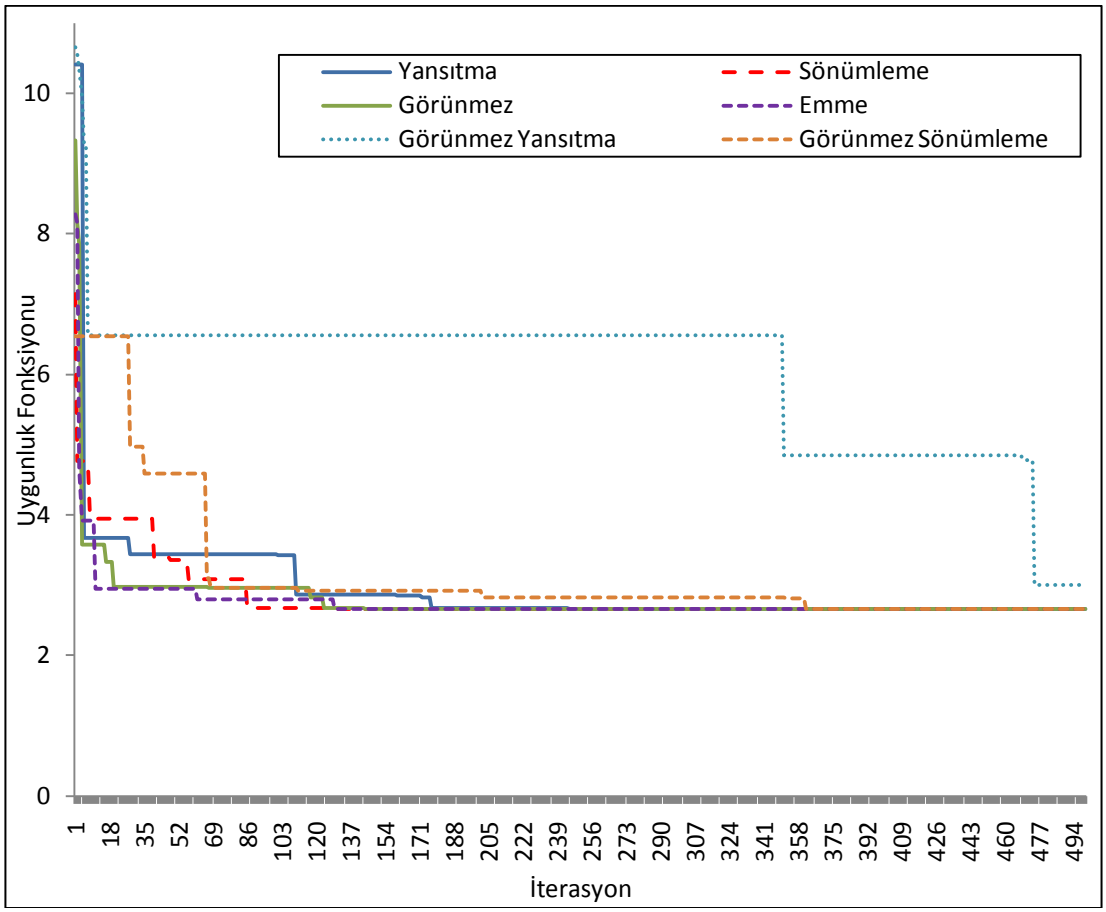
Görünmez Sönümleme: 20 defa çalıştırılan program optimum değer olan 2,658 değerini 3 kez bulunurken, 20 sonucun ortalaması olarak 2,755 sonucunu bulmuştur. Bu yöntem yaklaşık 200. iterasyon 1,346 saniyede sonucunda ortalama optimum değer olan 2,658 değerine ulaşılmıştır. Yalnız bu problem için bu yöntemin PSO'ya uygun olmadığı ve gerekli yakınsama yapamadığı ortaya çıkmıştır. Bu yöntem için iterasyonlar boyunca elde edilen uygunluk değerleri Şekil 4.13'de ifade edilmiştir.



Şekil 4.13. Görünmez Sönümleme yöntemi uygunluk değerleri.

Sonuç olarak baskı yayı tasarım problemine yukarıda bahsettiğimiz altı yöntemi uyguladığımızda farklı değerler elde ettiğimiz görülmektedir. Şekil 4.14'de bütün yöntemlerin bir arada kıyaslamasının yapıldığı grafik görülmektedir. Elde edilen

sonuçlarına göre bu problem için en uygun yöntemin Sönümleme yöntemi olduğu görülmektedir. Sönümleme yöntemi optimum sonucu bulma ve yakınsama hızı olarak en güvenilir sonuçları vermiştir. Bunun yanı sıra Emme ve Görünmez yöntemleri de sonucu bulma hızı ve yakınsama hızı olarak Sönümleme yöntemi kadar hızlı olmasalar da optimum sonuç bulma konusunda gayet başarılıdırlar. Yansıtma yöntemi ilk iterasyonlarda uygun olmayan bölgeye takılmasına rağmen ileriki iterasyonlarda optimum sonucu bulmayı başarmıştır. Görünmez Sönümleme yöntemi ise optimum sonucu bulmasına rağmen son iterasyonlarda dahi gerekli yakınsamayı sağlayamamış ve sık sık uygun olmayan bölgelere takılmıştır. Görünmez Yansıtma yöntemi ise hem optimum sonuçtan oldukça uzak kalmış hem de uygun olmayan bölgelere takılarak gerekli olan yakınsamayı gerçekleştirememiştir. Genel olarak bakıldığında sınırlayıcı yöntemlerin sınırlayıcı olmayan yöntemlere bir üstünlüğü söz konusudur.



Şekil 4.14. Çözüm uzayını sınırlandıran yöntemlerin kıyaslanması.

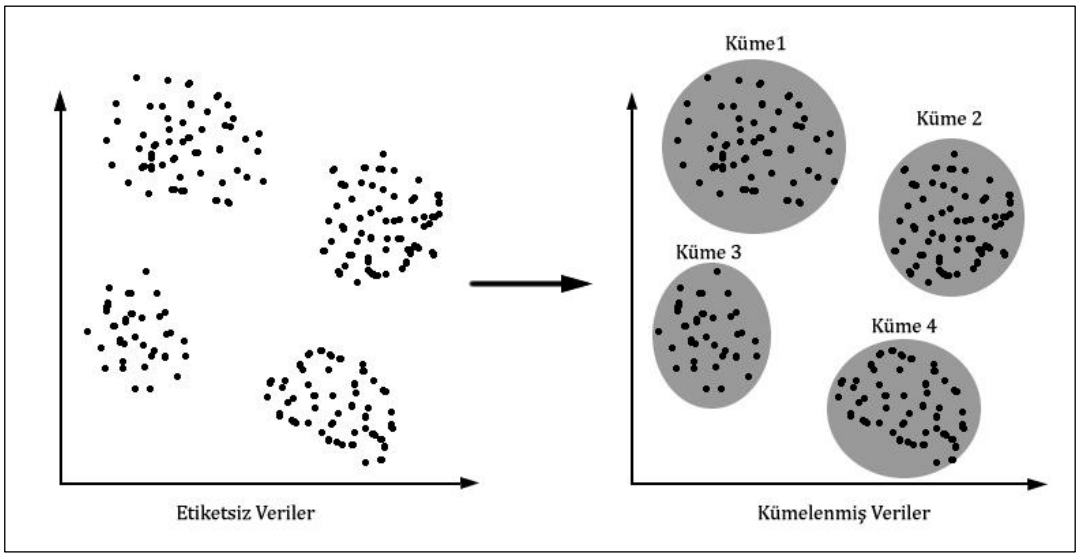
BÖLÜM 5

PARÇACIK SÜRÜ OPTİMİZASYONU İLE KÜMELEME

Bu bölümde PSO kümeleme algoritması olarak kullanılacak ve zambak çiçeği verilerinin kümelenmesi işlemi gerçekleştirilecektir. Öncelikle PSO ile zambak çiçeğinin küme sayısı bulunmaya çalışılacaktır. Daha sonra bulunan küme sayısı bir ön bilgi olarak kullanılarak yine PSO ile kümeleme dağılım işlemi gerçekleştirilecektir. Ayrıca Fuzzy C-Means kümeleme algoritması ile PSO algoritması hibrit bir şekilde kullanılarak birlikte zambak çiçeği verileri kümelenmeye çalışacaklardır.

5.1. KÜMELEME

Kümeleme, herhangi bir etikete sahip olmayan veri topluluklarının bir öğretici olmadan farklı gruplara ayrılmasıdır. Bu gruplar kendi içinde benzer özelliklere sahip veriler içerirken, oluşturulan diğer veri gruplardan da farklı özellikler gösterirler. Kümelemenin amacı n tane elemandan oluşan bir veri setini k tane veri kümesine bölmektir. Bu ayrıştırma işlemi yapılırken k tane kümenin birbirinden maksimum derecede farklılık göstermesi ve kendi içlerinde ise maksimum derecede benzerlik göstermesi istenir [33, 34]. Kümeleme algoritmaları makina öğrenmesi, veri madenciliği, veri analizi ve birçok mühendislik alanında yaygın olarak kullanılmaktadır. Şekil 5.1.'de bir çok sayıda elemandan oluşan bir veri seti mesafeye dayalı benzerlik ölçütüne göre dört farklı kümeye ayrılmıştır.



Şekil 5.1. Kümelenmiş veriler.

Kümeleme algoritmaları hiyerarşik yada bölümsel (partitional) yapıda olabilir. Hiyerarşik kümeleme yöntemlerinde çıktılar, herbiri farklı bir kümeyi ifade eden ağaç dalları şeklindedir.Hiyerarşik kümeleme algoritmalarının üstünlükleri:

- Küme sayısının önceden belirtilmesine ihtiyaç duymamaları,
- Başlangıç koşullarından bağımsız olmalarıdır.

Bu algoritmaların olumsuzlukları ise;

- Statik olması yani herhangi bir kümeye ait olan bir verinin farklı bir kümeye aktarılamaması,
- İç içe girmiş farklı veri kümelerinin ayırt edilememesidir.

Bölümsel kümeleme algoritmaları ise verileri birbirinden bağımsız kümelere ayırırlar ve bazı ölçütlere göre (örneğin kare hata fonksiyonuna göre) kümeleme problemini minimizasyon problemine dönüştürürler. Bu yönüyle de bir optimizasyon problemi olarak ele alınabilirler [35].

Kümeleme algoritmalarında küme elemanlarının kümelere aidiyeti yönünden iki farklı algoritma kullanılabilir. Bunlar bulanık (fuzzy) ve kesin (crisp) kümeleme

algoritmalarıdır. Kesin kümeleme algoritmasında veri setindeki bir veri ancak bir kümeye ait olabilir. Bulanık kümeleme algoritmasında ise veri belirlenen bulanık üyelik derecesi ile birden fazla kümeye üye olabilir [33, 36].

Kümeleme algoritmaları yöneticili (supervised) ve yöneticisiz (unsupervised) olmak üzere iki grupta değerlendirilir. Yöneticili kümeleme algoritmalarında hangi verilerin hangi kümelere ait olacağı konusunda bir öğretici tarafından yönlendirme yapılır. Yöneticisiz kümeleme algoritmalarında ise bir öğreticiye ihtiyaç duymadan veriler farklı özelliklerine göre gruplandırılırlar [37].

Daha önce de ifade edildiği gibi bölümsel kümeleme yöntemleri birer optimizasyon yöntemi olarak ele alınabilir. Amaç verilerin ayrılacağı grup sayısını ve her bir kümenin ağırlık merkezini bulmaktır. Verilerin gruplandırılması sırasında üç temel nokta ele alınır:

- Verilerin gruplandırılacağı optimum küme sayısının belirlenmelidir. Verilerin doğru gruplandırılmasının yanı sıra grup sayısının belirlemesi de kümeleme işleminde değerlendirmeye alınır.
- Verilerin gruplandırılması sırasında kullanılacak benzerlik ölçütünün belirlenmesidir. Bu benzerlik ölçütü aynı küme içerisndeki verilerin maksimum oranda benzerlik göstermesini sağlarken diğer gruptaki verilerle maksimum oranda farklılık göstermesini sağlamalıdır. En yaygın olarak kullanılan benzerlik ölçütü mesafeye dayalı benzerliktir.
- Kümeleme işlemine en hızlı şekilde gerçekleştirecek yöntemin belirlenmesidir.

Kümeleme problemlerin çözümünde birçok algoritma kullanılırken bunlardan K-Means [35, 36, 38, 39], Fuzzy C-Means yerel öğrenme algoritmaları [40, 41] en fazla kullanılan algoritmalardandır. Yalnız bu yerel öğrenme algoritmalarının yerel minimumlara takılma riskleri fazladır. Bu problemin üstesinden gelebilmek için GA [42, 43, 44] ve PSO gibi evrimsel algoritmalar kullanılmıştır [34, 45, 46].

Tezin bu bölümünde yapılacak uygulamalarda bölümsel kümeleme problemlerinin

PSO algoritması kullanılarak, yöneticili kesin kümeleme, yöneticisiz kesin kümeleme ve yöneticili bulanık kümeleme yöntemleriyle çözülmesi hedeflenmektedir.

5.2. PSO İLE KÜMELEME PROBLEMLERİN ÇÖZÜMÜ

5.2.1. Problem Tanımı

Farklı kümelere ayrılacak n tane elemandan oluşan veri setini $P = \{\overrightarrow{P_1}, \overrightarrow{P_2}, \overrightarrow{P_3}, \dots, \overrightarrow{P_n}\}$ şeklinde ifade edelim. Veri setindeki her bir elemanı temsil eden P_i, d boyutlu bir vektördür. $\overrightarrow{P_i} = \{p_{i,1}, p_{i,2}, p_{i,j}, \dots, p_{i,d}\}$ ve $p_{i,j}$ değeri veri setindeki $i.$ noktanın $j.$ boyutundaki gerçek değeri ifade eder. $i = 1,2,\dots,n$ ve $j = 1,2,\dots,d$'dir.

Kümeleme algoritmaları bu veri setindeki bütün noktaları k farklı sınıfa ayırarak, $C = \{C_1, C_2, \dots, C_k\}$ gibi bir sınıflandırma yapar. Kümeleme işlemi sonucu oluşan her bir grubu diğer gruplardan mümkün olduğunca farklı, kendi içersindeki elemanların da mümkün olduğunca benzer olması gerekir. Sınıflandırma işlemi sonucu oluşan kümelerin şu özellikleri taşıması gerekir [45]:

- Her bir kümeye en az bir eleman atanmalıdır. $C_i \neq \emptyset \; ve \; \forall \; i \in \{1,2,..k\}$
- İki farklı kümenin ortak elemanı olmamalıdır. $C_i \cap C_j = \emptyset \;\; ve \; \forall \; i \neq j \;\; ve \; i,j \in \{1,2,..k\}$
- Her bir veri bir kümeye atanmalıdır. $\bigcup_{i=1}^{k} C_k = P$

Kümeleme işleminin sırasında veri setindeki iki eleman arasındaki benzerlik miktarı genel olarak öklit mesafesi kullanılarak belirlenir. Örneğin $\overrightarrow{P_i}$ ve $\overrightarrow{P_j}$ gibi d boyutlu iki örnek eleman arasındaki benzerlik miktarı şu şekilde hesaplanır [33]:

$$d(\overrightarrow{P_i}, \overrightarrow{P_j}) = \sqrt{\sum_{dim=1}^{d} \left(p_{i,dim} - p_{j,dim}\right)^2} = \left\|\overrightarrow{P_i}, \overrightarrow{P_j}\right\| \tag{5.1}$$

5.2.2. PSO' nun Kümeleme İşlemine Uygulanması

n veriden oluşan P veri setinin kümelenmesi sırasında karşımıza çok fazla kümeleme seçeneği çıkmaktadır. Bu seçeneklerden en iyisini seçmek de bir optimizasyon konusudur.

Daha öncede belirtildiği gibi PSO'daki her bir parçacık problem için muhtemel bir çözüm önerisidir. Kümeleme problemlerinde de sürüdeki parçacıklar kümeler için en uygun ağırlık merkezlerini bulmaya çalışırlar. Bulunan bu merkezlere göre veri setindeki bütün elemanlar uygun kümelere dağıtılarak en uygun kümeleme seçeneği bulunmuş olur.

Sürünün başlangıç populasyonunu X; $X = \{ X_1, \dots, X_i, \dots, X_{pop}\}$ ve $i = 1,2,\dots, pop$; pop, sürüdeki parçacık sayısı şeklinde ifade edilir.

Yöneticili kümeleme işlemleri sırasında da her bir parçacık oluşturulan kümelerin merkezini temsil eden birer vektördür. Her bir parçacık, $X_i = \{ \overrightarrow{o_1}, \overrightarrow{o_2}, \dots, \overrightarrow{o_j}, \dots, \overrightarrow{o_k}\}$ ve $j = 1,2,\dots., k$; k, oluşturulan küme sayısı, o_j, j. kümenin ağırlık merkezi şeklinde ifade edilir.

Her bir kümenin ağırlık merkezi $\overrightarrow{o_j} = \{o_{j,1}, o_{j,2}, \dots, o_{j,d}\}$ d boyutlu bir vektördür. Ağırlık merkezlerinin boyutu veri setindeki kümelecek her biri elemanın boyutuna eşittir. Her bir boyutta gerçek değerler vardır.

Yöneticiz kümeleme algoritmalarında veri setinin kaç kümeye ayrılacağı da diğer bir optimizasyon konusudur. Yapılan çalışmada bu iki birbirine bağlı optimizasyon problemini için PSO'da farklı parçacıklar kullanılmıyacaktır. Bunun yerine her iki probleme de çözüm önerisi aynı parçacık üzerinden getirilecektir. Yani bir parçacık hem verilerin kaç ayrı kümeye ayrılacağı konusunda hem de kümelemenin en doğru şekilde yapılması konusunda çözüm üretecektir. Buna göre sürüdeki $i.$ parçacığın yapısı; $X_i = \left\{ \overrightarrow{o_1}, \overrightarrow{o_2}, \dots, \overrightarrow{o_j}, \dots, \overrightarrow{o_k}, z_1, z_2, \dots, z_j, \dots, z_k \right\}$ ve $j = 1,2,\dots, k$ şeklinde

olacaktır. Gösterimdeki k sayısı kullanıcı tarafından girilen olabilecek maksimum küme sayısıdır. z_j ise [0,1] aralığında değerler alabilen ve j. kümenin aktif olup olmadığını gösteren bir değerdir.

EĞER $z_j \geq 0{,}5$ ise

j. kümenin ağırlık merkezi AKTİF

DEĞİLSE (Yani $z_j < 0{,}5$ ise)

j. kümenin ağırlık merkezi PASİFTİR.

Yani z_j belirtilen eşik değerine eşit ya da eşik değerinden daha büyük ise ise j. küme aktif, eşik değerinden küçük ise pasif durumdadır. Örneğin z= { 0,45 ; 0,63; 0,37; 0,91; 0,23; 0,11; 0,88; 0,20; 0,54; 0,17 } değerine sahip bir parçacık için, eşik değerinin 0,5'den büyük olduğu $\vec{m}$ ağırlık merkezi vektörlerinin 2. , 4. , 7. , 9. elemanları aktiftir. Bu durumda i. parçacığın dikkate alınması gereken hali $X_i = \{ \overrightarrow{o_2}, \overrightarrow{o_4}, \overrightarrow{o_7}, \overrightarrow{o_9}, z_2, z_4, z_7, z_9\}$ şeklinde olacaktır.

Sonuç olarak d boyutlu, en fazla k kümeden oluşan i. parçacığın temsili gösterimi, $X_i = \{ o^i_{1,dim}, o^i_{2,dim}, \dots, o^i_{k,dim}, z^i_1, z^i_2, \dots, z^i_k \}$; $i: 1,2,\dots,n$; k: maksimum küme sayısı; $dim: 1,2,\dots,d$ şeklinde olacaktır. Bu formülasyona göre bir parçacık $(k * d + k)$ tane gerçek değer içeren haneden oluşacaktır. Aynı şekilde parçacığın hız vektörü de konum vektörü ile aynı yapıda olacaktır [34].

5.2.3. Kümeleme Doğruluk İndeksi

Kümeleme doğruluk indeksi (KDİ), kümeleme algortimaları tarafından üretmiş kümeleme seçeneklerinin sayısal tabana dayandırılarak uygunluğunun değerlendirildiği fonksiyondur. Birçok PSO probleminde uygunluk fonksiyonu olarak adlandırılan bu fonksiyon, kümeleme algoritmalarında kümeleme doğruluk indeksi olarak adlandırılmıştır.

Kümeleme algoritmalarında çözüm üretilirken iki temel esas alınır:

- Oluşturulan kümelerin elemanları kendi içlerinde azami derecede birbirine yakın olmalı,
- Oluşturulan farklı kümeler arası da azami ayrışma olmalıdır.

KDİ genel olarak kesin kümeleme algoritmalarında kullanılan bir kavramdır. Literatürde birçok KDİ'nin kullanıldığı ve kıyaslandığı çalışmalar yapılmıştır [47, 48]. Tez çalışmasında KDİ olarak üç farklı uygunluk fonksiyonu kullanılacaktır.

5.2.3.1. Kümeleme Doğruluk İndeksi 1 (F_1)

Bu kümeleme doğruluk indeksi [49]'dan esinelenerek elde edilmiştir. Bir parçacığın uygunluk değeri aşağıdaki fonksiyona göre hesaplanır:

$$F_1 = \frac{1}{n}\sum_{i=1}^{n} d(o_j, {p_i}^{C_j}) \tag{5.2}$$

n: Veri setindeki toplam eleman sayısı,
${p_i}^{C_j}$: j. kümeye ait bir eleman,

Öncelikle veri setindeki bütün elemanlar en yakın olduğu kümelere dağıtılır. Veri setindeki bütün elemanlar ile ait oldukları kümelerin ağırlık merkezi arasındaki mesafeler toplanır. Sonuç veri setindeki toplam elaman sayısına bölünerek bulunur. ${F_1}'$ in küçük değer alması uygunluğun artması demektir. Yani F_1 bir minimizasyon fonksiyonudur.

5.2.3.2. Kümeleme Doğruluk İndeksi 2 (F_2)

Bu KDİ, [50]'den esinelenerek elde edilmiştir. Bir parçacığın uygunluk değeri aşağıdaki fonksiyona göre hesaplanır:

$$F_2 = \frac{\sum_{j=1}^{k}\left(\sum_{\forall p_i \in C_j} d(p_i, o_j)/\left|C_j\right|\right)}{k} \tag{5.3}$$

k: toplam küme sayısı,

$|C_j|$: j. kümedeki toplam eleman sayısı,

Öncelikle veri setindeki bütün elemanlar en yakın olduğu kümelere dağıtılır. Oluşturulan her bir küme için, kümeye ait olan elemanların kümenin ağırlık merkezine olan uzaklıkları ayrı ayrı hesaplanır ve toplanır. Elde edilen değer kümenin eleman sayısına bölünür. Bu işlem her bir küme için yapılır ve elde edilen değerler toplanır. Elde edilen toplam, küme sayısına bölünerek uygunluk değeri hesaplanmış olur. F_2' nin küçük değer alması uygunluğun artması demektir. Yani F_2 bir minimizasyon fonksiyonudur.

5.2.3.3. Kümeleme Doğruluk İndeksi 3 (F_3)

Bu kümeleme doğruluk indeksi [34] ve [36]'den esinelenerek elde edilmiştir. Bir parçacığın uygunluk değeri aşağıdaki fonksiyona göre hesaplanır:

$$F_3 = \frac{\frac{1}{k}\sum_{j=1}^{k}\left\{\frac{1}{|c_j|}\sum_{\forall p_i \in C_j} max_{\forall p_k \in C_j}\{d(p_i, p_k)\}\right\}}{\frac{1}{k}\sum_{j=1}^{k}\{min_{t\in k \vee t\neq j}\{d(o_j, o_t)\}\}} \tag{5.4}$$

sadeştirirsek,

$$F_3 = \frac{\sum_{j=1}^{k}\left\{\frac{1}{|c_j|}\sum_{\forall p_i \in C_j} max_{\forall p_k \in C_j}\{d(p_i, p_k)\}\right\}}{\sum_{j=1}^{k}\left\{min_{t\in k \vee t\neq j}\{d(o_j, o_t)\}\right\}} \tag{5.5}$$

$$o_j = \frac{1}{|C_j|}\sum_{p_i \in C_j} p_i \;\; j = 1,2, \dots, k \tag{5.6}$$

k: toplam küme sayısı

C_j: j. küme

$|C_j|$: j. kümedeki toplam eleman sayısı,

Öncelikle veri setindeki bütün elemanlar en yakın olduğu kümelere dağıtılır. Uygunluk fonksiyonun pay kısmı hesaplanmadan küme içindeki her bir elemanın yine aynı küme içinde en uzağındaki diğer bir eleman belirlenir. Bunlar arasındaki mesafe hesaplanır. Bu hesaplama kümedeki bütün elemanlar için yapılır. Hesaplanan bu mesafelerin hepsi toplanır ve kümedeki eleman sayısına bölünür. Bu işlem her bir küme için ayrı ayrı yapılır ve sonuçlar toplanarak, payın değeri elde edilmiş olur. Fonksiyonun paydası hesaplanmadan önce ise her bir kümenin elemanlarının her bir boyutunun aritmatik ortalaması bulunarak, kümenin ağırlık merkezi belirlenir. Her bir kümenin ağırlık merkezine en yakın uzaklıktaki diğer bir kümenin ağırlık merkezi bulunup aralarındaki mesafeler toplanarak payda elde edilmiş olur. Elde edilen pay ve paydanın oranı da uygunluk değeridir. F_3'ün küçük değer alması uygunluğun artması demektir. Yani F_3 bir minimizasyon fonksiyonudur.

5.3. PSO İLE KÜMELEME ALGORİTMASININ GERÇEKLEŞTİRİLMESİ

PSO ile kümeleme işlemini gerçekleştirmek için izlenmesi gereken adımlar aşağıdaki sözde kodda ifade edildiği gibidir [36]:

Girdiler: Veri seti $P = \{\overrightarrow{P_1}, \overrightarrow{P_2}, \overrightarrow{P_3}, \dots, \overrightarrow{P_n}\}$
Çıktılar: Kümeler $C = \{C_1, C_2, \dots, C_k\}$

```
BEGIN
Sürünün pop Tane Elemanına Başlangıç Değeri Ata
REPEAT
    FOR i=1 TO n
        FOR j=1 TO k
            d(p_i,o_j) Mesafesini Hesapla (Formül 5.1)
        END FOR
        Eğer o_j, p_i'ye En Yakın Ağırlık Merkezi ise p_i'yi C_j Kümesine Ata
    END FOR
    FOR s=1 TO pop
        Parçacığın KDİ'sini Hesapla
        p_best Değerini Hesapla
```

END FOR

g_{best} Değerini Hesapla

FOR $s=1$ TO pop

Parçacığın Hızını Güncelle

Parçacığın Konumunu Güncelle

END FOR

UNTIL Durdurma Kriterine Ulaşıncaya Kadar

END

Burada;

pop: Sürüdeki parçacık sayısı,

n: Veri setinin eleman sayısı,

k: Küme sayısı,

p_i: Veri setinin i. elemanı,

m_j: j. kümenin ağırlık merkezi'dir.

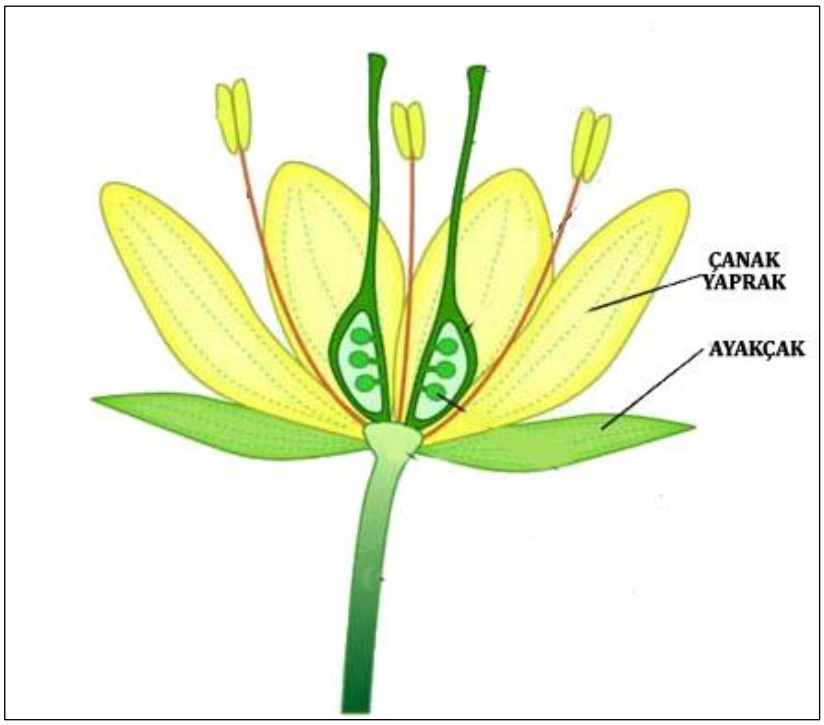
5.4. PROBLEM UYGULAMASI

Tez kapsamında PSO kümeleme uygulamasında zambak çiçeği veri seti kümelenecektir. Zambak çiçeği veri seti [51], çiçeğinin santimetre cinsinden çanak yaprak uzunluğu, çanak yaprak genişliği, ayakçak yaprak uzunluğu ve ayakçak yaprak genişliğini gibi dört farklı değişken içeren 150 elemandan oluşan dört boyutlu bir veri setidir. Şekil 5.2'de zambak çiçeğinin resmi görülmektedir.

Üç farklı kümeden oluşan zambak çiçeği veri seti eleman sayıları ve elemanların dağılımı Çizelge 5.1'deki gibidir.

Çizelge 5.1. Zambak çiçeği veri setinin dağılımı.

Küme Adı	Kümenin Eleman Sayısı
Iris Setosa	50
Iris Virginica	50
Iris Versicolor	50



Şekil 5.2. Zambak çiçeği ve yapraklarının yapısı.

Yukarıda bahsedilen üç kümeleme doğruluk indeksinin zambak çiçeği verileri üzerinde kümeleme performansları değerlendirilecektir. Bu kümeleme işlemi için geliştirilen kod Microsoft Visual Studio 2008 C# ortamında geliştirilmiştir. Geliştirilen bu kod Window 7 Professional 64 bit işletim sistemin sahip Intel Core2 Duo 2,63GHz işlemcili ve 4 GB RAM'li bir dizüstü bilgisayar üzerinde çalıştırılmıştır. Uygulamada kullanılacak parametrik değerler hızlandırma katsayıları $c_1 = 1{,}49$, $c_2 = 1{,}49$, atalet ağırlığı $w = 0{,}72$, maksimum iterasyon sayısı $t_{max} = 1000$, parçacık sayısı $p = 20$, hız güncelleme limiti V_{max} değer aralığının %1'idir. Elde edilen sonuçlar programın her bir yöntem için 20'şer defa birbirinden bağımsız çalıştırılması ile elde edilmiştir.

Öncelikle yöneticisiz kümeleme algoritması üç farklı kümeleme indeksine göre ayrı ayrı uyarlanmıştır. Bu uygulamada amaç veri setinin ayrılacağı küme sayısını bulmaktır. Yapılan uygulamalar sonucunda elde edilen değerler Çizelge 5.2'de gösterilmiştir.

Çizelge 5.2. Yöneticisiz kümeleme indekslerinin sonuçları.

Veri	KDİ	Ortalama KDİ Değeri	Ortalama Küme Sayısı	Sonucun Bulunduğu Ortalama İterasyon
İris	F_1	0,047	4,9	198
	F_2	0,479	6,0	661
	F_3	0,602	2,7	93

Çizelge 5.2'deki sonuçlarda da görüldüğü üzere zambak çiçeği verisinin gerçek küme sayısı olan üçe en yakın değerin F_3 kümeleme indeksine ait olduğu görülmektedir. Diğer iki kümeleme indeksinin yöneticisiz kümeleme algoritmalarında kullanımının doğru sonuçlar vermeyeceği görülmektedir. Çizelge 5.3'de de görüldüğü gibi F_3 kümeleme indeksi kullanılan kümeleme algoritmasının 20 defa çalıştırılmasında 14 kez küme sayısı 3, 6 kez 2 bulunmuştur. Ortalama 2,7 olarak bulunan küme sayısı yuvarlama ile gerçek zambak çiçeği verisinin ayrıldığı küme sayısı olan üçe eşit olmaktadır.

Çizelge 5.3. F_3 Kümeleme indeksinin uygulama sonuçları.

Bulunan F_3 Değeri	Sayısı	Küme Sayısı
0,595	14	3
0,626	6	2

Zambak çiçeği veri setine yöneticili kümeleme algoritmasının uygulanmasında amaç, verilerin gerçeğine en yakın şekilde kümelenmesidir. Bu üç kümeleme indeksinin kullanıldığı yöneticili kümeleme algoritmasına zambak çiçeği veri setinin küme sayısı olan üç değeri ön bilgi olarak verilmektedir. Buna göre elde edilen kümeleme indeks değerleri Çizelge 5.4'de gösterilmiştir.

Çizelge 5.4. Kümeleme indeksi uygunluk değerleri.

Veri	KDİ	En iyi	En kötü	Ort.	$\frac{\text{Ort.} - \text{En İyi}}{\text{En İyi}}$	$\frac{\text{Ort.} - \text{En Kötü}}{\text{En Kötü}}$
İris	F_1	0,0314	0,0323	0,0316	0,006	-0,021
	F_2	0,6290	0,6490	0,6360	0,011	-0,020
	F_3	0,595	0,595	0,595	0,000	0,000

Kümeleme doğruluk indekslerinin kümeleme başarımı ise hatalı kümelenen veri sayısının % olarak Kümeleme Hata (CE) oranları Çizelge 5.5'de gösterilmiştir. Kümeleme hata oranları;

$$CE = (Hatalı\ kümelenen\ eleman\ sayısı/toplam\ \ eleman\ sayısı) \quad (5.7)$$

formülüne göre hesaplanmıştır.

Çizelge 5.5. Kümeleme indekslerinin kümeleme hata oranları *(CE)*.

Veri	Kümeleme İndeksi	En İyi	En Kötü	Ort.	$\frac{\text{Ort.} - \text{En İyi}}{\text{En İyi}}$	$\frac{\text{Ort.} - \text{En Kötü}}{\text{En Kötü}}$
İris	F_1	% 4,0	% 10,6	% 5,8	0,45	-0,45
	F_2	% 10,0	% 14,6	% 11,1	0,11	-0,23
	F_3	% 32,0	% 32,0	% 32,0	0,00	0,00

F_3 kümeleme indeksi kararlı bir yapı göstermesine rağmen istenenden uzak bir kümeleme dağılımı göstermekte ve bu üç yöntem arasında en başarımı düşük yöneticili kümeleme algoritması olarak görülmektedir. F_1 kümeleme doğruluk indeksi ise yöneticili kümeleme algoritmaları arasında kümeleme başarımı en yüksek olan ve gerçeğe en yakın dağılımı sağlayan bir kümeleme indeksidir.

Herhangi bir veri setinin kümelenmesi aşamasında öncelikle küme sayısının tespiti önemlidir. Yapılan bu çalışma sonucunda küme sayısının tespitinde F_3 kümeleme doğruluk indeksini kullanılan yöneticisiz kümeleme algoritmasının kullanılabileceği görülmüştür. Küme sayısı tespit edilen veri setinin kümeleme dağılımı için ise en uygun yöntem F_1 kümeleme doğruluk indeksini kullanan yöneticili kümeleme algoritmasının olduğu görülmüştür.

5.5. BULANIK KÜMELEME ALGORİTMASI

Bulanık C-Means (FCM), bulanık kümeleme algoritmaları arasında en popüler olan algoritmadır. İlk olarak 1981 yılında Bezdek tarafında geliştirilerek şekil tanıma

çalışmalarında kullanılmıştır [52]. FCM, K-Means kümeleme algoritmasından türetilerek oluşturulmuş bir kümeleme algoritmasıdır. FCM algoritması bulanık teorisindeki [53] bulanık kümeleri ve bu kümelere üyelik kavramı kullanılarak geliştirilmiştir. Kesin (crisp) kümeleme yöntemlerinden farklı olarak bir veri sadece bir kümeye değil farklı üyelik değerleri ile birden fazla kümeye ait olabilir.

n elemandan oluşan bir veri seti $A = \{ a_1 , a_2, \dots , a_i , \dots , a_n\}$ şeklinde tanımlansın. a_i, $a_i = \{a_{i,1}, a_{i,2}, \dots, a_{i,d}\}$ şeklinde d boyutlu bir vektördür. $n, d, k \in N$ ve $1 < k < n$ olmak üzere, A veri setinin elemanları k kümeye ayrılırken μ_{ij} üyelik değeri kullanılır. μ_{ij}, veri setindeki $i.$ verinin $j.$ kümeye ait olan üyelik değerini ifade eder. μ_{ij} ile ilgili kısıtlar aşağıdaki gibidir [54, 55]:

- $\mu_{ij} \in$ [0,1]; $\forall\, i = 1, 2, \dots, n \; ve \; \forall\, j = 1,2, \dots, k.$ Her bir üyelik değeri [0,1] aralığında olmalıdır.
- $\sum_{j=1}^{k} \mu_{ij} = 1; \; \forall\, i = 1, 2, \dots, n.$ Veri setindeki bir verinin k kümeye üyelikleri toplamı 1'e eşit olmalıdır.
- $0 < \sum_{i=1}^{n} \mu_{ij} <$ n; $\forall\, j = 1,2, \dots, k.$ Bir kümeye ait üyelik değerleri toplamı $(0, n)$ aralığında olmalıdır.

M, μ_{ij}'lerden oluşan $(n \times k)$ boyutlu üyelik matrisi, $O = \{o_1, o_2, \dots, o_k\}$ olmak üzere, O oluşturulan her bir kümenin ağırlık merkezini gösteren $(k \times d)$ boyutlu küme merkezleri matrisi iken FCM'nin uygunluk fonksiyonu**;**

$$J_{FCM}(M, O; X) = \sum_{j=1}^{k} \sum_{i=1}^{m} {\mu_{ij}}^{m} {d_{ij}}^{2} \tag{5.8}$$

formülüne göre hesaplanır. d_{ij} $i.$ veri ile $j.$ küme merkezi arasındaki öklit uzaklığıdır. m değeri üyelik değerini bulanıklaştırmak üzere kullanılan ve $m > 1$ şeklinde değerler alan bir bulanıklaştırma parametresidir. FCM algoritmasında amaç J_{FCM} değerini minimuma indirmektir.

Veri setindeki $i.$ elemanın *j.* kümeye üyelik değeri;

$$\mu_{ij} = \frac{1}{\sum_{c=1}^{k} (\frac{d_{ij}}{d_{ic}})^{\frac{2}{(m-1)}}} \tag{5.9}$$

formülüne göre hesaplanır. Bir kümenin ağırlık merkezi ise;

$$o_j = \frac{\sum_{i=1}^{n} (\mu_{ij}{}^{m}\, a_i)}{\sum_{i=1}^{n} \mu_{ij}{}^{m}} \tag{5.10}$$

formülüne göre hesaplanır. Bu formüllere göre FCM'nin sözde kodu [54]:

BEGIN
m, c and ε tanımla
O martisine başlangıç değerleri ata
T=0;
DO
    Formül 5.9'a göre M matrisini hesapla
    Formül 5.10'a göre O matrisini hesapla
    T=T+1;
WHILE ($\sum_{j=1}^{c} \left\| o_j{}^{T} - o_j{}^{T-1} \right\| > \varepsilon$)
END

5.6. PSO DESTEKLİ FCM (FPSO)

Teoride n veriden oluşan P veri setinin $C = \{C_1, C_2, \dots, C_k\}$ gibi k farklı kümeye ayrılması sırasında karşımıza çok fazla kümeleme seçeneği çıkar. Bu kümeleme seçeneklerden en iyisini seçmek de bir optimizasyon konusudur. FCM'nin en büyük olumsuzluğu başlangıç değerlerine bağlı olmasıdır. Başlangıç değerlerinin iyi değerler olmaması FCM'nin yerel optimumlara takılmasına sebep olabilir. Bu sebeple FCM, PSO gibi populasyon tabanlı bir arama algoritması ile desteklenmesi başarımını artıracaktır [55]. PSO'daki her bir parçacık problem için bir çözüm önerisidir. Kümeleme problemlerinde de sürüdeki her bir parçacık ayrı bir kümeleme seçeneği sunar. Her bir kümeleme seçeneği Formül 5.8'deki uygunluk fonksiyonuna

göre değerlendirilir ve PSO'daki iterasayonlar boyunca en uygun kümeleme gerçekleştirilmeye çalışılır.

Kesin kümeleme algoritmalarından farklı olarak FPSO'da FCM'de olduğu gibi bir veri birden fazla kümeye ait olabilir. Bütün verilerin her bir kümeye μ_{ij} gibi ayrı bir üyelik değeri ile ait olabilirler. Her bir küme o_j gibi d boyutlu bir küme merkezine sahiptir. FCM ile PSO'nun entegrasyonu sırasında Küme Merkezine Dayalı FPSO ve Üyeliğe Dayalı FPSO olmak üzere iki farklı yöntem izlenebilir [56].

5.6.1 Küme Merkezine Dayalı FPSO (C-FPSO):

C-FPSO'da bir parçacık, kümelerin ağırlık merkezlerinin değerlerini tutar ve arama işlemi boyunca en iyi ağırlık merkezlerini bulmaya çalışır. Her parçacık $(k \times d)$'lik kümele merkezi O matrisini ifade eder [55].

$$O = \begin{bmatrix} o_{11} & \cdots & o_{1d} \\ \vdots & \ddots & \vdots \\ o_{k1} & \cdots & o_{kd} \end{bmatrix}_j \tag{5.11}$$

C-FPSO'nun her bir iterasyonunda Formül 5.9'a göre verilerin her bir kümeye olan üyelik değeri μ_{ij} hesaplanır. Önerilen sözde kod:

BEGIN
p parçacık için O matrisini oluştur
FOR i=1 TO t_{max}
    FOR i=1 TO p
        Formül 5.9'a göre M matrisini hesapla
        J_{FCM} değerini hesapla,
        Hız ve pozisyon değerlerini güncelle
    END FOR
END FOR
END

5.6.2. Üyeliğe Dayalı FPSO (M-FPSO):

M-FPSO'da bir parçacık bütün verilerin her bir kümeye ayrı ayrı üyelik değerlerini tutar ve arama işlemi boyunca en iyi üyelik değerleri bulunmaya çalışır. Her parçacık $(n \times k)$'lik M matrisini temsil eder [57].

$$M = \begin{bmatrix} \mu_{11} & \cdots & \mu_{1k} \\ \vdots & \ddots & \vdots \\ \mu_{n1} & \cdots & \mu o_{nk} \end{bmatrix} \tag{5.12}$$

M-FPSO'nun her bir iterasyonunda Formül 5.10'a göre her bir kümenin ağırlık merkezi o_j hesaplanır. Önerilen sözde kod:

BEGIN
p parçacık için M matrisini oluştur
FOR i=1 TO t_{max}
    FOR i=1 TO p
        Formül 5.10'e göre O matrisini hesapla
        J_{FCM} değerini hesapla,
        Hız ve pozisyon değerlerini güncelle
    END FOR
END FOR
END

5.7. PROBLEM UYGULAMASI

Tez kapsamında yapılan çalışmada, M-FPSO ve C-FPSO yöntemlerinin kümeleme performansları değerlendirecektir. Bu kümeleme işlemi için geliştirilen kod Microsoft Visual Studio 2008 C# ortamında geliştirilmiştir. Geliştirilen bu kod Window 7 Professional 64 bit işletim sistemine sahip Intel Core2 Duo 2.63 GHz işlemcili ve 4GB RAM'li bir dizüstü bilgisayar üzerinde çalıştırılmıştır. M-FPSO ve C-FPSO uygulamalarında kullanılacak parametrik değerler; hızlandırma katsayıları $c_1 = 2.0, c_2 = 2.0$, atalet ağırlığı $w = 0.75$, maksimum iterasyon sayısı $t_{max} = 1000$, hız güncelleme limiti V_{max} değer aralığının %1'i, populasyon sayısı p= 50;

bulanıklaştırma parametresi $m = 2.0$ şeklindedir.

Değerlendirmede zambak çiçeği veri seti kullanılacaktır. Her iki yöntem içinde program 20'şer defa birbirinden bağımsız olarak çalıştırılmıştır. Elde edilen uygunluk değeri ile ilgili sonuçlar Çizelge 5.6'da gösterilmiştir.

Çizelge 5.6. Uygunluk fonksiyon değerleri.

Veri	Yöntem	En İyi	En Kötü	Ort.	$\frac{\text{Ort.} - \text{En İyi}}{\text{En İyi}}$	$\frac{\text{Ort.} - \text{En Kötü}}{\text{En Kötü}}$
Iris	C-FPSO	60,505	60,505	60,505	0,0	0,0
	M-FPSO	64,949	73,690	67,205	0,034	-0,088

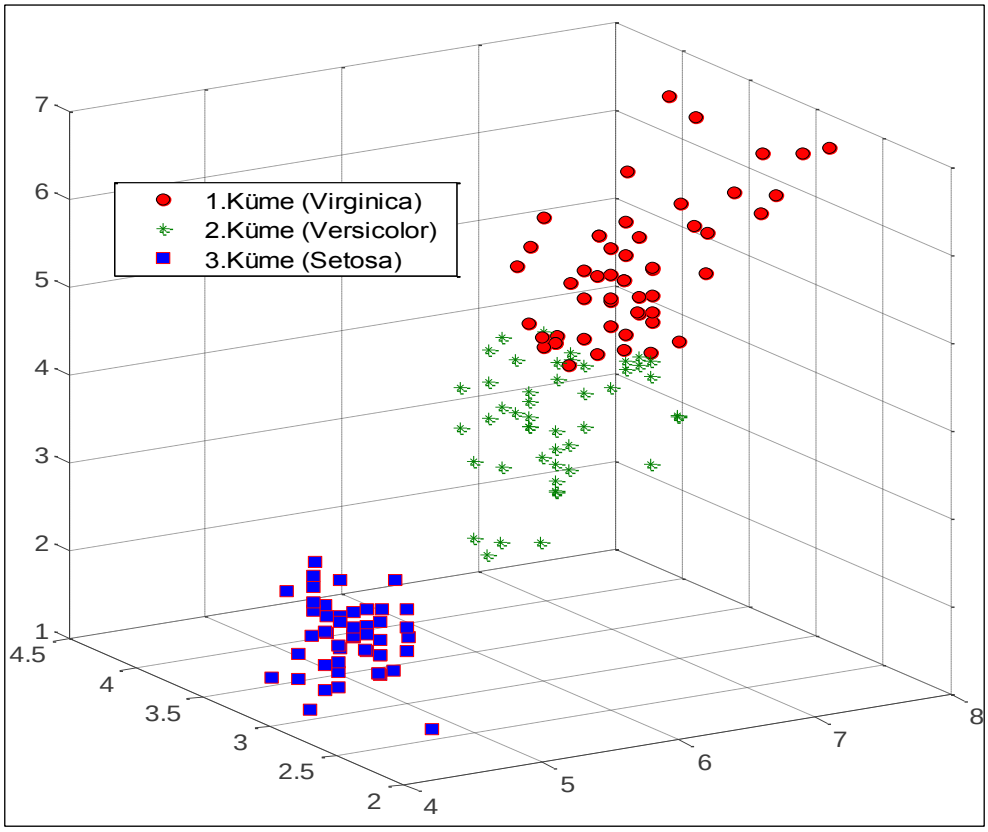
Bu uygunluk değerlerinin yanı sıra iki yöntemin kümeleme başarımları ve yüzde olarak hata oranları Çizelge 5.7'de gösterilmiştir. Bu tablodaki değerler her iki yöntem için 20'şer defa çalıştırılan kodların en iyi kümeleme sonuçları temel alınarak yapılmıştır.

Çizelge 5.7. Kümeleme sonuçları.

Yöntem	Küme Adı	1. Küme	2. Küme	3. Küme	Kümeleme Hata Oranları(CE)
C-FPSO	Setosa	50	0	0	% 10,66
	Virginica	0	37	13	
	Versicolor	0	3	47	
M-FPSO	Setosa	50	0	0	%6,00
	Virginica	0	3	47	
	Versicolor	0	44	6	

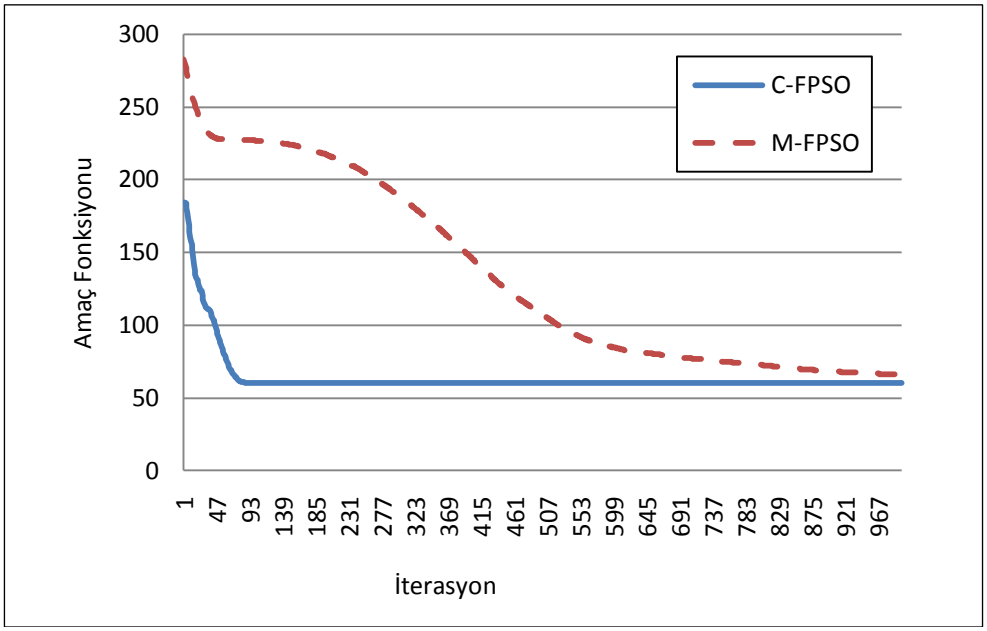
Çizelge 5.6'daki sonuçlara göre C-FPSO'nun M-FPSO' ya göre daha kararlı yapıda olduğu ve standart sapmasının 0.0 olduğu görülmektedir. C-FPSO yöntemi 20 defa çalıştırıldığında da aynı sonucu vermiştir. Aynı zamanda C-FPSO'nun uygunluk fonksiyonun değeri M-FPSO'ya göre daha düşüktür. Fakat Çizelge 5.7'deki sonuçlara göre M-FPSO'nun kümeleme hata oranının daha düşük olduğu görülmektedir. Bu da M-FPSO'nun gerçek kümeleme dağılımına daha yakın sonuçlar ürettiğini

göstermektedir. Şekil 5.2'de M-FPSO'nun oluşturduğu kümeleme dağılımı görülmektedir. Zambak çiçeği veri setindeki elemanlar dört boyutlu olmasına rağmen üç boyutlu ele alınarak Şekil 5.2'deki grafik oluşturulmuştur.



Şekil 5.3. Zambak çiçeği veri setinin üç boyutlu eksende dağılımı.

Yapılan çalışma sonucunda $t_{max} = 1000$ için M-FPSO'nun C-FPSO'ya göre kümeleme hata oranının daha düşük olduğu görülmüştür. Fakat C-FPSO 20 defa çalıştırılarak, uygunluk fonksiyonu değeri olarak M-FPSO'dan daha küçük değerler bulmuş ve standart sapması da 0.0 olmuştur. Ayrıca Şekil 5.3'de görüldüğü gibi C-FPSO yaklaşık olarak 100. iterasyonda optimum değeri bulurken, M-FPSO 1000 iterasyonda bu değere yaklaşabilmiştir.



Şekil 5.4. Uygunluk fonksiyon değerleri.

Bu durum C-FPSO ve M-FPSO'daki parçacıkların yapısından kaynaklanmaktadır. M-FPSO'da bir parçacık bütün verilerin her bir kümeye ayrı ayrı üyeliğini gösteren $(n \times k)$'lık M matrisini ifade eder. Zambak çiçeği veri seti için bu M matrisi (150×4) boyutludur. C-FPSO'da ise bir parçacık her bir kümenin d boyutlu merkezlerini ifade eden $(k \times d)$ boyutlu O matrisidir. Zambak çiçeği veri seti için bu O matrisi (3×4) boyutludur. Görüldüğü gibi M-FPSO'da her bir parçacık (150×4)'lük bir matrisin optimum değerlerini bulmaya çalışırken C-FPSO'da (3×4)'lük bir matrisin optimum değerini bulmaya çalışır. Bu sebeple C-FPSO, M-FPSO'ya göre çok daha hızlı bir yakınsama yapar ve daha kararlı bir yapı gösterir. Zambak çiçeği verilerinin kümelenmesi açısından C-FPFO'nun M-FPSO'ya göre daha hızlı sonuç bulduğunu söylenebilir.

BÖLÜM 6

SONUÇ VE ÖNERİLER

Uygulanan problemlerde PSO'nun az kontrol parametresine sahip olması ve çözüme hızlı ulaşması bakımından tercih edilebilir bir optimizasyon tekniği olduğu görülmüştür. Popülasyon temelli olduğu için çözüm uzayında yerel minimumlara takılmayan veya yerel minimumlardan kolay kurtulan PSO'nun araştırmacı ve kaşif bir yapısı da vardır. Sürüdeki parçacıklar iterasyonlar boyunca sahip oldukları değerleri iyileştirirler ve sonuçta genel olarak en iyi ve en iyiye yakın değerlere sahip bir popülasyon elde edilir.

Tezde yapılan ilk uygulamada;

- Bir çok kısıta sahip genel bir optimizasyon problemi üzerinde çalışılmıştır. Problemin çözümünde PSO'nun çözüm uzayını sınırlandıran farklı yöntemleri incelenmiş ve başarımlarını değerlendirilmiştir.
- Bu problem için kullanılan yöntemler arasından *Sönümleme* yönteminin en hızlı ve en güvenilir sonuç veren yöntem olduğu ortaya çıkmıştır.
- Bunun yanı sıra *Emme* ve *Görünmez* yöntemleri da sonucu bulma hızı konusunda Sönümleme yöntemi kadar başarılı olmasalar da optimum sonuç bulma konusunda gayet başarılı oldukları gözlemlenmiştir.
- Genel olarak bakıldığında sınırlayıcı yöntemlerin sınırlayıcı olmayan yöntemlere bir üstünlüğünden söz edilebilmektedir.
- Tez kapsamında kullanılan altı yöntemin genel olarak optimum sonuca ulaşma süreleri bakımından PSO'yu etkilediği görülmektedir.

Tezde çalışılan ikinci uygulamada zambak çiçeği verilerinin kümelenmesi konusunda yöneticisiz ve yöneticili olmak üzere iki farklı kümeleme algoritması PSO kullanılarak denenmiştir. Bu uygulamalarda PSO'da uygunluk fonksiyonu

olarak F_1, F_2, F_3 kümeleme doğruluk indeksleri ayrı ayrı kullanılmıştır.
Bu ikinci uygulamada;

- Bir veri setini doğru sayıda kümeye bölmek için kullanılan yöneticisiz kümeleme algoritmalarında F_1, F_2 kümeleme doğruluk indekslerinin kullanıldığı PSO'nun başarısız olduğu görülmüştür.
- F_3 doğruluk indekslerinin kullanıldığı PSO'nun ise doğru küme sayısı olan 3'ü birçok çalıştırmada bulduğu görülmüştür.
- Doğru küme sayısının bir ön veri olarak verildiği yöneticili kümeleme uygulamalarında ise kümeleme doğruluk indeksi F_1 olan yönteminin ortalama % 5,8 oranında hatalı sonuç verdiği, F_2 olan yönteminin ortalama % 11,1 oranında hatalı sonuç verdiği ve F_3 olan yönteminin ise ortalama %32,0 oranında hatalı sonuç verdiği görülmüştür. Bu durumda F_3 kümeleme indeksinin hata oranın kabul edilemeyecek derecede olduğu görülmüştür.
- En başarılı yöneticili kümeleme algoritması ise kümeleme doğruluk indeksi F_1 olan PSO kümeleme yöntemi olduğu görülmüştür.
- Bu sonuçlardan hareketle, herhangi bir veri setinin küme sayısını bulmak için yöneticisiz F_3 kümeleme indeksli PSO algoritması, küme sayısı tespit edildikten sonra en doğru kümeleme dağılımını yapmak için ise yöneticili F_1 kümeleme indeksli PSO algoritması kullanılabileceği görülmüştür.

Ayrıca bu uygulamanın devamında FCM kümeleme algoritması ile PSO hibritleştirilerek farklı bir kümeleme algoritması (FPSO) oluşturulmuştur. Bu kümeleme algoritması da Üyeliğe Dayalı FPSO (M-FPSO) ve Küme Merkezine Dayalı FPSO (C-FPSO) olmak üzere iki farklı yöntem kullanılarak uygulanmıştır.

Küme Merkezine Dayalı FPSO'nun parçacık yapısından dolayı hızlı sonuç bulduğu fakat Üyeliğe Dayalı FPSO'nun ise daha başarılı bir kümeleme performansı gösterdiği görülmüştür.

İleriye dönük çalışmalar ise şu şekilde öneriler yapılabilir;

- PSO'nun hızlı çözüm üretmesi ve yerel optimumlara takılmama gibi özellikleri ile optimizasyon gerektiren bir çok probleme uygulanabileceği görülmektedir.
- PSO'nun bir çok algoritma ile hibrit olarak kullanılabileceği ortaya çıkmıştır.
- Özellikle başlangıç değerlerinin rastgele seçildiği ve bunun bir sorun teşkil ettiği bir çok bilimsel uygulamada bu başlangıç değerleri PSO tarafından temin edilerek diğer yöntemlerle hibrit bir algoritmanın üretilebileceği görülmüştür.

KAYNAKLAR

1. Bonyadi, M. R., Azghadi, M. R. and Hamed, H. S., "Population-based optimization algorithms for solving the travelling salesman problem", Traveling Salesman Problem, Federico Greco, ***InTech***, Vienna, 1-34 (2008).

2. Holland, J. H., "Adaptation in natural and artificial systems", ***Ann Arbor MIT Press***, MI, 1-228 (1975).

3. Rechenberg, I., "Cybernetic solution path of an experimental problem", ***Royal Aircraft Establishment***, Library Translation 1122, Farnborough, 297-318 (1965).

4. Schwefel, H. P., "Numerical optimization of computer models", ***Wiley & Sons***, New York, 1-398 (1981).

5. Koza, J. R., "Genetic programming: on the programming of computers by means of natural selection", ***MIT Press***., London, 79-237 (1992).

6. Göloğlu, C. ve Arslan, Y., "Zigzag machining surface roughness modelling using evolutionary approach", ***Journal of Intelligent Manufacturing***, 20 (2): 203-210 (2009).

7. Deb, K., "A population-based algorithm-generator for real-parameter optimization", ***Soft Computing A Fusion Of Foundation, Methodologies and Applications***, 9 (4): 236-253 (2004).

8. Dorigo, M. and Stutzle, T., "Ant colony optimization", ***MIT Press Prentice Hall***, London, 121-151 (2004).

9. Kirkpatrick, S., Gellat, C. D. and Vecchi, M. P., "Optimization by simulated annealing" ***Science,*** 220 (4598), 671-680 (1983).

10. Teodorovic, D., Lucic, P., Markovic, G. and Dell'Orco, M., "Bee colony optimization: principles and applications." ***8th Seminar on Neural Network Applications in Electrical Engineering***, Belgrade, 151-156 (2006).

11. Nocedal, J. and Wright, S. J., "Numerical optimization second edition" Series in Operations Research, ***Springer***, 349-352 (1999).

12. Leunberger, D. G., Ye, Y., "Linear and nonlinear programming third edition", International Series and Operation Research and Management Science, ***Springer***, 1-551 (2008).

13. Smith, A. E., Coit, D. W., Baeck, T., Fogel, D. and Michalewicz, Z. "Penalty functions", Handbook of Evolutionary Computation, ***A Joint Publication of Oxford University Press and Institute of Physics Publishing***, 1-11 (1995).

14. Gologlu, C. and Zeyveli, M., "A genetic approach to automate preliminary design of gear drives", ***Computers & Industrial Engineering***, 57 (3): 1043-1051, (2009).

15. Macdonald, D. C., "Linear programming", ***Students' Quarterly Journal***, 31 (122): 113-120 (1960).

16. Tucker, A. W., "Solving a matrix game by linear programming", ***IBM Journal of Research and Development***, 4 (5): 507-517 (1960).

17. Pearson, J. and Sridhar, R., "A discrete optimal control problem", ***IEEE Transactions on Automatic Control***, 11 (2): 171-174 (1966).

18. Huper, K. and Trumpf, J., "Newton-like methods for numerical optimization on manifolds", ***Signals, Systems and Computers Conference Record of the Thirty-Eighth Asilomar Conference on***, California, 136-139 (2004).

19. Clark, R. and Wheeler, P., "A self-adjusting control system with large initial error", ***Automatic Control IRE Transactions on***, 7 (1): 33-38 (1962).

20. Pantoja, J. F. O. and Mayne, D.Q., "A sequential quadratic programming algorithm for discrete optimal control problems with control inequality constraints", ***Decision and Control, Proceedings of the 28th IEEE Conference on***, Florida, 353-357 (1989).

21. Vieira, D. A. G., Lisboa, A. C. and Saldanha, R. R., "An enhanced ellipsoid method for electromagnetic devices optimization and design", ***Magnetics, IEEE Transactions on***, 46 (8): 2843-2851 (2010).

22. Kennedy, J. and Eberhart, R., "Particle swarm optimization.", ***Neural Networks Proceedings IEEE International Conference on***, Perth, 1942-1948 (1995).

23. Hu, X., Shi, Y. and Eberhart R., "Recent advances in particle swarm", ***Evolutionary Computation CEC2004 Congress on***, CEC, 1: 90-97 (2004).

24. Valle, Y. D., Vanayagamoorthy, G. K., Mohagheghi, S., Hernandez, J. C. and Harley R. G., "Particle swarm optimization: basic concepts, variants and applications in power system", ***IEEE Transactions on Evolutionary Computations***, 12 (2): 171-195 (2008).

25. Eberhart, R. and Kennedy, J., "A new optimizer using particle swarm theory", ***In: Sixth International Symposium on Micro Machine and Human Science***, Nagoya, 39-43 (1995).

26. Helwig, S., Neumann, F. and Wanka R., "Particle swarm optimization with velocity adaptation", ***International Conference on Adaptive and Intelligent Systems***, Klagenfurt, 146-151 (2009).

27. Robinson, J. and Rahmat-Samii, Y., "Particle swarm optimization in electromagnetics", ***IEEE Transactions on Antennas and Propagation***, 52 (2): 397-407 (2004).

28. Shi, Y. and Eberhart, R., "Empirical study of particle swarm optimization", ***Proc. of the Congress on Evolutionary Computation***, Washington, 3: 1945-1950 (1999).

29. Sun, J. L., Zeng, J. C. and Pan, J. S., "A particle swarm optimization with feasibility-based rules for mixed-variable optimization problem", ***Ninth International Conference on Hybrid Intelligent Systems***, 1: 543-547 (2009).

30. Sun, C., Zeng J. and Pan, J., "An improved particle swarm optimization with feasibility-based rules for constrained optimization problems", ***Fourth International Conference on Innovative Computing, Information and Control***, Taiwan, 897-903 (2009).

31. He, S., Prempain, E. and Wu Q. E., "An improved particle swarm optimizer for mechanical design optimization problems", ***Engineering Optimization***, 36 (5): 585-605, (2004).

32. Xu, S. and Rahmat-Samii, Y. (2007) "Boundary conditions in particle swarm optimization revisited" ***IEEE Transactions On Antennas And Propagation***, 55 (3): 760-765, (2007).

33. Das S., Abraham, A. and Konar, A., "Automatic clustering using an improved differential evolution algorithm", ***Systems Man and Cybernetics Part A: Systems and Humans IEEE Transactions on***, 38 (1): 218-237 (2008).

34. Chen, C. Y., Feng H. M. and Ye F., "Automatic particle swarm optimization clustering algorithm", ***International Journal Of Electrical Engineering***, 13 (4): 379-387 (2005).

35. Omran, M. G. H., Engelbrecht A. P. and Salman A., "Dynamic clustering using particle swarm optimization with application in unsupervised image classification", ***World Academy of Science***, 199-204 (2005).

36. Chou, C. H., Su, M. C. and Lai, E., "A new cluster validity measure and its application to image compression"***, Pattern Analysis & Applications***, 7 (2): 205-220 (2004).

37. Merwe, V. D. and Engelhrecht A. P., "Data clustering using particle swarm optimization", ***Evolutionary Computation, CEC '03 Congress on***, CEC, 215-220 (2003).

38. Na, S., Xumin, L. and Yong, G., "Research on k-means clustering algorithm", ***Third International Symposium on Intelligent Information Technology and Security Informatics***, Jinggangshan, 63-67 (2010).

39. Wilkin G. A. and Xiuzhen H., "K-Means clustering algorithms: implementation and comparison", ***Computer and Computational Sciences IMSCCS. Second International Multi-Symposiums on,*** Iowa City, 133-136 (2007).

40. Wang, W., Zhang, Y. and Li, Y., Zhang, X., "The global fuzzy c-means clustering algorithm", ***Intelligent Control and Automation. WCICA The Sixth World Congress on***, Dalian, 3604-3607 (2006).

41. Wang, Z., "Comparison of four kinds of fuzzy c-means clustering methods", ***Information Processing (ISIP) Third International Symposium on***, Qingdao, 563-566 (2010).

42. Tseng, L. Y. and Yang S. B., " Genetic algorithms for clustering, feature selection and classification", ***Neural Networks International Conference on***, Houston,1612-1616 (1997).

43. Kudova, P., "Clustering genetic algorithm", ***Database and Expert Systems Applications. DEXA '07. 18th International Workshop on***, Regensburg, 138-142 (2007).

44. Jımenez, J. F., Cuevas, F.J., Carpıo, J.M., "Genetic algorithms applied to clustering problem and data mining", ***Proceedings of the 7th WSEAS International Conference on Simulation Modelling and Optimization***, Beijing, 219-224 (2007).

45. Das, S., Abraham, A. and Konar, A., "Automatic kernel clustering with a multi-elitist particle swarm optimization algorithm", ***Science Direct Pattern Recognition Letters***, 29 (5): 689-699 (2008).

46. Premalatha, K. and Natarajan, A. M. ,"A new approach for data clustering based on pso with local search", ***Computer and Information Science***, 1 (4): 139-145 (2008).

47. Legancy, C., Juhasz, S. and Babos, A., "Cluster validity measurement techniques", ***AIKED'06 Proceedings of the 5th WSEAS International Conference on Artificial Intelligence Knowledge Engineering and Data Bases***, Wisconsin, 388-393 (2006).

48. Maulik, U. and Bandyopadhyay, S., "Performance evaluation of some clustering algorithms and validity indices", ***IEEE Transactions On Pattern Analyses And Machine Intelligence***, 24 (12): 1650-1654 (2002).

49. Falco, D., Cioppa, A. D. and Tarantino, E., "Evaluation of particle swarm optimization effectiveness in classification", ***Lecturer Notes in Computer Science***, 3849: 164-171 (2006).

50. Olesen, J. R., Cordero, J. and Zeng, Y., "Auto-Clustering using particle swarm optimization and bacterial foraging", ***Agents and Data Mining Interaction 4th International Workshop,*** Budapest, 5680: 69-83 (2009).

51. İnternet: UCL Mechine Learning Repository, "Iris dataset", **http://archive.ics.uci.edu/ml/datasets/Iris** (2011).

52. Bezdek, J. C., "Pattern recognition with fuzzy objective function algorithms", ***Kluwer Academic Publishers Norwell***, MA, 1-256 (1981).

53. Gologlu, C. and Mizrak, C., "An ıntegrated fuzzy logic approach to customer oriented product design", ***Journal of Engineering Design***, 22 (2): 113–127 (2011).

54. Wang, W., Zhang, Y., Li, Y. and X. Zhang, "The global fuzzy c-means clustering algorithm", ***Proceedings of the 6th World Congress on Intelligent Control and Automation,*** Dalian Chine, 3604-3607 (2006).

55. Izakian, H., Abraham, A. and Snášel, V., "Fuzzy clustering using hybrid fuzzy c-means and fuzzy particle swarm", ***World Congress on Nature & Biologically Inspired Computing*** , 1690-1694 (2009).

56. Runkler, T. A. and Katz, C., "Fuzzy clustering by particle swarm optimization", ***IEEE International Conference on Fuzzy Systems***, BC Canada, 601-608 (2006).

57. Yih, J., Lin, Y. and Liu, H., "Clustering analysis method based on fuzzy c-means algorithm of pso and pso with application in real data", ***International Journal Of Geology***, 89-98 (2007).

ÖZGEÇMİŞ

Yasin ORTAKCI 1981 yılında Karabük'te doğdu. İlkokulu Safranbolu Karıt Köyü İlkokulu'nda tamamladı. Ortaokulu eğitimini Safranbolu İmam Hatip Lisesi'nde lise eğitimini ise Karabük Demir Çelik Yabancı Dil Ağırlıklı Lisesi'nde tamamladı. 1999 yılında orta öğretimden mezun olduğu yıl Sakarya Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümünü kazanarak bu bölümde dört yıl lisans eğitimi gördü. 2003 yılında lisans diploması almaya hak kazandı. 2004-2005 yıllarında Antalya Hava Meydan Komutanlığı'nda asteğmen olarak vatani görevini yaptı. 2005 yılında Zonguldak Karaelmas Üniversitesi Safranbolu Meslek Yüksek Okulu'nda Okutman olarak göreve başladı. 2008 yılında Karabük Üniversitesi Fen Bilimleri Enstitüsü Elektronik-Bilgisayar Eğitimi Anabilim Dalında başladığı yüksek lisans eğitimini 2011 yılında Karabük Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalında tamamladı. 2007 yılından bu yana Karabük Üniversitesi'nde Okutman olarak görev yapmaktadır. Evli ve bir çocuk babasıdır.

ADRES BİLGİLERİ

Adres : Karabük Üniversitesi
Mühendislik Fakültesi
Balıklarkayası Mevkii / KARABÜK

Tel : (555) 487 8363
E-posta : yasinortakci@karabuk.edu.tr