T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

K-GEZGİN SATICI PROBLEMİNİN EMPERYALİST REKABETÇİ ALGORİTMASI İLE KÜMELEME TABANLI OPTİMİZASYONU

OKTAY KÖSE

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMAN

PROF. DR. PAKİZE ERDOĞMUŞ

DÜZCE, 2021

T.C.
DÜZCE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

K-GEZGİN SATICI PROBLEMİNİN EMPERYALİST REKABETÇİ ALGORİTMASI İLE KÜMELEME TABANLI OPTİMİZASYONU

Oktay KÖSE tarafından hazırlanan tez çalışması aşağıdaki jüri tarafından Düzce Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı

Prof. Dr. Pakize ERDOĞMUŞ

Düzce Üniversitesi

Jüri Üyeleri

Prof. Dr. Pakize ERDOĞMUŞ
Düzce Üniversitesi _____

Prof. Dr. Cihan KARAKUZU
Düzce Üniversitesi _____

Doç. Dr. Öğr. Üyesi Ferzan KATIRCIOĞLU
Düzce Üniversitesi _____

Tez Savunma Tarihi: 24/06/2021

BEYAN

Bu tez çalışmasının kendi çalışmam olduğunu, tezin planlanmasından yazımına kadar bütün aşamalarda etik dışı davranışımın olmadığını, bu tezdeki bütün bilgileri akademik ve etik kurallar içinde elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara kaynak gösterdiğimi ve bu kaynakları da kaynaklar listesine aldığımı, yine bu tezin çalışılması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını beyan ederim.

24 Haziran 2021

Oktay KÖSE

TEŞEKKÜR

Yüksek lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Prof. Dr. Pakize ERDOĞMUŞ'a en içten dileklerimle teşekkür ederim.

Bu çalışma boyunca yardımlarını ve desteklerini esirgemeyen sevgili aileme ve çalışma arkadaşlarıma sonsuz teşekkürlerimi sunarım.

24 Haziran 2021 **Oktay KÖSE**

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

ÇİZELGE LİSTESİ

Sayfa No

KISALTMALAR

BTA	Benzetilmiş tavlama algoritması
ERA	Emperyalist rekabetçi algoritması
GSP	Gezgin satıcı problemi
KGM	Karayolları Genel Müdürlüğü
KKA	Karınca kolonisi algoritması
NP-ZOR	Polinom zamanlı olmayan zor
PSO	Parçacık sürü optimizasyonu

SİMGELER

c_{ij}	i şehrinden j şehrine olan mesafe
C_n	Emperyalist maliyeti
c_n	Normalize edilmiş maliyet
i	Şehir noktası
j	Şehir noktası
$j_k(i)$	i noktasının ziyaret etmediği noktalar
n	Şehir sayısı
(n-1)!	Çözüm sayısı
N_{var}	Problem matrisi
N_{pop}	Başlangıç popülasyon matrisi
N_{imp}	Emperyalist ülke matrisi
N_{col}	Koloniler matrisi
q_0	Başlangıç feromen olasılığı
r	Bağ sayısı
x	Koloniler arası rastgele seçilen mesafe
x_{ij}	i şehri ile j şehri arasındaki seyahat kaydı
$\propto$	Feromen üstel ağırlık
β (KKA)	Buluşsal üstel ağırlık
β (ERA)	Yön değeri
γ	Sapmayı etkileyen parametre
η(i,j)	Seçilme ihtimali
θ	Sapma değeri
ξ	Kolonilerin gücü
ς(i,j)	Feromen miktarı

ÖZET

K-GEZGİN SATICI PROBLEMİNİN EMPERYALİST REKABETÇİ ALGORİTMASI İLE KÜMELEME TABANLI OPTİMİZASYONU

Oktay KÖSE
Düzce Üniversitesi
Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı
Yüksek Lisans Tezi
Danışman: Prof. Dr. Pakize ERDOĞMUŞ
Haziran 2021, 68 Sayfa

Hızla ilerleyen teknoloji ve insan ihtiyaçlarının çoğalması nedeniyle günlük yaşantıda zamanın önemi her geçen gün daha da artmaktadır. İnsanlar ihtiyaçlarını gidermenin yanında ihtiyaçlarına da bir an önce ulaşmayı hedeflemektedir. Gezgin Satıcı Problemi (GSP) de özellikle lojistik, ulaşım ve ürün tedarik alanların da kullanılan bir çözüm problemidir. Bu çalışma da Türkiye haritası üzerinde 81 il koordinatları ve iller arası mesafeler için matrisler oluşturularak, bu matrisler üzerinden yeni gelişmekte olan Emperyalist Rekabetçi Algoritması (ERA) ile en kısa tur bulunmaya çalışılmıştır. ERA ile bulunan en kısa tur, Benzetilmiş Tavlama Algoritması (BTA), Parçacık Sürü Optimizasyonu (PSO) ve Karınca Kolonisi Algoritmaları (KKA) ile bulunan en kısa turlar işe karşılaştırılmıştır. Bu karşılaştırmalar turların harita üzerinde gösterimleri, toplam mesafe ve yolculuk süreleri ile sağlanmıştır.

Ayrıca zamanın önemli olmasından dolayı 81 il için tek tur yerine, bu illeri 7, 8 ve 9 kümeye ayırarak daha avantajlı sonuçlar bulunmuştur. Oluşturulan kümeler için küme merkezleri arası ve küme iç turları için GSP çözümlemesi yapılmıştır. Tüm bu çıkan sonuçlar Karayolları Genel Müdürlüğü (KGM) sayfasındaki iller arası mesafeler ve yolculuk süreleri açısından değerlendirilmiştir.

Anahtar sözcükler: Gezgin satıcı problemi, Emperyalist rekabetçi algoritması, K-Ortalamalar kümeleme analizi

ABSTRACT

THE CLUSTER BASED OPTIMIZATION WITH THE IMPERIALIST COMPETITIVE ALGORITHM OF THE K-TRAVELING SALESMAN PROBLEM

Düzce University
Graduate School of Natural and Applied Sciences, Department of Computer Engineering
Master's Thesis
Supervisor: Prof. Dr. Pakize ERDOĞMUŞ
June 2021, 68 pages

Due to the rapidly advancing technology and the increase in human needs, the importance of time in daily life is increasing day by day. In addition to meeting their needs, people aim to reach their needs as soon as possible.

The Traveling Salesman Problem (TSP) is a solution problem that is used especially in logistics, transportation and product procurement areas. In this study, by creating matrices for 81 provincial coordinates and distances between cities on the map of Turkey, it was tried to find the shortest tour with the new developing Imperialist Competitive Algorithm (ICA) through these matrices. The shortest tours found with ICA were compared with the shortest tours found with Simulated Annealing Algorithm (SAA), Particle Swarm Optimization (PSO) and Ant Colony Algorithms (ACA). These comparisons are provided by the representations of the tours on the map, the total distance and travel times.

In addition, due to the importance of time, more advantageous results were found by dividing these provinces into 7, 8 and 9 clusters instead of a single tour for 81 provinces. TSP analysis was performed for cluster centers and cluster inbound rounds for the formed clusters. TSP analysis was performed for cluster centers and cluster inbound rounds for the formed clusters. All these results are evaluated in terms of distances and travel times between provinces on the General Directorate of Highways page.

Keywords: Traveling salesman problem, Imperialist competitive algorithm, K-means

1. GİRİŞ

Birçok bilim dalındaki problemlerin çözümünde optimizasyon tercih edilmektedir. Bir problemin çözüm kümesinin içinden verilen kısıtlayıcı şartlar altında en iyisinin bulunmasına optimizasyon denir. Matematiksel model de seçilen çözüm kümesi elemanları sistematik olarak probleme ait amaç fonksiyonu değişkenleri için uygulanır. Burada hedeflenen matematiksel fonksiyonun optimum değeri olan, fonksiyonun türevini sıfır yapan değeri bulmaktır [1].

Bir optimizasyon probleminde; amaç fonksiyonu, kısıtlar ve değişkenler olmak üzere üç unsur vardır. Optimizasyon problemlerinde genellikle amaç fonksiyonunu minimum yapan değerler aranır. Burada amaç fonksiyonu minimize edilerek problem çözülmeye çalışılır. Kesin çözüm yöntemleri ile tam sonucun bulunamadığı problemler için sezgisel yöntemler vazgeçilmez olmuşlardır. Kâğıt üzerinde bir problemin çözüm kümesi sayısını bulabiliriz fakat buradaki her değeri uygulamada tek tek denemek uzun sürelere mal olacaktır. Sezgisel yöntemler özellikle çözüm süresi açısından avantaj sağlamaktadır. Sezgisel yöntemlerin dezavantajı ise problemin her zaman tam sonucunu bulamaması ve yaklaşık değerlerde kalmasıdır [1].

Optimizasyon problemleri arasında büyük bir yere sahip güzergah belirleme problemleri de çözümü zor bulunanlar arasındadır. Fakat güzergah belirleme hayatımızda özellikle ticari alanda önem arz etmektedir. İşletmeler için gelişen piyasa şartlarında ürünlerinin kalitesinin yanında bu ürünlerin ulaşımının sağlanması da önemli bir konudur. Birçok işletme için değerlendirme ve şikâyet sebebi ürünlerin ulaşma(ulaşım) süresi ön planda bulunmaktadır. Özellikle kurumsal ve geniş bir coğrafyaya hitap eden işletmeler için ulaşımın kalitesini arttırmak için ideal bir ulaşım ağı kurulması ve bu ağın yönetimi önemlidir. Bu ulaşım ağı, ürünün üretiminden başlayıp kişiye ulaşana kadar ki süreci kapsamaktadır. Doğal olarak da bu geniş bir yelpazeye oluşturmakta ve buradaki akış şeması için yol haritasının iyi belirlenmesi gerekmektedir. Ayrıca işletmeler için zamanın yanında ulaşım maliyeti de büyük bir gider kalemidir.

Güzergah belirleme problemlerinden biri olan GSP, ilk olarak İngiliz ve İrlandalı matematikçiler tarafından ele alınmış, daha sonrada Harvard ve Viyana'da İrlandalı

matematikçi tarafından çalışılmıştır [2]. GSP, tanımlama açısından kolay fakat çözüm açısından zor bir problemdir. Tanımlanması yapıldığında bir küme üzerindeki noktaların birinden başlayıp hepsine sadece bir defa ve eksizsiz olarak uğrayarak tekrar başladığı noktaya dönme işlemidir. Nokta sayısı arttıkça problemin çözümsüzlüğü de artmaktadır [3].

GSP ile ilgili gelişmelere tarihsel açıdan bakılacak olursa; 1950-1960 yıllarında 49 şehirden oluşan bir GSP'de el ile en iyi optimal sonucu bulmuşlar ve daha iyisi olmadığını ispatlamışlardır. Amerikalı bilim adamları tarafından, tek bir ağaçtan oluşan optimalden %1 gevşeme sağlayan bir yöntem bulmuşlardır [4]. Çinli ve Kanadalı bilim adamları tarafından, Lin-Kernighan algoritması olarak da bilinen köşe değiştirmeli bir yöntem önermişlerdir. Turdan k adet köşe çıkarıp onun yerine k adet başka köşeler ekleyerek değiştirme işlemi uygulamışlardır [5]. İngiliz bilim adamı tarafından, hala en sıkı yaklaşım oranı olarak da bilinen, optimal turdan 3/2 oranında daha kötü tur oluşturma yöntemine dayanmaktadır [6]. Amerikalı bilim adamı, k cinsinden genişletilmiş yüksek verimli bir ikili arama ağacı yapısında k-d ağacı yöntemini uygulamıştır [7]. Alman bilim adamı tarafından, üzerinde 50 yıl çalışılmış ve içinde birçok test problemini içeren TSLIB kitaplığını düzenlemiş ve yayınlamıştır [8]. Danimarkalı bilim adamı tarafından, Lin-Kernighan algoritmasını geliştirerek Lin-Kernighan-Helsgaun adını verdiği yeni sürümü yayınlamıştır. Ayrıca 1904711 şehirlik bir rekoru, bu algoritmayı Concorde programı ile birlikte kullanarak kırmıştır [9].

Böyle önemli bir yere sahip olan GSP birçok çalışmada kullanılmış birçok algoritmanın performansını ölçmek için test edilmiş ve çok değişik alanlardaki sorunlar için çözüm odağı olmuştur. GSP sadece ulaşım problemlerinin çözümlemesi haricinde, devre kartlarının tasarımı, bilgisayar kablolama, ekip planlama, vinç güzergahının planlanması, baz istasyonlarının yerleşim yerlerinin tespiti gibi bir çok alanda faydalanılan bir problemdir [3].

Bu çalışmada da yeni bulunmuş olan ERA'nın GSP üzerindeki performansı, tek gezgin satıcı üzerinden BTA, KKA ve PSO ile karşılaştırılmıştır. Karşılaştırma ülkemizin 81 ili üzerinde yapılmış ve iller arası mesafeler kullanılmıştır. İlk olarak eşit parametreler ile daha sonrada optimal parametreler ile verdikleri tepkiler ve sonuçlar değerlendirilmiştir. Bulunan en kısa turlar harita üzerinde kuş bakışı ve karayolları çizimleri yapılmış olup algoritma çıktısı ile gerçek ulaşım yolları değerlendirmesi yapılmıştır. Ayrıca dört

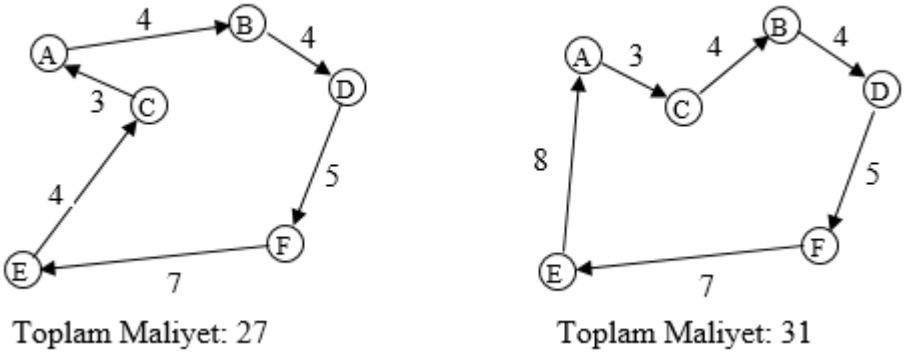
algoritmanın buldukları en kısa turun dışında program çalışma sürelerinin de değerlendirmesi yapılmıştır.

Günümüz teknolojisinde ve yaşam hızında zamanın ne kadar önemli olduğunu varsayarsak 81 ilin tek gezgin satıcı üzerinden yapılması çok fazla gerçekçilik göstermemektedir. Bu yüzden de k-ortalamalar algoritması ile 81 il 7, 8 ve 9 kümeye ayrılarak küme sayısının, küme merkezleri ve küme iç turlarının değerlendirmesi yapılmıştır. 81 il kümeleme yapıldığında maksimum bir kümeye düşen il sayısı 18 olduğundan ERA performası çok iyi çıkmış ve her denemede en kısa tur bulunmuştur. Bu yüzdende küme değerlendirmesi bulunan küme iç turları mesafelerinin yolculuk süreleri üzerinden yapılmıştır.

En son olarak tek gezgin satıcı ile k gezgin satıcının uygulanabilirlik açısından, zaman açısından karşılaştırmaları yapılmış olup, sektöre katkıları değerlendirilmiştir.

2. GEZGİN SATICI PROBLEMİ

GSP, bir şehirden başlayıp tekrar başladığı şehre dönene kadar bütün şehirlere yalnızca bir kez uğrayıp en kısa turu tamamlama işlemidir [10]. Burada iki önemli kural vardır. Birincisi bütün şehirlere mutlaka uğranmalı, ikincisi uğranan şehre bir daha uğranmamalıdır. Bu kuralları sağlayan en kısa tura ayrıca Hamilton çevrimi de denir [11]. Şekil 2.1'de Hamilton çevrimi ile ilgili örnekler gösterilmiştir.



Şekil 2.1 Hamilton çevrim örnekleri.

Problemlerin sınıflandırma türlerinden biri de polinom zamanda çözülüp çözülmediğine göredir. P sınıfında değerlendirilen kolay problemler polinom fonksiyonu artan bir algoritma ile çözülebilirken, GSP'nin çözümünde ise polinom zamanlı bir algoritma bulunmamaktadır. Bu yüzden de NP-ZOR problem kategorisinde değerlendirilir. Bundan dolayı da özellikle şehir sayısının çok olduğu bir problemde kesin sonucu bulmak imkânsıza yakındır. Hamilton çevriminin şehir sayısına göre zaman karşılaştırması Çizelge 2.1'de gösterilmiştir [3].

GSP bir şehirden başlayıp tekrar aynı şehre dönmeyi amaçladığı için başlangıç şehrinden sonra geri kalan şehirlerle tur ihtimalleri hesaplandığında, n adet şehir varsa gerçek çözüm sayısı (n-1)!'e eşittir [12]. Şehir sayısının az olduğu problemlerde bütün turlar bulunup bunların içinden en kısa olanı seçmek kesin sonucu verir. Fakat şehir sayısı arttıkça kesin tur sonucunu bulmak imkânsızdır. Bu durumlarda sezgisel veya meta-sezgisel yöntemleri kullanmak daha uygun olacaktır [13].

Çizelge 2.1. Hamilton çevriminin karşılaştırılması.

Düğüm Sayısı	Döngü Sayısı (n-1)!	Gerekli Zaman
12	39.916.800	0.004 saniye
13	479.001.600	0.05 saniye
14	6.227.020.800	1 saniye
15	87.178.291.200	9 saniye
16	1.307.647.368.000	2 dakika
17	2.1 * 1013	35 dakika
18	3.6 * 1014	10 saat
19	6.4 * 1015	7.5 gün
20	1.2 * 1017	140 gün
21	2.4 * 1018	7.5 yıl
22	5.1 * 1019	160 yıl
23	1.1 * 1021	3.500 yıl
24	2.6 * 1022	82.000 yıl
25	6.2 * 1023	2 milyon yıl

2.1. GEZGİN SATICI PROBLEMİNİN MATEMATİKSEL GÖSTERİMİ

GSP problem türü olarak bakıldığında bir minimizasyon problemidir. a_{ij}; i ile j şehri arasındaki uzaklığı, x_{ij}; i'den j'ye gidiş varsa 1, yoksa 0 değerini, N'de şehir sayısını tutar. Denklem (2.1)'de amaç fonksiyonu gösterilmiştir. Denklem (2.2) her şehre giriş olma, denklem (2.3) ise her şehirden çıkış olma kısıtını belirtmektedir. Denklem (2.4) i şehrinden j şehrine gidiş varsa j şehrinden i şehrine gidiş olmaması gerektiğini ifade etmektedir. Denklem (2.5) tüm şehirlerin gezilmesi gerektiğini ifade etmektedir. Denklem (2.6) tüm şehirlerde 0 veya 1 değerini almalıdır [14]

$$Min\ Z = \sum_{i=1}^{N}(\sum_{j=1}^{N} a_{ij} * x_{ij}) \tag{2.1}$$

$$\sum_{i=1}^{N}(\sum_{j=1}^{N} x_{ij}) = 1 \tag{2.2}$$

$$\sum_{j=1}^{N}(\sum_{i=1}^{N} x_{ij}) = 1 \tag{2.3}$$

$$x_{ij} + x_{ji} \leq 1 \qquad i: 1,2,3,\dots,N \tag{2.4}$$

$$j: 1,2,3,\dots,N$$

$$\sum_{i=1}^{N}\sum_{j=1}^{N}(x_{ij}) = N \tag{2.5}$$

$$x_{ij} \in \{0,1\} \qquad i: 1,2,3,\dots,N \tag{2.6}$$

$$j: 1,2,3,\dots,N$$

2.2. GEZGİN SATICI PROBLEMİ TÜRLERİ

Literatürde standart GSP'nin kullanım alanının farklılaştırılması veya kısıtlamaların değiştirilmesi sonucu farklı türleri elde edilmiştir

Simetrik GSP, şehirlerarası mesafelerin hem giderken hem de dönerken eşit olduğu durumlardır. Asimetrik GSP ise eşit olmadığı durumlardır [1].

Kârlı GSP, tüm şehirlere uğrayarak tur oluşturma şartı aramamaktadır. Burada her şehir bir kâr gücüne sahiptir ve her şehre ulaşmak için bir maliyete yapılmaktadır. Bu iki değerin hesabı yapılarak bazı şehirler turdan çıkarılmaktadır. Bu da amaç fonksiyonunda kısıt olarak yazılmaktadır [15].

Zaman Pencereli GSP, tur oluştururken her şehre ulaşılması gereken bir zaman aralığını göz önünde bulundurmak zorundadır. Bunu da kısıt olarak eklemektedir [16].

İki Depolu Heterojen GSP, iki adet deponun olduğu ve iki farklı araç ile en düşük maliyetli turu hesaplamaktadır [17].

Çoklu GSP'de, şehirler bölgelere ayrılır ve her bölge için ayrı bir tur hesaplanır ve toplamda en düşük maliyet ortaya çıkar. GSP'den zor olan kısmı ise grupların önceden oluşturulmasıdır [18].

Açık Döngülü GSP, en iyi turu hesaplarken her şehre uğrama şartını aramakla birlikte tekrar başladığı şehre dönme şartını göz ardı etmektedir [19].

Belirsiz GSP, günlük hayattaki zor şart ve durumları problem içine dahil edip hesaplamaları buna göre yapmaktadır. Bu şartlar, trafik yoğunluğu, hava şartları, yol çalışması gibi etkenlerdir. En iyi tur hesabı için ağırlaştırmalar veya belirsizlik teoremleri uygulanarak çözüm aranmaktadır [20].

Genelleştirilmiş GSP, n adet merkezden ve bu merkezlere bağlı m adet şehirlerden oluşur. Burada amaç her şehir ile bağlı olduğu merkez arasında bir kere gidip gelinecek ve daha sonra merkezler arası turlar olacak. Bu işlem yapılırken de en iyi tur hesabı yapılacak [21].

2.3. GEZGİN SATICI PROBLEMİ ÇÖZÜM TEKNİKLERİ

GSP tanımsal olarak kolay bir problem olmasına karşın çözüm aşamasında o kadar kolay değildir. Özellikle şehir sayısı arttıkça problemin çözümü zorlaşmakta, süre ve işlem

açısından maliyeti artmaktadır [22]. GSP çözümünde kesin yöntemler, sezgisel yöntemler ve metasezgisel yöntemler kullanılmıştır.

Kesin yöntemler kesin sonuçları hedef almaktadır, ancak bu az şehir sayısı olan problemler için iyi performans göstermektedir. Şehir sayısı arttıkça kesin yöntemler hem süre hem de maliyet açısından yüksek değerlere ulaşmaktadır [22]. Sayma Yöntemi, Dal-Sınır Yöntemi, Dal Kesme Yöntemi kesin yöntemler için örneklerdir.

Sezgisel yöntemler kesin yöntemler gibi kesin çözümlere giden yolları göstermemektedir. Sezgisel yöntemler tüm çözüm kümesinin içinden belli bir çözüm kümesi seçilir ve bunlara ilgili sezgisel yöntemin şartları uygulanarak mevcut çözüm kümesi değiştirilir. Bir problem için çalışan bir sezgisel yöntem başka bir problem için çalışmayabilir.

Sezgisel algoritmalara örnek verecek olursak; En Yakın Komşu Sezgiseli, Açgözlü Sezgiseli, Ekleme Sezgiseli, Christofides Sezgiseli, R-opt Sezgiseli, Lin-Kernighan Sezgiseli en büyük örneklerdir [23].

Metasezgisel yöntemler, klasik yöntemlerle çözüm bulunamayan problemlerin sınırlı süre içerisinde uygun ve etkin çözümlerini bulan yöntemlerdir. Sezgisel yöntemler gibi problemin kendine ait bir çözüm sunmayan her problem için mutlaka etkin ve uygun çözüm bulan yöntemlerdir.

Metasezgisel yöntemleri algoritma akışı olarak göstermek istersek [24];

- Başlangıcın rastgele seçilen bir çözümden başlaması
- Parametrelere uygun komşu çözümlerin araştırılması
- Daha iyi sonuçlar bulunmuşsa anlık durumun güncellenmesi
- Kabul edilen kurallara uygun çözüm bulunana kadar işlemin yenilenmesi

2.3.1. Parçacık Sürü Optimizasyonu

PSO, 1995 yılında Kennedy ve Eberhart tarafından geliştirilmiş popülasyon tabanlı ve sürü zekasına dayalı bir optimizasyon yöntemidir. Konuşarak birbiri ile iletişim kuran insanlar gibi balıkların ve kuşların yön tayini ve ortak hareket etmelerini sağlayan sosyal zekâlarını kullanma esasına dayanmaktadır [21],[22].

PSO doğrusal olmayan problemlerin çözümünde kullanılan rastgele arama algoritmasıdır. PSO'da her parçacık bir çözüm demektir ve başlangıç adımında bir popülasyon oluşturulur. Her parçacık diğer optimal çözümler ile kendi arasında karşılaştırma yaparak en iyiye ulaşmaya çalışır. Bu işlemi yaparken de herhangi bir metot geliştirmesine gerek

yoktur. Bu da problemlerin çözümünde karmaşıklığı ortadan kaldırır [27]. Her parçacık, kendi doğrultusunda sabit olarak hareket etme, kendi geçmişine ait konumlardan faydalanarak hareket etme ve komşularının konumlarından faydalanarak hareket etme olarak üç şekilde konum değiştirirler [28].

Parçacıklar eski pozisyonları arasındaki en iyi değeri hafızasına alır ve bu o parçacığın kişisel en iyi değeri ($pbest_i$) olur. Bütün sürü parçacıklarının en iyi konum değeri ise sürünün en iyi değeri ($gbest_i$) olur. Parçacıklar daha iyi çözümler bulmak için kendi en iyi değerlerini ve sürünün en iyi değerini karşılaştırarak hız ve yön tespiti yaparlar. Parçacıklar için komşuluk bağları önemlidir. Çünkü parçacıklar bir sonraki adımda hızlarını ve yönlerini belirlerken komşularının en iyi değerini ve kendi en iyi değerini göz önünde bulundurarak ayarlayacaklardır [29].

Örneğin, D bilinmeyenli bir optimizasyon problemi için, D boyutlu bir vektör ile gösterilmeli ve bu popülasyon n adet parçacıktan oluşuyor ise bir parçacığın pozisyon vektöründen oluşan pozisyon matrisi Denklem (2.7)'deki gibi olacaktır.

$$X = \begin{bmatrix} x_{11}x_{12} \dots x_{1D} \\ x_{21}x_{22} \dots x_{2D} \\ \dots \quad \dots \quad \dots \quad \dots \\ x_{i1}x_{i2} \dots x_{iD} \\ \dots \quad \dots \quad \dots \quad \dots \\ x_{n1}x_{n2} \dots x_{nD} \end{bmatrix} \tag{2.7}$$

Pozisyon matrisine göre $i.$ parçacığın pozisyonu $i.$ satırda bulunmaktadır ve $X_i = [x_{i1}, x_{i2}, \dots, x_{iD}]$ olarak gösterilir. O parçacığa ait kişisel en iyi pozisyonu ise $pbest_i = [p_{i1}, p_{i2}, \dots, p_{iD}]$ şeklinde gösterilir. Popülasyona ait en iyi pozisyon yani global en iyi, $gbest = [g_1, g_2, \dots, g_D]$ ile gösterilir. Parçacıkların pozisyonları her bir döngü de hız matrisine göre güncellenir ve $i.$ satıra ait hız vektörü de $V_i = [V_{i1}, V_{i2}, \dots, V_{iD}]$ ile gösterilir. En iyiler bulunduktan sonra güncelleme Denklem (2.8)'deki formül ile yapılmaktadır. Pozisyon vektörü güncellemesi de Denklem (2.9) ile gösterilmiştir [29], [30].

$$V_{i,n}^{k+1} = w.V_{i,n}^{k} + c_1.rnd_1^k\left(pbest_{i,n}^k - x_{i,n}^k\right) + c_2.rnd_2^k\left(gbest^k - x_{i,n}^k\right) \tag{2.8}$$

$$x_i^{k+1} = x_i^k + V_i^{k+1} \qquad i = 1, 2, \dots, S; \qquad n = 1, 2, \dots, N \tag{2.9}$$

Denklemlerde kullanılan w atalet ağırlığını temsil eder ve 0 ile1 arasında bir değer olmalıdır. Atalet ağırlığı azaldıkça yerel aramayı, arttıkça da global aramayı kolaylaştırmaktadır. Parçacığın kendi tecrübelerine göre hareket sağlayan c_1 ve sürüdeki

diğer parçacıkların tecrübelerine göre hareket sağlayan c_2 hızlandırma sabiti olarak adlandırılan değişkenlerdir. rnd ise düzgün dağılımı sağlayan rastgele sayılardır. i parçacığın indeksini, k döngü sayısını temsil eder [29],[30].

Atalet ağırlığının azalması da Denklem (2.10) ile hesaplanabilir. Buradaki T değeri döngü sayısını belirtmektedir.

$$w = w_{maks} - T\frac{w_{maks}-w_{min}}{T_{maks}} \tag{2.10}$$

Parçacığın bir yerden bir yere hareketi Şekil 2.2'de gösterilmiştir. Buradaki değişkenleri şöyle açıklayabiliriz.

x_i^k: i parçacığının k döngüsündeki konumu

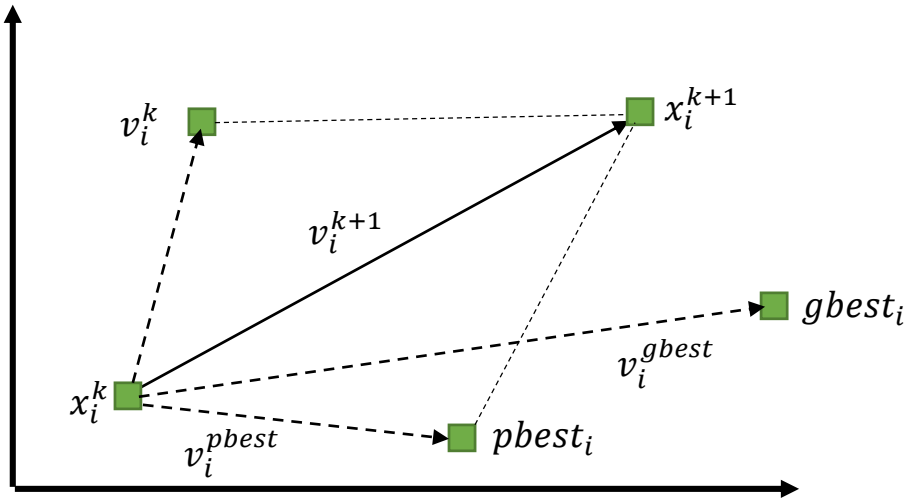
v_i^k: i parçacığının k döngüsündeki hızı

$gbest_i$: sürüdeki en iyi konuma sahip parçacığın konumu

$pbest_i$: i parçacığına ait en iyi konum

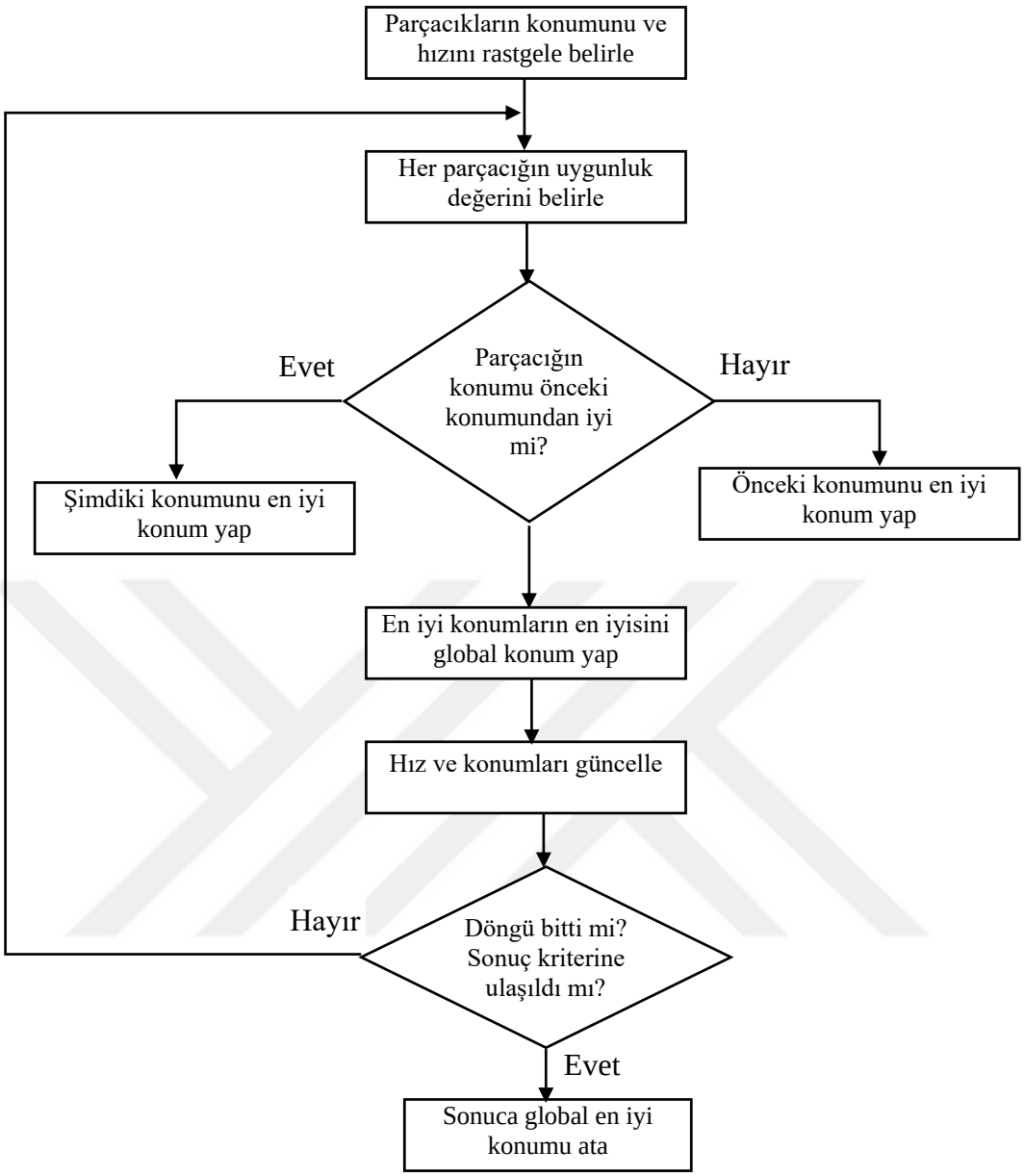
x_i^{k+1}: i parçacığının $k+1$ döngüsündeki konumu

v_i^{k+1}: i parçacığının $k+1$ döngüsündeki hızı



Şekil 2.2. Parçacığın pozisyon değiştirmesi.

PSO çalışma prensibi Şekil 2.3'teki akış diyagramı üzerinde gösterilmiştir [31].



Şekil 2.3. PSO akış diyagramı.

PSO'da kullanılan parametreleri aşağıdaki gibi listeler ve açıklayabiliriz [29],[31].

Sürü Büyüklüğü: Parçacık adediyle ölçülen değerdir. Parçacık adedi çözüm sayısı ile doğru, çözüm süresi ile ters orantılıdır. Ayrıca parçacık adedinin faza olması çözümde kullanılan döngünün de az olmasını sağlar. Çoğunluk olarak parçacık adedi 10 ile 50 arasında alınır. Özel problemler için 50'den fazla da alınmaktadır.

Döngü Sayısı: En iyi sonuç için probleme göre değişik değerler alınabilir. PSO'da iyi bir çözüm için döngü sayısı fazla alınmalıdır. Fakat bu da çözüm süresini arttıracaktır.

Hızlanma Katsayıları: Parçacıklar iki değere göre konumlarını değiştirirler. Bunlardan bir tanesi parçacığın kendine ait olan en iyi konum, diğeri ise sürüdeki bütün parçacıkların

sahip olduğu en iyi konumdur. Parçacıklar hızlanma katsayısını bu iki değere göre belirler. Hızlanma katsayısı sıfır olarak alınmışsa parçacık sahip olduğu hızla hareket etmeye devam eder. Bu değerin küçük alınması hedefe uluşma süresini geciktirirken, büyük bir değer alınması da hem hedefe ulaşmayı hızlandırır hem de yanlış bir hareket sonucu hedef alanın görmezden gelinmesini sağlar. Yapılan araştırmalar sonucu hem parçacığın kendi değeri hem de sürünün en iyi değeri olarak 2 değeri kullanılmıştır.

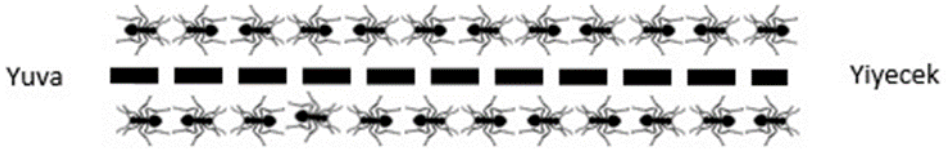
Eylemsizlik Ağırlığı: Parçacığın kendi en iyi sonuçları arasında araması ile sürüdeki en iyi sonuçlar arasındaki aramayı dengeler. Eylemsizlik ağırlığının büyük olması yerel arama becerisini, küçük olması global arama becerisini arttırır.

Sonlandırma Kriteri: Büyük boyutlu problemlerin gerçek çözümlerini bulmak hem imkânsız hemde uzun süreler aldığı için sezgisel algoritmalar yakın değerler bulmayı hedeflemektedir. Bu yüzdende algoritma bir yerde durdurularak en iyi sonuç elde edilmeye çalışılır. Durdurma kriteri olarak genellikle döngü sayısı seçilir ve amaç daha az döngü ile daha iyi sonuç elde etmektir. Bir başka durdurma kriteri olarak da kullanıcı tarafından bir hedef sonuç belirlenir ve bu sonuca ulaştığında algoritma durdurulur.

2.3.2. Karınca Kolonisi Algoritması

Gerçek hayattan esinlenerek ve hayvanların hareketleri baz alınarak matematiksel modeller oluşturmak kombinasyon hesapları içeren problemlerin çözümünde etkili olmaktadır. KKA rastgele arama mantığına dayanan, sürü merkezli ve en yakın çözümü bulmayı hedefleyen bir metasezgisel yöntemdir. İlk defa 1989 yılında Gross ve diğerleri tarafından ele alınmıştır. Karıncaların yuvalarından yiyeceklere giden en kısa yolu bulma ihtiyaçlarından ortaya çıkmıştır [32].

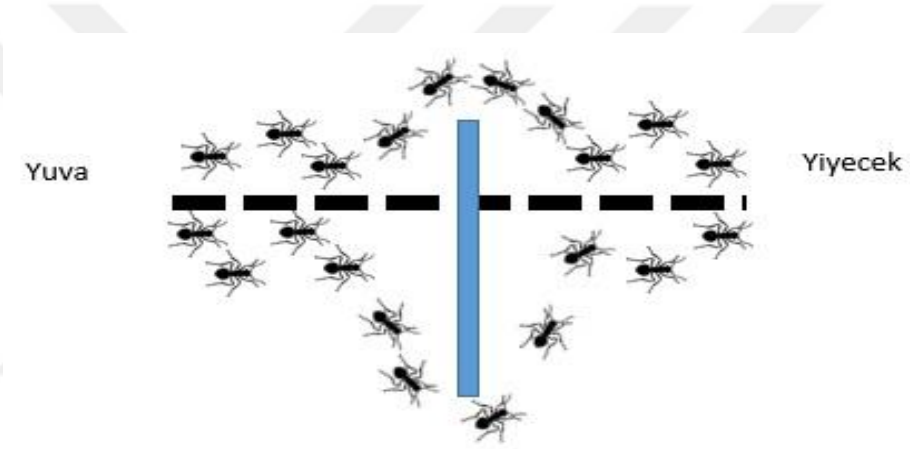
Gerçek karıncalar, yiyecek kaynağı ile yuva arasındaki en kısa yolu görsel bir mekanizma kullanmadan bulma yetisine sahiptirler. Ayrıca herhangi bir engel durumunda ise kendi yetileri ile çözüm bulup yeni en kısa yolu bulurlar [33]. Şekil 2.4'da karıncaların yuva ve yiyecek kaynağı arasındaki doğrusal yol gösterilmektedir.



Şekil 2.4. Karıncaların yuva ve yiyecek kaynağı arasında izlediği doğrusal yol.

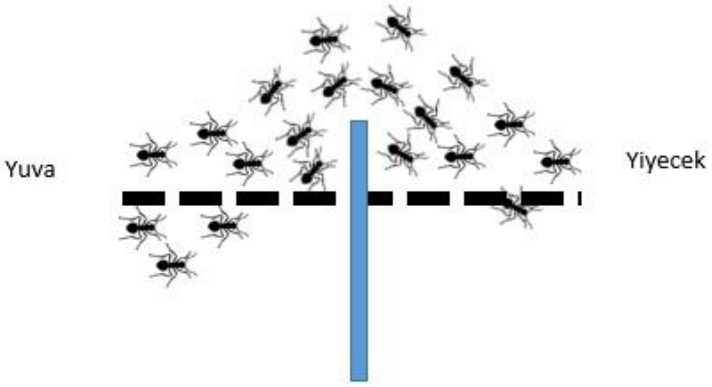
Karıncalar yiyecek bulduklarında kaynaktan yuvaya dönerken buldukların yiyeceğin miktarı ve kalitesine göre dönüş yoluna feromen koyar. Feromen sayesinde diğer karıncalar bu yiyecek kaynağı hakkında bilgiye sahip olurlar [34]. Diğer karıncalar yiyecek kaynağı seçerken feromen miktarı fazla olan yolu tercih eder. Yiyecek kaynağının yakın olması daha çok tercih edileceğinden bu yoldaki feromen miktarı da giderek artacaktır. Bu içgüdüsel davranış yiyecek kaynağına giden en kısa yolu nasıl bulduklarının açıklamasıdır [33].

Şekil 2.5'deki gibi yiyecek kaynağı ile yuva arasındaki yolda bir engel oluştuğunda karıncalar yoldaki feromen izlerini takip edemezler. Bu yüzden de olası olan diğer yolları denemek zorundadırlar. Karıncaların olası yolları seçme ihtimali birbirine eşittir [34],[35].



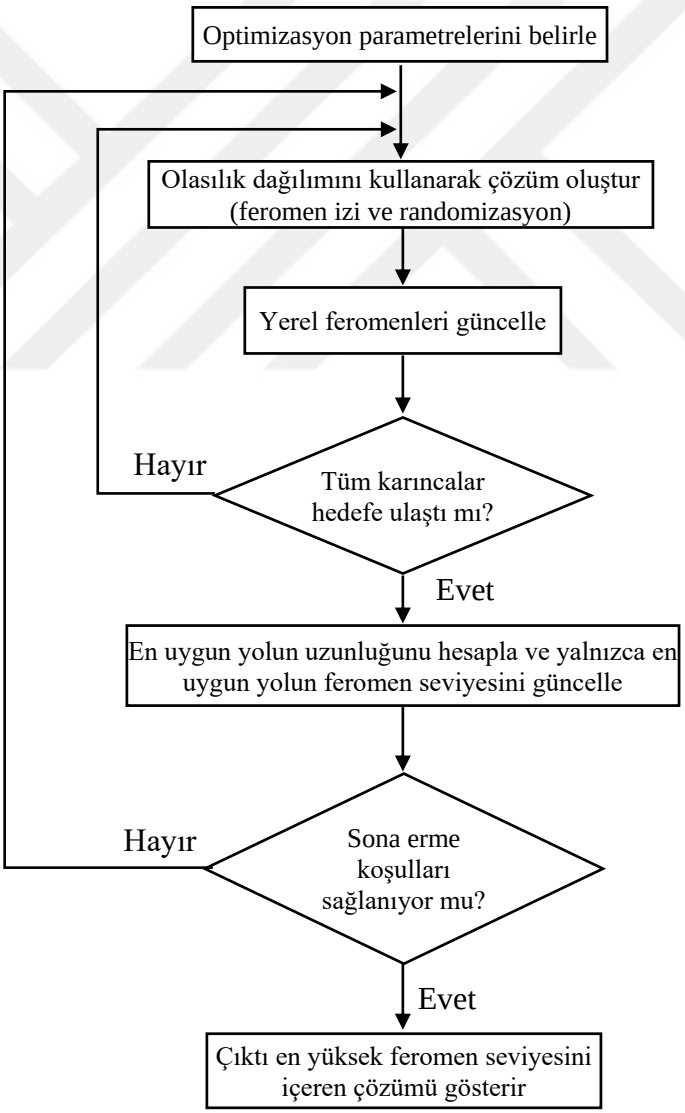
Şekil 2.5. Karıncaların yiyecek kaynağı yolundaki engele karşı eşit yol seçimi.

Başlangıçta karıncalar iki yol içinde eşit davransalar da belli bir süre sonra otomatik olarak kısa yolu seçmeye başlayacaklardır. Karıncaların eşit hızda ve eşit miktarda feromen bıraktığı varsayılırsa kısa yoldan daha fazla karınca geçeceği için buradaki feromen miktarı giderek artacaktır. Bundan sonra gelen karıncalar da yol seçimi yaparken bu feromen miktarını göz önünde bulunduracaklardır. Bu işlem başlangıçta bir süre alsa da genel olarak bakıldığında toplamdaki yol sürecini kısaltacaktır [34],[36]. Şekil 2.6'de karıncaların en kısa yol seçimi gösterilmiştir.



Şekil 2.6. Karıncaların en kısa yolu seçmeleri.

KKA'na ait akış diyagramı Şekil 2.7'de gösterilmiştir.



Şekil 2.7. KKO akış diyagramı.

KKO algoritmasında ilk turda gidilecek bütün yolların eşit seçilme hakkı olması için bütün yollara atanan feromen miktarı aynıdır. İlk turun sonucunda feromen miktarları, karıncaların geçiş sayısına, yolun seçilme sayısına ve yolun uzunluğuna göre güncellenir. Bir sonraki turda ise karıncalar artık güncellenen feromen miktarına göre yollarını seçerler. KKO'nda yol seçimi iki şekilde yapılır. İlk olarak, feromen miktarının fazla olduğu q_0 olasılıkla seçilmesidir. i ve j noktaları arasındaki feromen miktarı $\tau(i,j)$ ile gösterilir. Ayrıca α ve β parametreleri ile ayarlamalar yaparak i noktasından gidecek karıncanın hedef noktası aşağıdaki gibi belirlenir.

$$j = \max_{u \in j_k(i)} \{[\tau(i,u)]^{\alpha} x [\eta(i,u)]^{\beta}\} \tag{2.2}$$

Bu denklemde $\eta(i,j)$ seçilme ihtimali parametresi, i ve j noktaları arasındaki mesafenin tersine $(\frac{1}{\delta(i,j)})$ eşittir.

$$\eta(i,j) = \frac{1}{\delta(i,j)} \tag{2.3}$$

İkinci seçenek, gidilecek yolun üzerindeki feromen miktarına göre seçmektir. Böylece seçim olasılığı $1 - q_0$ oranındadır. $j_k(i)$, i noktasındaki karıncanın ziyaret edilmemiş ve gidebileceği noktaları gösterir. Tüm noktalar için seçilme ihtimali aşağıdaki gibi hesaplanır.

$$P_k(i,j) = \frac{\{[\tau(i,u)]^{\alpha} x [\eta(i,u)]^{\beta}\}}{\sum_{u \in j_k(i)} \{[\tau(i,u)]^{\alpha} x [\eta(i,u)]^{\beta}\}}, j \in j_k(i) \tag{2.4}$$

Bu ihtimaller doğrultusunda yollar seçilir ve feromen miktarı fazla olan yolun ihtimali daha fazladır.

Döngü tüm karıncalar yollarını tamamladığında son bulur ve döngü sonucuna göre yollardaki feromen miktarı güncellenir. Feromen güncellemesi, arıtma ve buharlaşma fonksiyonları ile yapılır. Arıtma işlemi ile kısa yolların önemi arttırılır buharlaşma işlemi ile de yollardaki feromen miktarı belli bir oranda azaltılarak yapılır. Yeni feromen değerini aşağıdaki gibi hesaplarız.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t+1) \tag{2.5}$$

Bu denklemde $\tau_{ij}(t)$, t döngüsüne gelene kadar biriken feromen miktarını, $\Delta\tau_{ij}^{k}(t+1)$, t döngüsündeki feromen miktarını gösterir. ρ parametresi ise feromen buharlaşma oranını gösterir ve $(0 \leq \rho \leq 1)$ aralığında seçilir.

t. döngüdeki feromen miktarı aşağıdaki denklemle hesaplanır.

$$\Delta\tau_{ij}^{k}(t+1) = \frac{1}{L^{k}(t+1)} \tag{2.6}$$

k karıncasının toplam yol miktarı $L^{k}(t+1)$ ile gösterilir. Yollara ait feromen güncellemeleri ile her döngüde seçilen yol kısaltılmaya çalışılır. Böylece algoritmanın dinamik bir hal alması sağlanır [37],[38].

2.3.3. Benzetilmiş Tavlama Algoritması

BTA birden fazla parametrelere sahip fonksiyonların minimum ve maksimum değerlerinin bulunması ve en önemlisi çok sayıda minimumlara sahip doğrusal olmayan fonksiyonların minimum değerlerini bulmak için tasarlanmıştır [39].

BTA, 1983 yılında Kirkpatrick tarafından demirin işlenerek elverişli hale getirilmesi için uygulanan tavlama tekniğinden esinlenerek geliştirilmiştir. Tavlama işlemi, demirin yüksek sıcaklıklara hızla yükseltilerek daha sonra yavaş yavaş soğumasını sağlayarak yapılmaktadır. Burada demir en yüksek sıcaklığa çıkana kadar içindeki parçacıklar gelişigüzel sıvaya dönüşmekte ve buradan yavaşça soğutulma yapıldığında düzenli bir yapıya sahip demir haline gelmektedir [40],[41].

BTA yaklaşımı aşağıdaki dört parametreyi kullanmaktadır.

Sıcaklık (T): Sıcaklık değerin gösterir.

Alfa (α): Soğutma oranın gösterir.

Epsilon (ε): Donma noktasını gösterir.

Döngü sayısı (S): Her sıcaklığın döngü sayısını gösterir.

Ayrıca algoritma çalışması için üç farklı değer daha kullanılır.

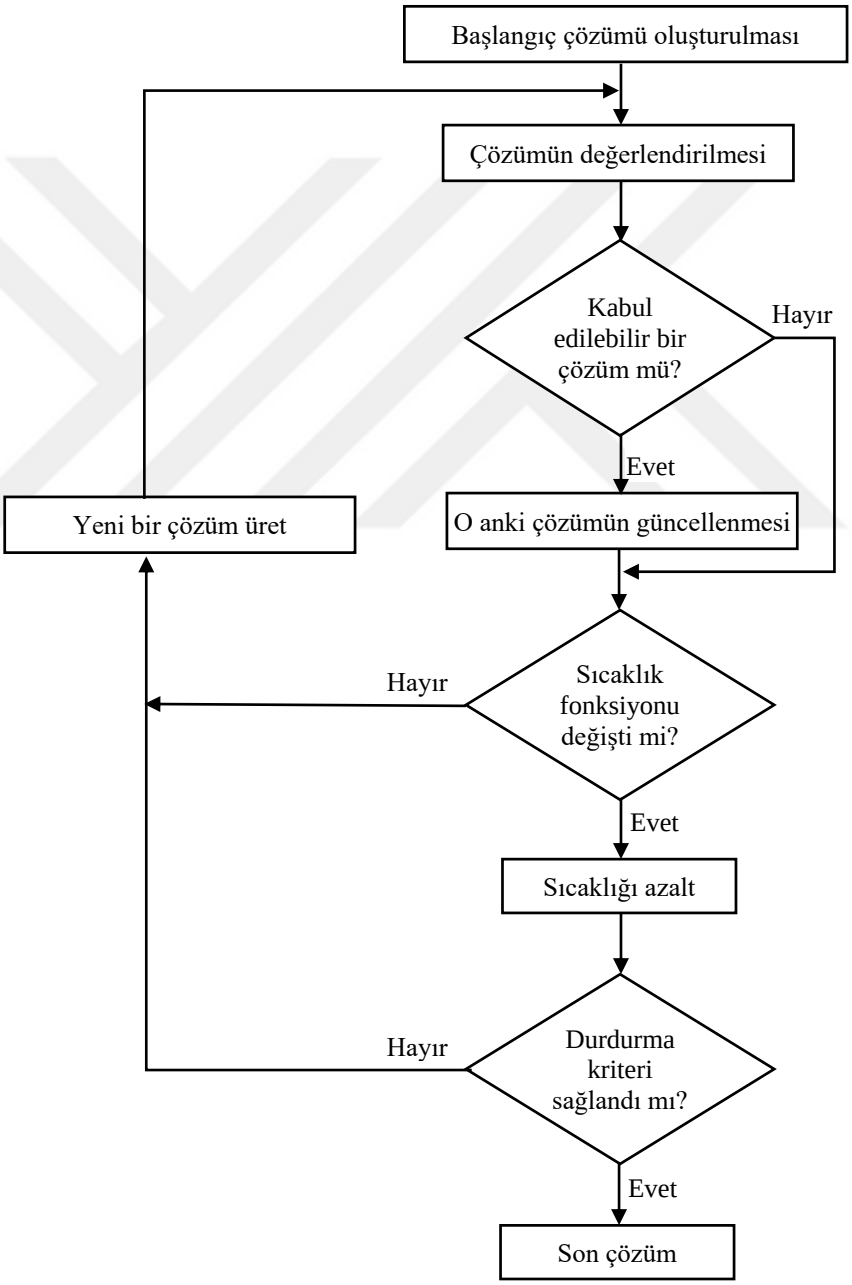
Delta (ΔC): Önceki çözüm ile mevcut çözüm arasındaki fark.

Boltzmann dağılımı (P): Yeni çözümün geçerliliği için kıyaslama değeri.

Maliyet fonksiyonu (C_x): x çözümünün maliyet fonksiyonu.

BTA'nın optimizasyon problemindeki karşılıklarına bakıldığında, demirin katı haldeki durumu problemin çözümlerine ve bu durumlardaki enerji, çözümlerin amaç fonksiyon değerlerini temsil eder. Optimal çözüm minimum enerjiye, yerel optimum çözüm ise hızlı soğutma işlemine karşılık gelir [42].

BTA'nın performansında soğutma işleminin payı büyüktür. Tavlama planı metalin sıvı hale gelecek kadar T sıcaklık değerine çıkarılarak kurulur. Bu sıcaklık değeri bütün çözüm ihtimallerini deneyecek kadar yüksek bir başlangıç değeridir. Daha sonra sıcaklık değeri yavaş yavaş azaltılmaya başlanır. Her sıcaklık değerinde çalıştırılacak döngü sayısı bir sabit değer seçilir ve uygun olan denge durumuna ulaşılmak amaçlanmaktadır. Bu işlemler sonucunda çıkan sonuçlar belli bir süre sonra aynı olmaya başladığında algoritma durdurulur. Böylece algoritmanın donma durumu denilen son nokta bulunur. BTA'ya ait akış diyagramı Şekil 2.8'da gösterilmiştir [43].



Şekil 2.8. BTA akış diyagramı.

Algoritmadan optimal çözüm bulmak için kullanılan parametreler için uygun değerler seçilmek zorundadır. Seçilen parametre ve değerlerinin belirlenmesine tavlama veya soğutma planı denir. Tavlama planı için aşağıdaki parametreler kullanılmaktadır [44].

Başlangıç sıcaklığı: Başlangıç sıcaklığı tavlama yöntemiyle bulunmaya çalışılan sabit değerlerin aranacağı aralığı ifade eder. Bu değer ne kadar büyük seçilirse arama alanı da o kadar geniş olacaktır. Fakat gereğinden yüksek bir değer seçildiğinde gereksiz arama yapıp uzun süreceği gibi gereğinden küçük bir değer seçildiğinde de algoritmanın yerel minimumlara takılma ihtimali artacaktır. Kesin bir değere sahip değilse yüksek seçilmesi her zaman için avantaj sağlayacaktır.

Soğutma oranı: Soğutma oranı α ile temsil edilen 0 ile 1 arasında olan bir değerdir. Bu değer 1'e yakın seçilirse soğuma yavaş olur, 0'a yakın seçilirse soğuma hızlı olacaktır. Bu yüzden seçilecek değer önemlidir ve duruma göre farklılıklar içerir. Soğuma hızlı olursa demir sert ve dayanıklı olur, yavaş olursa demir esneklik kazanır. Metalin kullanım amacına göre seçileceğinden dolayı BTA'da da problemin türüne göre değer verilir. Algoritma çalışması hızlı olursa çözüm süresi kısa olur ama en yakın veya kesin sonuç bulunamayabilir. Algoritma çalışması yavaş olursa süre uzar ama daha kesin ve yakın sonuçlar elde edilir. Problem çözümünde kesin sonuçlar süreden daha önemli olduğu için α değeri 1'e ne kadar yakın olursa daha iyidir.

Döngü sayısı: Döngü sayısı her sıcaklık değeri için arama yapılacak döngü sayısını göstermektedir. Döngü sayısı sona ulaştığında sıcaklık değeri α değeri ile çarpılarak sıcaklık değeri düşürülür. Döngü sayısı ile soğutma oranı ters orantıya sahiptir. Soğutma oranı arttırılır ve döngü sayısı da arttırılırsa arada kalan bazı alanların araması yapılamaz. Bu yüzden soğuma oranı yükseltilirken döngü sayısı azaltılırsa her sıcaklık değeri arasındaki fark azalacağı için daha detaylı arama yapılacaktır.

Durdurma kriteri: Döngünün belli bir koşulu sağlandığında durdurulma işlemini ifade etmektedir. BTA'da en yaygın kullanılan durdurma kriteri sıcaklığın belli bir değere ulaşmasıdır. Bu değer genellikle parçaların soğuduğu 0 °C olarak seçilir. Ayrıca döngü belli bir süre sonra değişmez değerler almaya başladığında da durdurulabilir. Bunların dışında durdurma kriteri olarak zaman da seçilebilir. Algoritmanın çalışma süresi tanımlanır ve o süre dolduğunda döngü durdurulur.

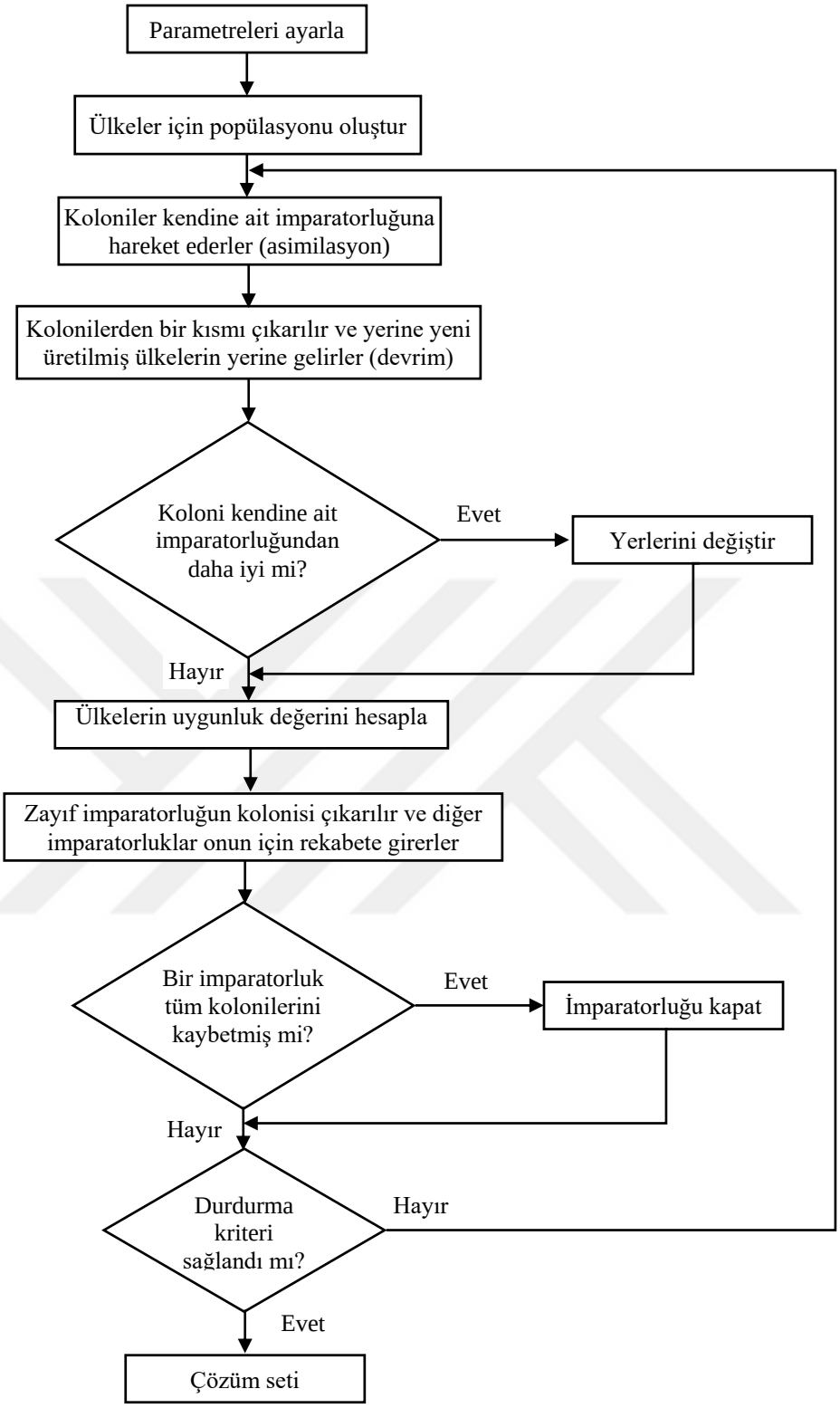
3. UYGULAMADA KULLANILAN ALGORİTMALAR VE YÖNTEMLER

3.1. EMPERYALİST REKABETÇİ ALGORİTMASI

Tüm evrimsel algoritmalar gibi ERA'nın da bir başlangıç popülasyonuna sahiptir. Bu popülasyondaki her üye yeryüzündeki bir ülke olarak değerlendirilir ve bunların arasındaki en güçlü üyeler emperyalist olarak belirlenir. Geride kalan diğer üyeler ise bu emperyalist ülkelerin kolonileri olarak atanır. Atama işlemi emperyalist ülkelerin güçlerine oranla paylaştırılır. Böylece imparatorluklar kurulmuş olur.

İmparatorluklar içinde koloniler emperyalist ülkelere doğru yaklaşmaya başlarlar. İmparatorlukların potansiyelleri emperyalistler ve onlara bağlı kolonilerin güçleriyle orantılıdır. Bu oranın asıl payı emperyalist ülkenin gücüne bağlıdır. Kolonilerin güçleri ise ortalama değerlerinin belli bir oranda eklenmesiyle hesaplanır.

Algoritma çalışmaya başladığında imparatorluklar arası savaş başlar. Savaşta emperyalist ülkeler diğer imparatorluklardaki kolonileri bünyelerine katarak güçlerini arttırırlar veya ellerindeki kolonileri kaybederek güçlerini azaltırlar. Savaş devam ettikçe güçsüz imparatorluklar yok olmaya başlayacak ve en sonunda güçlü tek bir imparatorluk kalacaktır. Diğer tüm ülkelerde kalan bu emperyalistin kolonisi olacaktır. Artık tüm koloniler emperyalist ülkenin gücüne sahip olacaktır [45]. ERA'ya ait akış diyagramı Şekil 3.1'de [46] ve ERA'ait sözde kod Şekil 3.2'de gösterilmiştir [47].



Şekil 3.1. ERA akış diyagramı.

1. Parametreleri başlat
2. Popülasyonu rastgele oluştur
3. İmparatorluğu başlat

   $For\ i = 1\ to\ N_{pop}$ (N_{pop} = popülasyon sayısı)

   Değerlendirme maliyetini hesaplamak c_i
   Tüm popülasyon için hesaplanan maliyeti c_i'yi azalan düzende sıralayın

   N_{pop}'dan N_{imp}'i (emperyalist ülke sayısı) seç

   Her emperyalistin maliyetini normalleştirin C_n

   Her emperyalistin normalleştirilmiş gücünü hesaplayın P_n

   Kalan ülkeleri emperyalistlere emanet etmek N_{col}

   Döngüyü sonlandır
4. Asimilasyon, devrim, emperyalist rekabet süreçleri

   $For\ j = 1\ to\ N_{imp}$

   Koloniyi ilgili emperyalistlere doğru hareket ettirin

   Asimile olmuş ülkelerin maliyetlerini hesaplamak

   Yeni kolonide devrim yap

   Eğer yeni koloninin maliyeti emperyalistin maliyetinden azsa

   Sömürge ve emperyalistin konumunu değiştir

   En zayıf imparatorluktan en zayıf koloniyi seç ve ona sahip olma olasılığı en yüksek olan imparatorluğa ata

   Döngüyü sonlandır
5. Eleme süreci

   Eğer kolonisi olmayan emperyalist varsa

   Emperyalistleri yok et
6. Durma koşuluna ulaşılana kadar

Şekil 3.2. ERA sözde kodu.

3.1.1. Başlangıç İmparatorluklarını Üretme

Algoritma da her değişken bir dizi içinde ifade edilir. ERA'da ki her ülke GA'da ki kromozomla aynı şeyi ifade etmektedir. N_{var} boyutlu bir problem için her ülke $1XN_{var}$ dizisi ile gösterilir. Oluşturulan bu dizi denklem (3.1)'deki gibi oluşturulur [48].

$$country = [p_1, p_2, \dots, p_{N_{var}}] \tag{3.1}$$

Ülkedeki değişken değerleri noktalı kayan sayılarla ifade edilir. Ülke potansiyeli $(cost)p_1, p_2, \dots, p_{N_{var}}$ değişkenlerindeki fonksiyonlar ile hesaplanır ve denklem (3.2)'deki gibi gösterilir [49].

$$cost = f(contry) = f\left(p_1, p_2, \dots, p_{N_{var}}\right) \tag{3.2}$$

Algoritmaya başlamadan önce N_{pop} boyutunda popülasyon üretilir. En yüksek güçteki N_{imp} kadar ülke emperyalist, diğer geri kalan ülkeler ise N_{col} kadarı emperyalistlere bağlı kolonilerdir.

$$N_{col} = N_{pop} - N_{imp} \tag{3.3}$$

İmparatorluklar emperyalist ülkelerin güçleri oranında kolonileri dağıtarak oluşturulur. En güçlü emperyalist ülke daha fazla koloniye sahiptir. Bu orantıyı sağlamak için denklem (3.4) kullanılır [49].

$$C_n = maks_i\{c_i\} - c_n \tag{3.4}$$

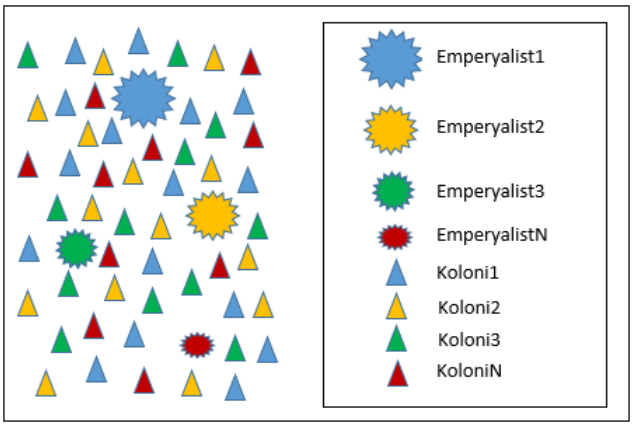
Burada C_n, n. emperyalistin maliyeti, c_n ise normalize edilmiş maliyetidir. Bütün emperyalistlerin normalize edilmiş maliyetlerini içeren, her bir emperyalistin normalize edilmiş gücü denklem (3.5)'deki gibi gösterilir.

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \tag{3.5}$$

Bir emperyalistin normalize edilmiş gücü imparatorluktaki koloni sayısını gösterir. Bu sayıda denklem (3.6)'deki gibi olacaktır.

$$N.C_n = round\{p_n.N_{col}\} \tag{3.6}$$

N_{col} tüm kolonilerin sayısını, $N.C_n$ ise n. imparatorluğun başlangıç koloni sayısın gösterir. Emperyalistlere $N.C_n$ koloni rastgele olarak seçilmektedir. Böylece n. imparatorluk oluşturulmaktadır. Şekil 3.3'de başlangıç imparatorlukları ve emperyalistin gücüne göre koloni sayıları gösterilmektedir [25].



Şekil 3.3. Başlangıç popülasyonu oluşturma.

Başlatma işlemi daha detaylı olması için aşağıdaki örnekle açıklanmıştır. Ülke sayısı 50 olan bir problemde 3 ülke emperyalist olarak, geri kalan 47 ülke ise koloni olarak seçilmiştir. Emperyalist olan 3 ülkenin maliyetleri sırası ile 100, 150 ve 180 olarak belirlenmiştir [47].

$N_{imp} = 3, N_{col} = 47$

Üç ülkenin maliyetleri şu şekilde gösterilir.

$c_1 = 100, c_2 = 150, c_3 = 180,\ \ maks\{c_1, c_2, c_3\} = 180$

Denklem (3.4)'deki formüle göre emperyalistlerin normalize edilmiş maliyetleri,

$C_1 = 180 - 100 = 80,\ C_2 = 180 - 150 = 30, C_3 = 180 - 180 = 0$

Denklem (3.5)'e göre emperyalistlerin güçleri,

$p_1 = 0.73, p_2 = 0.27, p_3 = 0$

Denklem (3.6)'ya göre emperyalistlerin normalize edilmiş güçleri,

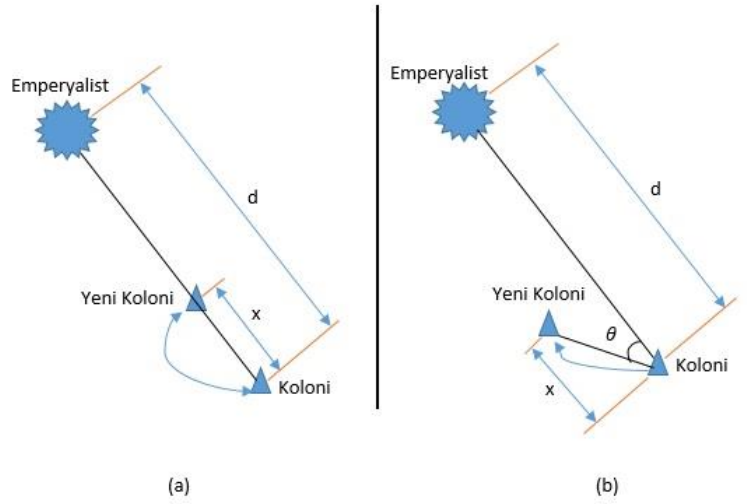
$NC_1 = round\{0.73 * 47\} = 34, NC_2 = round\{0.27 * 47\} = 13, NC_3 = \{0 * 47\} = 47$

3.1.2. Kolonilerin Başka Emperyalistlere Doğru Hareketleri

Belli bir süre sonra emperyalist ülkeler güçlerine oranla koloni sayılarını arttırmaya başlarlar. Bu işlem kolonilerin emperyaliste doğru hareketi ile olur. Şekil 3.4. a) Kolonilerin emperyaliste hareketi. b) Kolooninin yeni pozisyonu.'da bu işlem gösterilmiştir. Buradaki hareket emperyaliste doğru yönelen vektörel bir harekettir. Hareket mesafesi olarak seçilen x değeri rastgele seçilmiştir.

$$x \sim U(0, \beta \times d) \tag{3.7}$$

d emperyalist ile koloni arasındaki uzaklığı belirtir. Kolonininin emperyaliste yaklaşması için $\beta > 1$ olmalıdır. Koloniler emperyalistlere yaklaşırken farklı noktalara ulaşılması için Şekil 3.4. a) Kolonilerin emperyaliste hareketi. b) Kolonininin yeni pozisyonu.'de gösterildiği gibi hareket vektörüne rastgele bir θ sapma değeri eklenir [50].



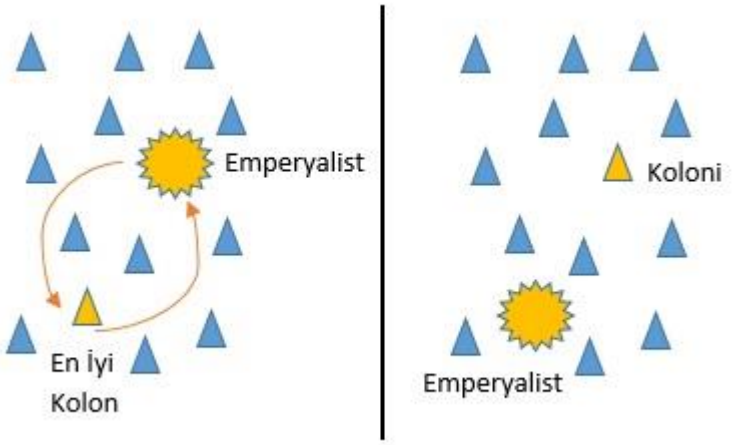
Şekil 3.4. a) Kolonilerin emperyaliste hareketi. b) Kolonininin yeni pozisyonu.

Denklem (3.8)'deki γ sapmanın değerini ayarlayan bir parametredir. β ve γ değerleri rastgele alınmış değerlerdir.

$$\theta \sim U(-\gamma, \gamma) \tag{3.8}$$

3.1.3. Emperyalist ve Kolonininin Yer Değiştirmesi

Emperyalistler sadece kolonilerin kendilerine doğru hareketi ile birbirlerine yaklaşmazlar. Kolonininin emperyaliste yaklaşması daha fazla maliyetli olabilir. Kolonininin bulunduğu yer emperyalistin yerinden daha iyi konumda olabilir ve bu durumda emperyalist ile koloni yer değiştirebilir. Bu işlem tek imparatorluk kalana kadar devam eder. Şekil 3.5'de emperyalist ile koloni yer değişimi gösterilmiştir [51].



Şekil 3.5. Emperyalist ile koloninin yer değişimi.

3.1.4. İmparatorluğun Toplam Gücü

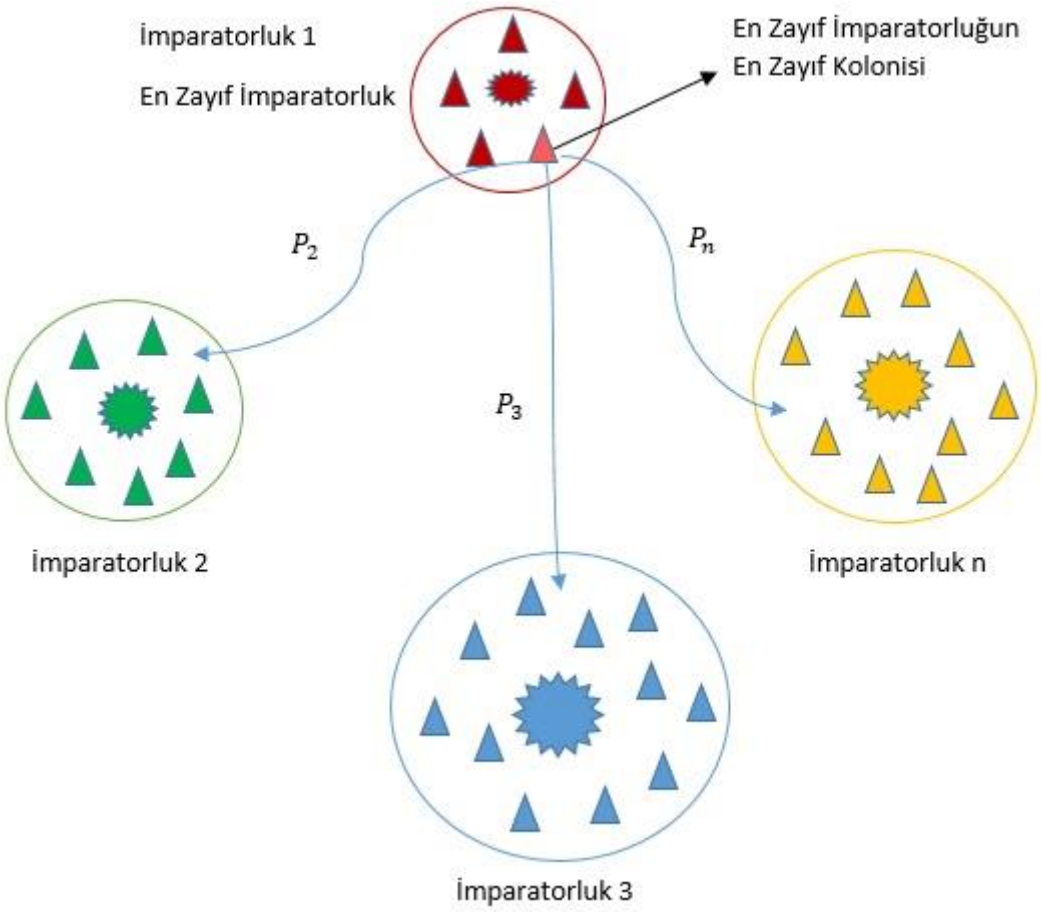
Bir imparatorluğun gücü genel olarak emperyalistin gücüne bağlıdır ve onla orantılıdır. Kolonilerin gücü çok fazla olmamak kaydıyla imparatorluk gücüne etki eder. İmparatorluğun toplam güç hesabı denklem (3.9)'de gösterilmiştir.

$$T.C_n = Cost(imperialist_n) + \xi\ mean\{Cost(colonies\ of\ empire_n)\} \quad (3.9)$$

İmparatorluğun toplam gücü $T.C_n$ ile ifade edilir ve ξ değeri kolonilerin gücünü etkileyen bir değişkendir. ξ 1'den küçük bir değer alınır ve birçok uygulama genellikle 0,1 değeri kullanılmıştır. Bu değer ne kadar büyürse kolonilerin imparatorluk içindeki etkisi o oranda artar [49].

3.1.5. Emperyalistik Yarış

Her imparatorluk gücünü daha fazla arttırmak için kendinden daha güçsüz olan imparatorlukların kolonilerini kendilerine katmaya çalışırlar. Emperyalistik yarışta güçlü imparatorlukların gücünü arttırması ve güçsüz imparatorlukların gücünü kaybetmesi üzerine kuruludur. Güçsüz imparatorluğa ait koloni diğer imparatorluklar tarafından ele geçirilmeye çalışılır ve en güçlü olan imparatorluğun şansı daha fazladır. Şekil 3.6'de bu işlem gösterilmiştir [52].



Şekil 3.6. Emperyalist yarış.

İlk olarak her imparatorluğa ait toplam güç hesaplanır ve bu oranda kolonileri kendine katma olasılığı belirlenmiş olur. Bu işlem denklem (3.10)'da gösterilmiştir.

$$N.T.C_n = T.C_n - maks_i\{T.C_i\} \tag{3.10}$$

Burada $T.C_n$ imparatorluğa ait toplam maliyeti, $N.T.C_n$ normalize edilmiş toplam maliyeti ifade etmektedir.

Kolonilerin hangi imparatorluğa gitme ihtimalini belirleyen P vektörü denklem (3.11)'deki gibi oluşturulur.

$$P = \left[p_{P_1}, p_{P_2}, \dots, p_{PN_{imp}}\right] \tag{3.11}$$

Daha sonra ***P*** ile aynı ölçüde olan R vektörü rastgele bir şekilde oluşturulur.

$$R = \left[r_1, r_2, \dots, r_{N_{imp}}\right] \tag{3.12}$$

$$r_1, r_2, \dots, r_{N_{imp}} \sim U(0,1)$$

D vektörü P vektöründen R vektörü çıkarılarak elde edilir. D vektörünün maksimum indeksi ile ilgili olan imparatorluğun kolonileri elde edilir. D vektörünün hesabı denklem (3.13)'de gösterilmiştir.

$$D = P - R = \left[D_1, D_2, \dots, D_{N_{imp}}\right] = \left[p_{P_1} - r_1, p_{P_2} - r_2, \dots, p_{PN_{imp}} - r_{N_{imp}}\right] \quad (3.13)$$

Yukarıda anlatılanları bir örnekle açıklamak daha iyi olacaktır. Üç emperyalist ülkeye ait toplam maliyetler sırasıyla 10, 20 ve 30 olsun [47].

$TC_1 = 10, TC_2 = 20, TC_3 = 30$

Her emperyalistin sahip olma ihtimalini hesaplamak için her emperyalistin toplam maliyetinin aşağıdaki gibi normalize edilmesi gerekir.

$\underset{1\le i\le 3}{\text{maks}}\, TC_i = 30$

$NTC_1 = TC_1 - \underset{1\le i\le 3}{\text{maks}}\, TC_i = 10 - 30 = -20$

$NTC_2 = TC_2 - \underset{1\le i\le 3}{\text{maks}}\, TC_i = 20 - 30 = -10$

$NTC_3 = TC_3 - \underset{1\le i\le 3}{\text{maks}}\, TC_i = 30 - 30 = 0$

Her bir emperyalistin sahip olma olasılığı şu şekilde hesaplanır,

$$p_1 = \left|\frac{(-20)}{(-20) + (-10) + 0}\right| = 0.67$$

$$p_2 = \left|\frac{(-10)}{(-20) + (-10) + 0}\right| = 0.33$$

$$p_1 = \left|\frac{0}{(-20) + (-10) + 0}\right| = 0$$

En düşük maliyete sahip olan emperyalistin sahip olma olasılığı en yüksek, en yüksek maliyete sahip emperyalistin sahip olma olasılığı ise sıfır çıkmıştır. Bulunan olasılıklara göre P vektörü aşağıdaki gibi hesaplanır.

$P = |0.67\ ,0.33\ ,0|$

Aynı boyutta P'ye sahip R vektörü aşağıdaki gibi tek tip fonksiyon kullanılarak hesaplanabilir.

$R = |0.28\ ,0.55\ ,0.96|$

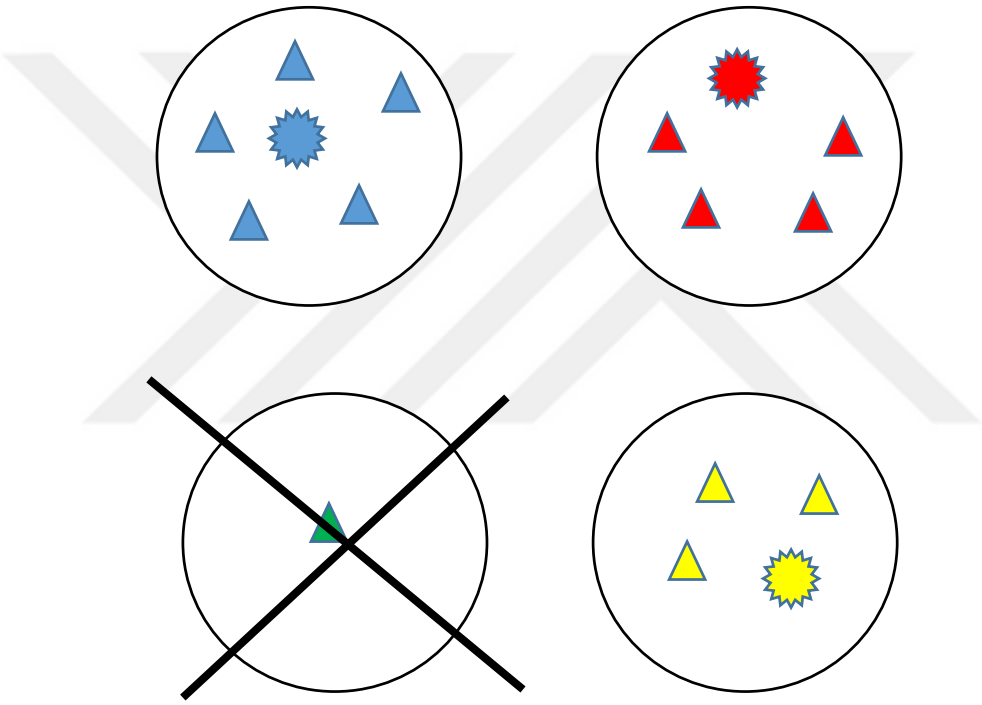
D vektörü ise P vektöründen R vektörünü çıkararak hesaplanır.

$$D = P - R = [0.39\ , -0.42\ , -0.96]$$

D vektörünün en büyük elemanı ve birinci indisteki birinci emperyalisti 0.39 değerine sahip olan emperyalisttir. Böylece bu emperyalist, en zayıf imparatorluğa ait en zayıf kolonini sahibi olur.

3.1.6. Zayıf İmparatorlukları Çökertmek

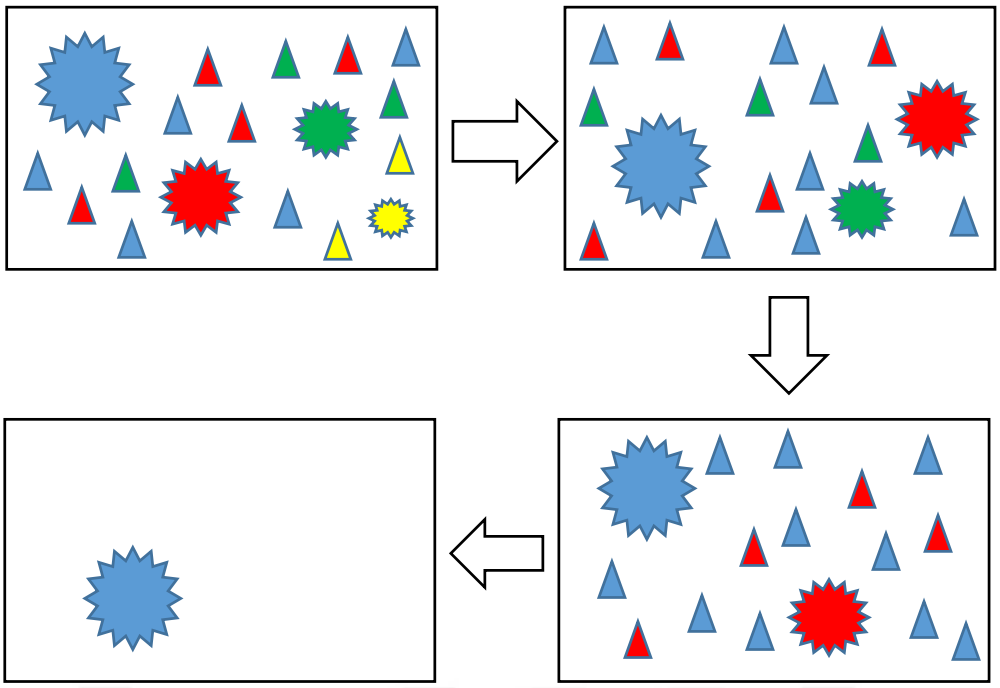
Güçlü imparatorluklara göre zayıf kalanlar sırasıyla kolonilerini kaybedeceklerdir. Böylece güçlü imparatorluklar daha güçlü hale gelecek ve zamanla zayıf imparatorluklar çökmek zorunda kalacaklardır. Şekil 3.7'da bu durum görsel olarak gösterilmiştir [47].



Şekil 3.7. En zayıf imparatorluğu çökertmek.

3.1.7. Yakınsama

ERA'da diğer algoritmalar gibi önceden belirlenmiş döngü sayısı ve çalışma süresi gibi şartlar sağlanıncaya kadar çalışmaya devam eder. Tüm imparatorlukların çöküp tek bir imparatorluğun kaldığı zaman ideal durdurma kriteri için uygundur. Bu durum da Şekil 3.8'de gösterilmiştir [47].



Şekil 3.8. ERA yakınsama örneği.

3.2. ERA PARAMETRELERİNİN ÇÖZÜM KALİTESİNE ETKİSİ

Metasezgisel algoritmaların başarısı, algoritma içindeki parametrelerin değerlerine bağlıdır. Belirlenen parametre değeri ne kadar iyiyse algoritmanın performansı da o derece iyi olacaktır. Bir parametre için en iyi değeri bulmakta çok kolay değildir. Bir parametrenin optimal değeri problemin büyüklüğü, karmaşıklığı veya kullanıcının izin verdiği çalışma süresine göre değişkenlik gösterir. Aşağıda parametrelerin ayarlanması için literatüre dayalı olarak açıklama yapılmıştır [47].

3.2.1. Sapma Parametresi (θ)

Sapma parametresi en kritik parametrelerden biri olup, küçük değerler seçildiğinde sömürü artarken, büyük değerler seçildiğinde ise aramayı arttırır. Sapma değerini belirlemenin yolu, büyük bir değer verip daha sonrada kademeli olarak küçük değerlere düşürmektir. Literatüre göre θ değeri için $\pi/4$ verildiğinde hem problemin doğruluğu hem de arama süresinin iyileştiği görülmüştür.

3.2.2. Yön Parametresi (β)

Yön parametresi, emperyalistlerin kolonileri kendi imparatorluklarına dahil etme sürecine koloninin ne kadar yaklaştığını gösterir. Sömürü ve keşif süreci için etken rol

konumundadır. Eğer küçük bir değer alırsa sömürüyü hızlandırır, büyük bir değer alırsa da keşfi artırır.

3.2.3. Ülke Sayısı (N)

Ülke sayısı çözüm doğruluğu ve çözüm süresi açısından belirleyici bir parametredir. Ülke sayısı az ise çözüm süresi kısa ama ideal çözümü bulmak zor olur. Büyük olduğunda ise çözüm süresi uzarken daha doğru sonuçlar bulunabilir. Emperyalist ülkeleri belirlerken de genellikle toplam ülke sayısının %10 ile %13'ü arasında bir değer alınır ve geri kalan ülkelerde koloni olarak seçilir.

3.2.4. İmparatorluğun Kolonilerinin Ortalama Gücü İle İlişkili Katsayısı (ξ)

İmparatorlukların güçleri hesaplanırken kolonilerin ortalama gücü de imparatorluk üzerinde etki etmektedir. Pozitif bir değer alır ve kolonilerin kümülatif gücünün, imparatorluğun gücüne etkisinin oranını belirlemek için kullanılır. Büyük değerler seçildiğinde koloni gücünün imparatorluğun toplam gücündeki etkisini arttırırken, küçük değerlerde imparatorluğun gücünü sadece emperyalist ülke gücüne bırakmış olur.

3.3. ERA AVANTAJ VE DEZAVANTAJLARI

ERA'nın dikkat çeken özelliklerinden bir tanesi, komşuluk hareketlerini hem sürekli hem de ayrık arama uzayında gerçekleştirme kolaylığının olmasıdır. Ayrıca ERA sistematik matematiksel hesaplamaya dayalı olarak kurulduğundan sağlamlığının ve yakınsamasının araştırılması kolaylaşmıştır [47].

Metasezgisel algoritmaların yapısı gereği kesin çözümü garanti etmez. Ama bir çok metasezgisel algoritmaya göre çok fazla parametreye sahip olmasından dolayı problemin çözümü için uygun parametre değerlerini belirlemek zordur. ERA'nın güçlü ve zayıf yönleri Çizelge 3.1'de özetlenmiştir.

Çizelge 3.1. ERA'nın avantaj ve dezavantajları.

Avantajlar	Dezavantajlar
1. İyi yakınsama oranı **2.** Farklı türde optimizasyon sorunları için uyumlu **3.** Diğer algoritmalarla birleştirmek nispeten daha kolaydır **4.** Hem sürekli hem de ayrık arama alanı için geçerlidir **5.** Güçlü komşular arama özelliği **6.** Küçük bir yerel optimuma yakalanma ihtimali **7.** Etkili küresel arama yaklaşımı **8.** Başlıca özellikleri; esneklik, sağlamlık ve ölçeklenebilirlik **9.** İlk çözümlere daha az bağımlılık **10.** Yazarı tarafından verilen temel ERA kodu **11.** Çok fazla parametre üzerinde işlem yapma **12.** Makul hesaplama süresi	**1.** Teorik yakınsaklık özelliği olmaması **2.** Çok fazla parametre ayarı **3.** Erken yakınsama ile sonlanabilme ihtimali **4.** Orijinal algoritma sürekli arama alanı için tasarlanmıştır.

3.4. K-ORTALAMA ALGORİTMASI

K-Ortalama algoritması çok fazla kullanılan ve tanınan kümeleme için kullanılan bir tekniktir. J. MacQueen tarafından 1967 yılında tanımlanmıştır. Algoritma isminde kullanılan k harfi küme adedini belirten sabit sayıyı temsil etmektedir. Bu sayı algoritma çalıştırılmadan girilen ve algoritma sonuçlandırılana kadar değiştirilemeyen bir tamsayıdır. Kümeleme işlemi algoritmada belirtilen k sayısı kadar, küme merkezlerinin birbirine en uzak olduğu, küme elemanlarının birbirine en yakın olduğu gruplara ayırarak yapılır. Kümeleme işleminde denklem (3.14)'de formülü verilen öklit algoritması kullanılır [53].

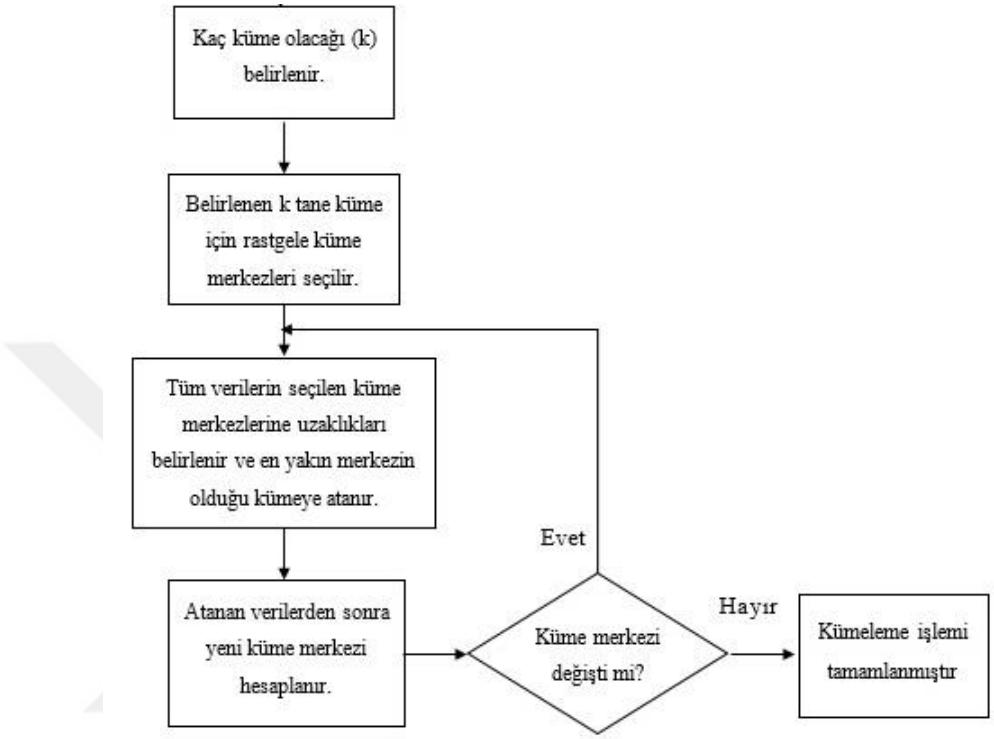
$$d(i,j) = \sqrt{\left(x_{i1} - x_{j1}\right)^2 + \left(x_{i2} - x_{j2}\right)^2 + \cdots + \left(x_{ip} - x_{jp}\right)^2} \tag{3.14}$$

K-Ortalamalar algoritmasının çalışma mantığı aşağıdaki gibidir [53].

- Küme sayısı belirlenir.
- Bir seferlik belirtilen küme sayısı kadar eldeki verilerden küme merkezleri seçilir.
- Verilerin hepsi için belirlenen merkezlere olan uzaklıkları öklit uzunluğuna göre hesaplanır ve merkezlere en yakın olanlar o merkezin kümesi olurlar.

- Kümeler oluşturulduktan sonra tekrar merkez hesaplanır.
- Eğer yeni merkezler aynı ise kümeleme tamamlanır, değilse 3. adımdan itibaren işlemler tekrarlanır.

K-ortalamalar algoritmasının akış diyagramı Şekil 3.9'da gösterilmiştir [54].



Şekil 3.9. K-ortalamalar akış diyagramı.

K-ortalamalar çalışma mantığını anlatmak için Çizelge 3.2'deki veriler üzerinden bir örnekleme yapılmıştır.

Çizelge 3.2. K-ortalamalar veri tablosu.

Veri Numarası	1. Konum	2. Konum
1	1.5	2
2	2	2
3	3	3.5
4	4	4.5
5	7	7
6	8	8
7	7.5	8
8	8	9

Burada ilk adım olarak, küme sayısını 2 olarak seçiyoruz. Bu yüzden de iki küme için küme merkezi olarak 3. ve 4. satırdaki veriler merkez1=(3, 3.5) ve merkez2=(4, 4.5) noktaları belirlenir.

Seçilen küme merkezlerine tüm veri noktalarının uzaklıkları Öklid hesaplaması üzerinden hesaplanır. Çıkan sonuçlar Çizelge 3.3 tablosundaki gibidir.

Çizelge 3.3. Birinci adım sonucu uzaklıklar ve oluşan kümeler.

Veri Noktası	1. Merkeze Uzaklık	2. Merkeze Uzaklık	Küme Numarası
1	2.12	3.53	1
2	1.80	3.20	1
3	0	1.41	1
4	1.41	0	2
5	5.32	3.90	2
6	6.73	5.32	2
7	6.36	4.95	2
8	7.43	6.02	2

Çıkan sonuçlara göre her küme için aritmetik ortalama ile yeni küme merkezleri hesaplanır.

$$merkez1 = \left(\frac{1.5 + 2 + 3}{3}, \frac{2 + 2 + 3.5}{3}\right) = (2.17, 2.50)$$

$$merkez2 = \left(\frac{4 + 7 + 8 + 7.5 + 8}{5}, \frac{4.5 + 7 + 8 + 8 + 9}{5}\right) = (6.90, 7.30)$$

Yeni küme merkezleri $merkez1 = (2.17, 2.50)$ ve $merkez2 = (6.90, 7.30)$ olarak belirlenmiş olup, tüm noktaların yeni merkezlere göre Öklid yöntemi ile uzaklıkları hesaplanır. Çıkan sonuçlar Çizelge 3.4 deki gibidir.

Çizelge 3.4. İkinci adıma göre yeni uzaklıklar ve yeni kümeler.

Veri Noktası	1. Merkeze Uzaklık	2. Merkeze Uzaklık	Küme Numarası
1	0.84	7.22	1
2	0.53	7.22	1
3	1.30	5.45	1
4	2.71	4.03	1
5	6.60	0.32	2
6	8.01	1.30	2
7	7.66	0.92	2
8	8.73	2.02	2

Oluşan ikinci küme yapısına göre şunu görüyoruz birinci adımdaki küme elemanları ile ikinci adımdaki küme elemanları aynı değildir. Bu yüzden de oluşan yeni kümelere göre aritmetik ortalama ile yeni küme merkezleri belirlenir.

$$merkez1 = \left(\frac{1.5 + 2 + 3 + 4}{4}, \frac{2 + 2 + 3.5 + 4.5}{4}\right) = (2.63, 3)$$

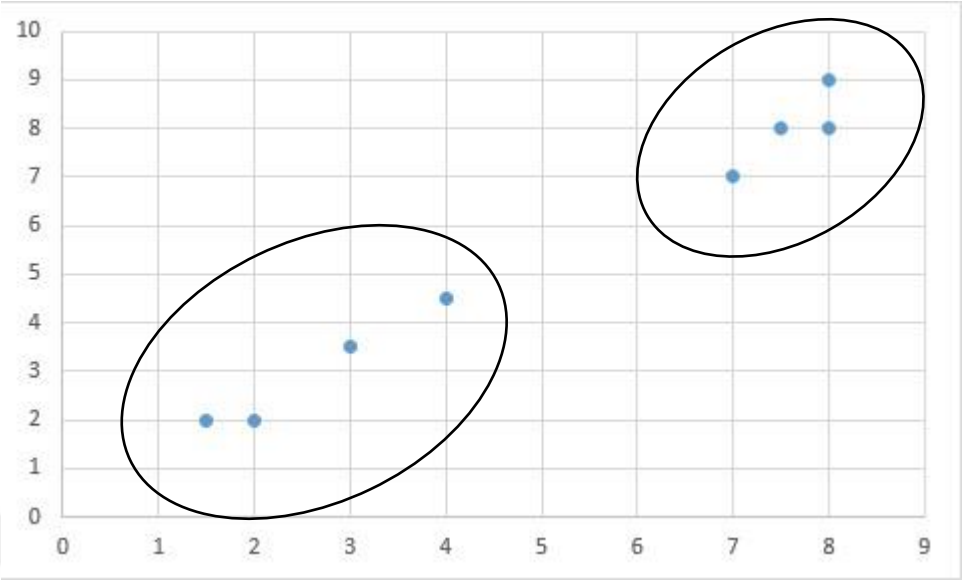
$$merkez2 = \left(\frac{7 + 8 + 7.5 + 8}{4}, \frac{7 + 8 + 8 + 9}{4}\right) = (7.63, 8)$$

Yeni küme merkezleri $merkez1 = (2.63, 3)$ ve $merkez2 = (7.63, 8)$ olarak belirlenmiş olup, tüm noktaların yeni merkezlere göre Öklid yöntemi ile uzaklıkları hesaplanır. Çıkan sonuçlar Çizelge 3.5 deki gibidir.

Çizelge 3.5. Üçüncü adıma göre yeni uzaklıklar ve yeni kümeler.

Veri Noktası	1. Merkeze Uzaklık	2. Merkeze Uzaklık	Küme Numarası
1	1.50	8.24	1
2	1.17	7.87	1
3	0.62	6.12	1
4	2.04	4.71	1
5	5.93	0.81	2
6	7.34	0.62	2
7	6.99	0.52	2
8	8.06	1.55	2

Üçüncü adım sonucunda oluşan kümeler ile ikinci adım sonucunda oluşan kümeler aynı verileri içermektedir. Oluşan kümeler düzlem üzerinde Şekil 3.10'de gösterilmiştir.



Şekil 3.10. Örnek küme oluşumları.

4. UYGULAMA

GSP Türkiye haritası üzerinde ERA kullanılarak daha önce bir çalışma yapılmamış olup, GSP için farklı haritalar üzerinde çalıştırılmış ve 6 farklı algoritma ile de performansı karşılaştırılmıştır. Bu karşılaştırma sonucu, sinir ağı, PSO ve arı kolonisi optimizasyonuna göre iyi sonuçlar elde edilmiştir. Tabu arama, KKA ve BTA ise ERA'ya göre daha iyi sonuçlar vermiştir [55].

Bu çalışmada ise GSP'nin ERA performans değerlendirmesi yapılmıştır. Bu değerlendirmede KKA, PSO ve BTA ile de sonuçlar bulunmuş ve bulunan sonuçlar üzerinden karşılaştırılmıştır.

Ayrıca k-ortalamalar kümeleme analizi ile k parametresine 7, 8 ve 9 değeri verilerek hem kümelemeler arası karşılaştırmalar yapılmış hem de tek tur ile kümelenmiş birden fazla tur arasında çıkan değerler üzerinden değerlendirmeler yapılmıştır.

Bu çalışmada veri kümesi olarak Türkiye'nin 81 il kümesi kullanılmıştır. Bu verilere ait iller arası mesafeler KGM'nin iller arası mesafeleri gösteren web sayfasından [56] ve koordinatlar kişisel web sayfasından alınmıştır [57]. Ayrıca program çıktıları sonucunda bulunan güzergah düzenine göre KGM'nin güzergah analizi sayfasından da mesafe ve yolculuk süresi sonuçları çıkarılmıştır [58]. Uygulamada illerin koordinatlarının bulunduğu ve iller arası mesafeleri tutan iki adet başlangıç matrisi kullanılmıştır. Matrislerdeki bilgilere illerin plaka kodları ile ulaşılmıştır.

Uygulama için kullanılan bilgisayar; İntel Core i5 8400 CPU 2,80 Ghz işlemci, 8 GB DDR4 ram, Nvidia GeForce GTX 1050 Ti ekran kartı, 256 GB SSD disk donanıma ve Windows 10 (64 bit) yazılım sistemine sahiptir. Kodlamalar Matlab R2018b programı üzerinde yazılmıştır.

4.1. GEZGİN SATICI PROBLEMİNİN ALGORİTMALAR İLE ÇALIŞTIRILMASI

Çalışmada ilk olarak her algoritma için ortak döngü sayısı ve popülasyon sayısı kullanılarak sonuçlar bulunmuştur. Bulunan değerler her algoritma 30 kez çalıştırılarak ortaya çıkarılmıştır.

Her algoritma için döngü sayısı 20000 ve popülasyon sayısı 25 alınarak çalıştırılmış ve Çizelge 4.1 deki sonuçlar bulunmuştur. Çıkan sonuçlara bakıldığında en kısa tur BTA tarafından bulunmuştur. Ortalama değerlere bakıldığında da yine en iyi değerler BTA tarafından bulunmuştur. ERA ile bulunan en kısa tur BTA ile bulunan en iyi çözüme yakındır. Sapma açısından değerlendirildiğinde ise ERA ve BTA aynı sapmalara sahiptir. En kısa tur ve en uzun tur arasında sapmanın en az yaşandığı algoritma ise KKA'dır ve daha kararlıdır.

Çizelge 4.1. Ortak parametreler ile bulunan mesafe değerleri.

Algoritma Türü	Döngü Sayısı	Popülasyon Sayısı	En Kısa Mesafe (km)	En Uzun Mesafe (km)	Ortalama Mesafe (km)
PSO	20000	25	10373	11799	10955.00
KKA	20000	25	10345	10630	10540.63
BTA	20000	25	10094	10911	10375.07
ERA	20000	25	10189	11424	10711.10

Bulunan en kısa tur mesafesi dışında algoritmaların bulma süreleri de önemlilik arz ediyor. Özellikle nokta sayısı arttıkça bu süreler katlanarak çoğalmakta. Çizelge 4.2'de algoritmalar tarafından bulunan en kısa ve en uzun tura ait algoritma çalışma süreleri gösterilmiştir. Burada KKA en uzun sürelere sahiptir ve ERA ile karşılaştırıldığında yaklaşık olarak 24 kat, BTA ile karşılaştırıldığında yaklaşık olarak 45 kat daha uzun sürelere sahiptir.

Çizelge 4.2. Ortak parametreler ile bulunan süre değerleri.

Algoritma Türü	Döngü Sayısı	Popülasyon Sayısı	En Kısa Turun Algoritma Çalışma Süresi (sn)	En Uzun Turun Algoritma Çalışma Süresi (sn)	Ortalama Program Çalışma Süresi (sn)
PSO	20000	25	23.21	32.14	27.08
KKA	20000	25	413.82	488.66	434.57
BTA	20000	25	9.14	10.93	9.69
ERA	20000	25	17.27	21.53	19.20

Algoritmalar ile en kısa turu bulurken, her algoritma parametreleri için optimal değerler üzerinden çalışılması önemlidir. Burada optimal parametrelerden kastımız en kısa turu,

en kısa algoritma çalışma süresinde bulmaktır. Çizelge 4.3'deki algoritma optimal parametreleri ile her algoritma 30 kez çalıştırılmış ve tablodaki değerler bulunmuştur. En kısa tur mesafesi BTA ile elde edilmiş olup, ikinci ise ERA ile bulunan en kısa tur mesafesi olmuştur.

Çizelge 4.3. Optimal parametreler ile bulunan mesafe değerleri.

Algoritma Türü	Döngü Sayısı	Popülasyon Sayısı	En Kısa Mesafe (km)	En Uzun Mesafe (km)	Ortalama Mesafe (km)
PSO	20000	25	10373	11799	10955
KKA	2500	10	10297	11019	10700.53
BTA	15000	15	9893	10959	10364.87
ERA	10000	40	9982	11194	10585.97

Ortak parametreler kullanılarak bulunan değerler ile optimal parametre kullanılarak bulunan değerler arasında çok yüksek farklar yoktur. Buradaki asıl farklar algoritmaların çalışma sürelerinde oluşmaktadır. Optimal parametreler ile en kısa turu en kısa algoritma çalışma süresinde bulunması hedeflenmiştir. Çizelge 4.4'de optimal parametreler ile algoritma çalışma sürelerini gösterilmiştir. BTA en kısa tur mesafesini en kısa algoritma çalışma süresi içinde bulmuştur. Diğer algoritmalar ile karşılaştırıldığında aradaki fark kayda değerdir.

Çizelge 4.4. Optimal parametreler ile bulunan süre değerleri.

Algoritma Türü	Döngü Sayısı	Popülasyon Sayısı	En Kısa Turun Algoritma Çalışma Süresi (sn)	En Uzun Turun Algoritma Çalışma Süresi (sn)	Ortalama Program Çalışma Süresi (sn)
PSO	20000	25	23.21	21.14	27.08
KKA	2500	10	26.88	28.39	27.33
BTA	15000	15	3.59	4.91	3.83
ERA	10000	40	13.50	16.53	15.07

4.1.1. Parçacık Sürü Optimizasyonu Sonuçları

PSO için döngü sayısı 20000, popülasyon sayısı 25, atalet ağırlığı 1 değerleri kullanılarak en kısa mesafeye ulaşılmıştır. Döngü sayısının ve popülasyon sayısının azaltılması en

kısa tur için büyük değerler bulmasına sebep olmakta, algoritma çalışma süresini de kısaltmaktadır. Döngü sayısı için 20000'den büyük ve popülasyon sayısı 25'den büyük değerlerde bulunan tur mesafelerinde değişiklik görülmemeye başlanmıştır. Bu veriler üzerinden çıkan sonuçlar Çizelge 4.5 tablosunda gösterilmiştir. Bulunan en kısa mesafeye ait illerin listesi de Çizelge 4.6 tablosunda gösterilmiştir.

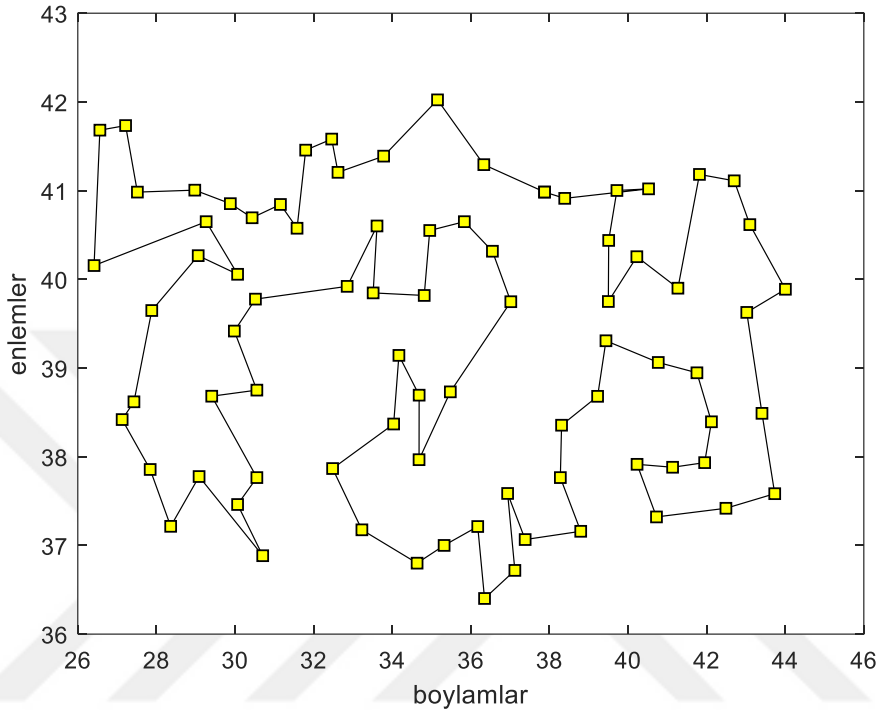
Çizelge 4.5. PSO sonucu bulunan veriler.

Programın Bulduğu En Kısa Mesafe (km)	**Programın Çalışma Süresi (sn)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Bulunan Mesafe (km)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Tahmini Yolculuk Süresi**
10373	23.21	10350	129 saat 7 dakika

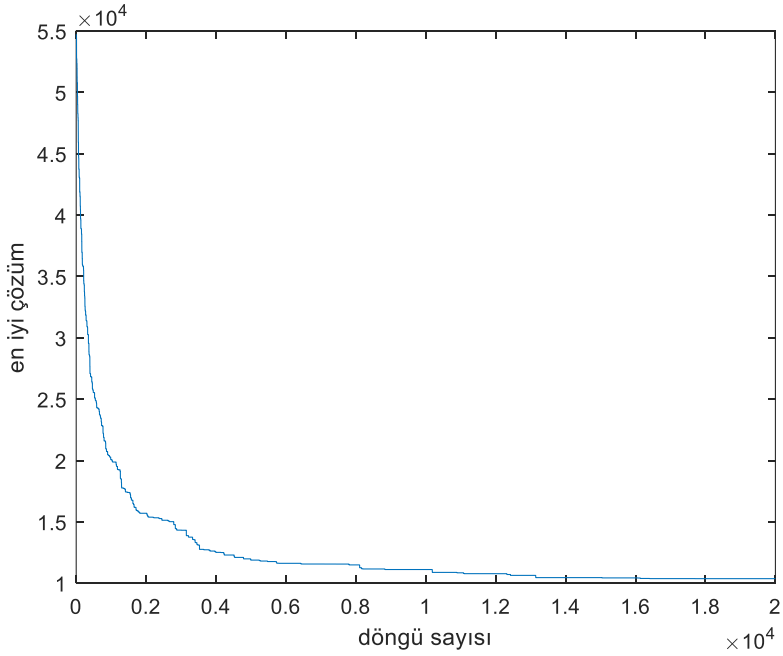
Çizelge 4.6. PSO sonucu bulunan en kısa mesafe il listesi.

1	ORDU	28	ŞANLIURFA	55	ISPARTA
2	GİRESUN	29	GAZİANTEP	56	BURDUR
3	RİZE	30	KAHRAMANMARAŞ	57	ANTALYA
4	TRABZON	31	KİLİS	58	DENİZLİ
5	GÜMÜŞHANE	32	HATAY	59	MUĞLA
6	ERZİNCAN	33	OSMANİYE	60	AYDIN
7	BAYBURT	34	ADANA	61	İZMİR
8	ERZURUM	35	MERSİN	62	MANİSA
9	ARTVİN	36	KARAMAN	63	BALIKESİR
10	ARDAHAN	37	KONYA	64	BURSA
11	KARS	38	AKSARAY	65	BİLECİK
12	IĞDIR	39	KIRŞEHİR	66	YALOVA
13	AĞRI	40	NEVŞEHİR	67	ÇANAKKALE
14	VAN	41	NİĞDE	68	EDİRNE
15	HAKKÂRİ	42	KAYSERİ	69	KIRKLARELİ
16	ŞIRNAK	43	SİVAS	70	TEKİRDAĞ
17	MARDİN	44	TOKAT	71	İSTANBUL
18	DİYARBAKIR	45	AMASYA	72	KOCAELİ
19	BATMAN	46	ÇORUM	73	SAKARYA
20	SİİRT	47	YOZGAT	74	DÜZCE
21	BİTLİS	48	KIRIKKALE	75	BOLU
22	MUŞ	49	ÇANKIRI	76	ZONGULDAK
23	BİNGÖL	50	ANKARA	77	BARTIN
24	TUNCELİ	51	ESKİŞEHİR	78	KARABÜK
25	ELAZIĞ	52	KÜTAHYA	79	KASTAMONU
26	MALATYA	53	AFYONKARAHİSAR	80	SİNOP
27	ADIYAMAN	54	UŞAK	81	SAMSUN

Bulunan en kısa mesafeye ait program çıktısı Şekil 4.1'de, bulunan en kısa tur değeri ile döngü sayısının grafiği Şekil 4.2'de, bulunan en kısa mesafenin harita üzerinde kuş bakışı çizimi Şekil 4.3'de, bulunan en kısa mesafenin karayolları üzerindeki yol haritası Şekil 4.4'de gösterilmiştir.



Şekil 4.1. PSO en kısa mesafe program çıktısı.



Şekil 4.2. PSO en kısa tur-döngü sayısı grafiği.

Şekil 4.3. PSO en kısa mesafe kuş bakışı gösterimi.

Şekil 4.4. PSO en kısa mesafe karayollarında gösterimi.

4.1.2. Karınca Kolonisi Algoritması Sonuçları

KKA için döngü sayısı 2500, popülasyon sayısı 10, alpha 1, beta 1, buharlaşma hızı 0.05 değerleri kullanılarak en kısa mesafeye ulaşılmıştır. Döngü sayısının ve popülasyon sayısının azaltılması en kısa tur için büyük değerler bulmasına sebep olmakta, algoritma çalışma süresini de kısaltmaktadır. Döngü sayısı için 2500'den büyük ve popülasyon sayısı 10'dan büyük değerlerde bulunan tur mesafelerinde değişiklik görülmemeye başlanmıştır. Bu veriler üzerinden çıkan sonuçlar Çizelge 4.7 tablosunda gösterilmiştir. Bulunan en kısa mesafeye ait illerin listesi de Çizelge 4.8 tablosunda gösterilmiştir.

Çizelge 4.7. KKA sonucu bulunan veriler.

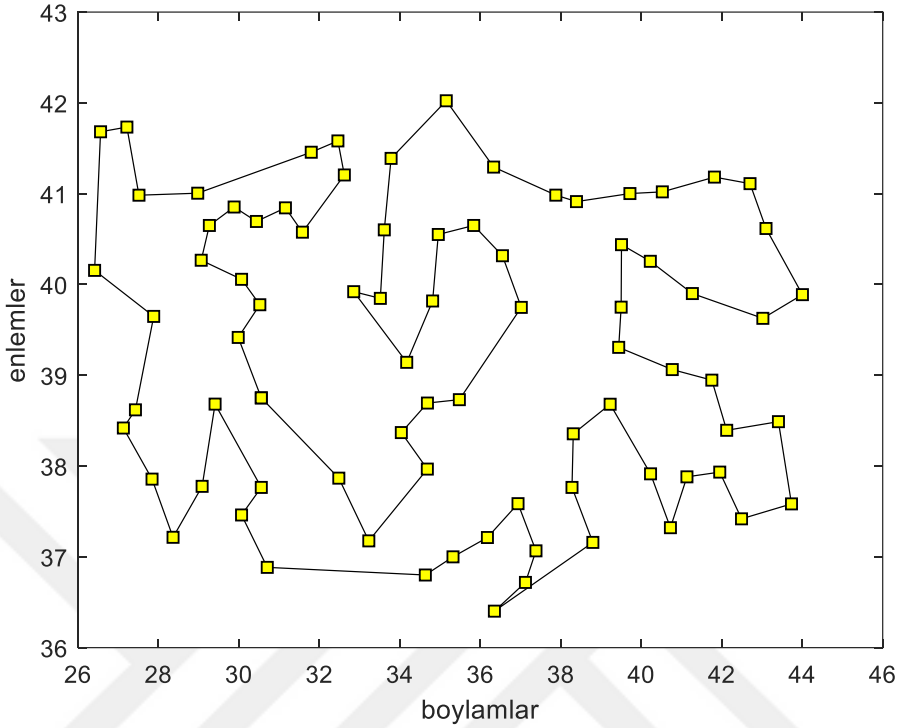
Programın Bulduğu En Kısa Mesafe (km)	**Programın Çalışma Süresi (sn)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Bulunan Mesafe (km)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Tahmini Yolculuk Süresi**
10297	26.88	10338	128 saat 50 dakika

Çizelge 4.8. KKO sonucu bulunan en kısa mesafe il listesi.

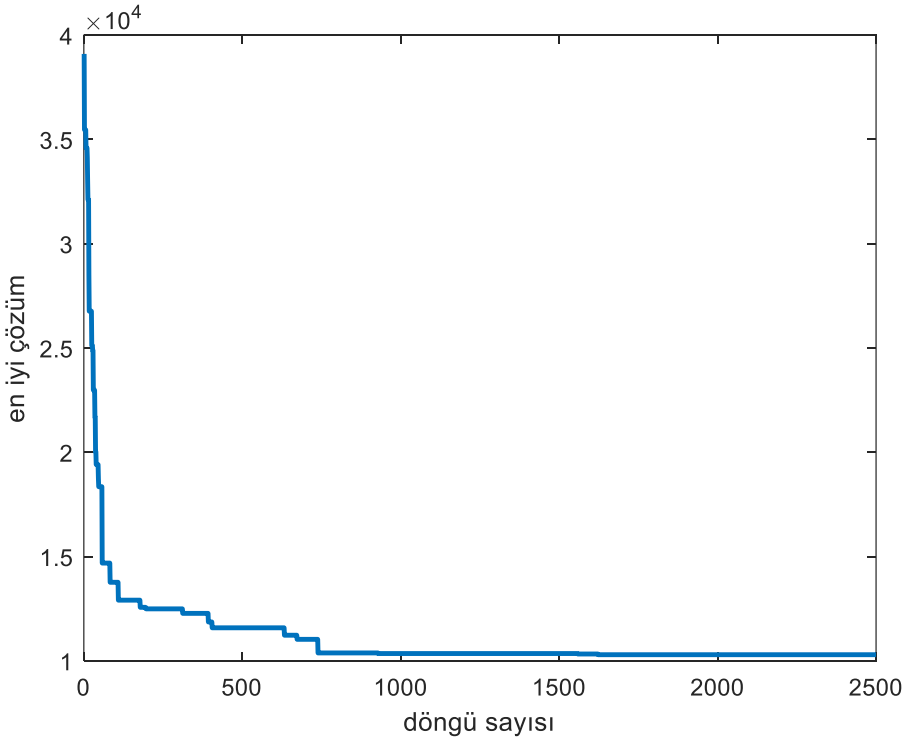
1	AFYONKARAHİSAR	28	ANTALYA	55	AĞRI
2	KÜTAHYA	29	MERSİN	56	IĞDIR
3	ESKİŞEHİR	30	ADANA	57	KARS
4	BİLECİK	31	OSMANİYE	58	ARDAHAN
5	BURSA	32	KAHRAMANMARAŞ	59	ARVİN
6	YALOVA	33	GAZİANTEP	60	RİZE
7	KOCAELİ	34	KİLİS	61	TRABZON
8	SAKARYA	35	HATAY	62	GİRESUN
9	DÜZCE	36	ŞANLIURFA	63	ORDU
10	BOLU	37	ADIYAMAN	64	SAMSUN
11	KARABÜK	38	MALATYA	65	SİNOP
12	BARTIN	39	ELAZIĞ	66	KASTAMONU
13	ZONGULDAK	40	DİYARBAKIR	67	ÇANKIRI
14	İSTANBUL	41	MARDİN	68	KIRIKKALE
15	TEKİRDAĞ	42	BATMAN	69	ANKARA
16	KIRKLARELİ	43	SİİRT	70	KIRŞEHİR
17	EDİRNE	44	ŞIRNAK	71	YOZGAT
18	ÇANAKKALE	45	HAKKÂRİ	72	ÇORUM
19	BALIKESİR	46	VAN	73	AMASYA
20	MANİSA	47	BİTLİS	74	TOKAT
21	İZMİR	48	MUŞ	75	SİVAS
22	AYDIN	49	BİNGÖL	76	KAYSERİ
23	MUĞLA	50	TUNCELİ	77	NEVŞEHİR
24	DENİZLİ	51	ERZİNCAN	78	AKSARAY
25	UŞAK	52	GÜMÜŞHANE	79	NİĞDE
26	ISPARTA	53	BAYBURT	80	KARAMAN
27	BURDUR	54	ERZURUM	81	KONYA

Bulunan en kısa mesafeye ait program çıktısı Şekil 4.5’de, bulunan en kısa tur değeri ile döngü sayısı grafiği Şekil 4.6’da, bulunan en kısa mesafenin harita üzerinde kuş bakışı

çizimi Şekil 4.7'de, bulunan en kısa mesafenin karayolları üzerindeki yol haritası Şekil 4.8'da gösterilmiştir.



Şekil 4.5. KKA en kısa mesafe program çıktısı.



Şekil 4.6. KKA en kısa tur-döngü sayısı grafiği.

Şekil 4.7. KKA en kısa mesafe kuş bakışı gösterimi.

Şekil 4.8. KKA en kısa mesafe karayollarında gösterimi.

4.1.3. Benzetilmiş Tavlama Algoritması Sonuçları

BTA için döngü sayısı 15000, alt yineleme sayısı 15, başlangıç sıcaklığı 0.025, sıcaklık azaltma oranı 0.99 değerleri kullanılarak en kısa mesafeye ulaşılmıştır. Döngü sayısının ve alt yineleme sayısının azaltılması en kısa tur için büyük değerler bulmasına sebep olmakta, algoritma çalışma süresini de kısaltmaktadır. Döngü sayısı için 15000'den büyük ve alt yineleme sayısı 15'den büyük değerlerde bulunan tur mesafelerinde değişiklik görülmemeye başlanmıştır. Bu veriler üzerinden çıkan sonuçlar Çizelge 4.9 tablosunda gösterilmiştir. Bulunan en kısa mesafeye ait illerin listesi de Çizelge 4.10 tablosunda gösterilmiştir.

Çizelge 4.9. BTA sonucu bulunan veriler.

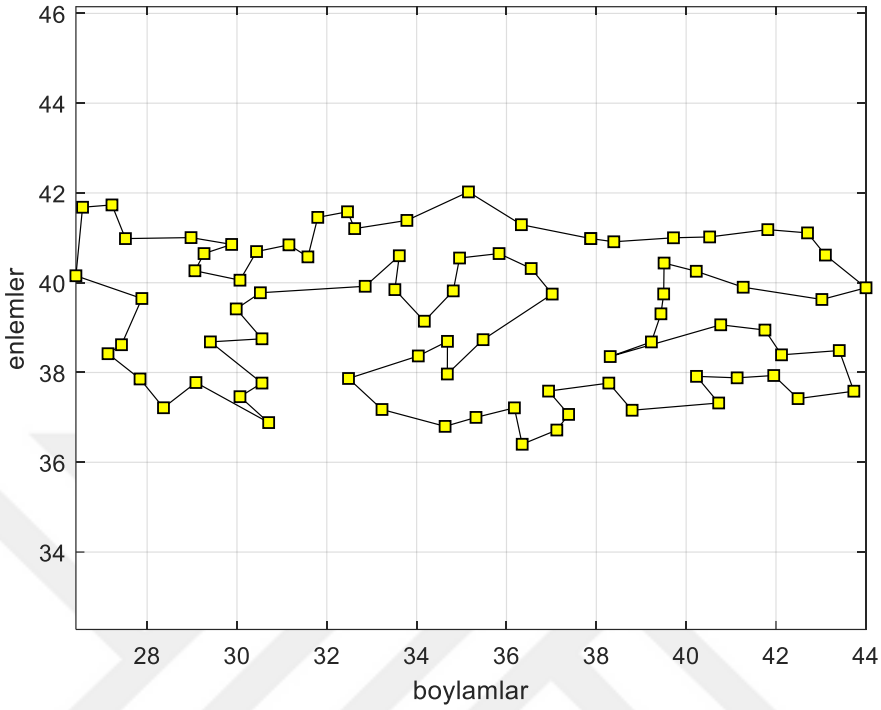
Programın Bulduğu En Kısa Mesafe (km)	**Programın Çalışma Süresi (sn)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Bulunan Mesafe (km)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Tahmini Yolculuk Süresi**
9893	3.59	9959	124 saat 14 dakika

Çizelge 4.10. BTA sonucu bulunan en kısa mesafe il listesi.

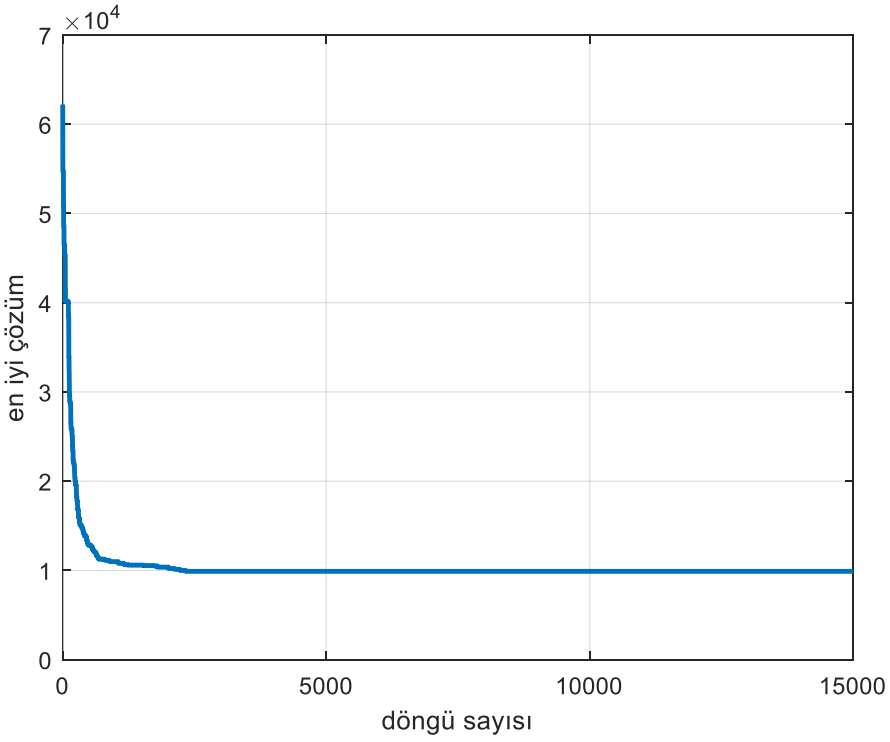
1	ORDU	28	ADIYAMAN	55	ISPARTA
2	GİRESUN	29	KAHRAMANMARAŞ	56	BURDUR
3	TRABZON	30	GAZİANTEP	57	ANTALYA
4	RİZE	31	KİLİS	58	DENİZLİ
5	ARTVİN	32	HATAY	59	MUĞLA
6	ARDAHAN	33	OSMANİYE	60	AYDIN
7	KARS	34	ADANA	61	İZMİR
8	IĞDIR	35	MERSİN	62	MANİSA
9	AĞRI	36	KARAMAN	63	BALIKESİR
10	ERZURUM	37	KONYA	64	ÇANAKKALE
11	BAYBURT	38	AKSARAY	65	EDİRNE
12	GÜMÜŞHANE	39	NEVŞEHİR	66	KIRKLARELİ
13	ERZİNCAN	40	NİĞDE	67	TEKİRDAĞ
14	TUNCELİ	41	KAYSERİ	68	İSTANBUL
15	ELAZIĞ	42	SİVAS	69	KOCAELİ
16	MALATYA	43	TOKAT	70	YALOVA
17	BİNGÖL	44	AMASYA	71	BURSA
18	MUŞ	45	ÇORUM	72	BİLECİK
19	BİTLİS	46	YOZGAT	73	SAKARYA
20	VAN	47	KIRŞEHİR	74	DÜZCE
21	HAKKÂRİ	48	KIRIKKALE	75	BOLU
22	ŞIRNAK	49	ÇANKIRI	76	ZONGULDAK
23	SİİRT	50	ANKARA	77	BARTIN
24	BATMAN	51	ESKİŞEHİR	78	KARABÜK
25	DİYARBAKIR	52	KÜTAHYA	79	KASTAMONU
26	MARDİN	53	AFYONKARAHİSAR	80	SİNOP
27	ŞANLIURFA	54	UŞAK	81	SAMSUN

Bulunan en kısa mesafeye ait program çıktısı Şekil 4.9’de, bulunan en kısa tur değeri ile döngü sayısının grafiği Şekil 4.10’da, bulunan en kısa mesafenin harita üzerinde kuş

bakışı çizimi Şekil 4.11'de, bulunan en kısa mesafenin karayolları üzerindeki yol haritası Şekil 4.12'de gösterilmiştir.



Şekil 4.9. BTA en kısa mesafe program çıktısı.



Şekil 4.10. BTA en iyi tur-döngü sayısı grafiği.

Şekil 4.11. BTA en kısa mesafe kuş bakışı gösterimi.

Şekil 4.12. BTA en kısa mesafe karayollarında gösterimi.

4.1.4. Emperyalist Rekabetçi Algoritması Sonuçları

ERA için döngü sayısı 10000, popülasyon sayısı 40, imparatorluk sayısı 10, asimilasyon katsayısı 2, kolonilerin ortalama maliyet katsayısı 0,1 değerleri kullanılarak en kısa mesafeye ulaşılmıştır. Döngü sayısının ve popülasyon sayısının azaltılması en kısa tur için büyük değerler bulmasına sebep olmakta, algoritma çalışma süresini de kısaltmaktadır. Döngü sayısı için 10000'den büyük ve popülasyon sayısı 40'dan büyük değerlerde bulunan tur mesafelerinde değişiklik görülmemeye başlanmıştır. Bu veriler üzerinden çıkan sonuçlar Çizelge 4.11 tablosunda gösterilmiştir. Bulunan en kısa mesafeye ait illerin listesi de Çizelge 4.12 tablosunda gösterilmiştir.

Çizelge 4.11. ERA sonucu bulunan veriler.

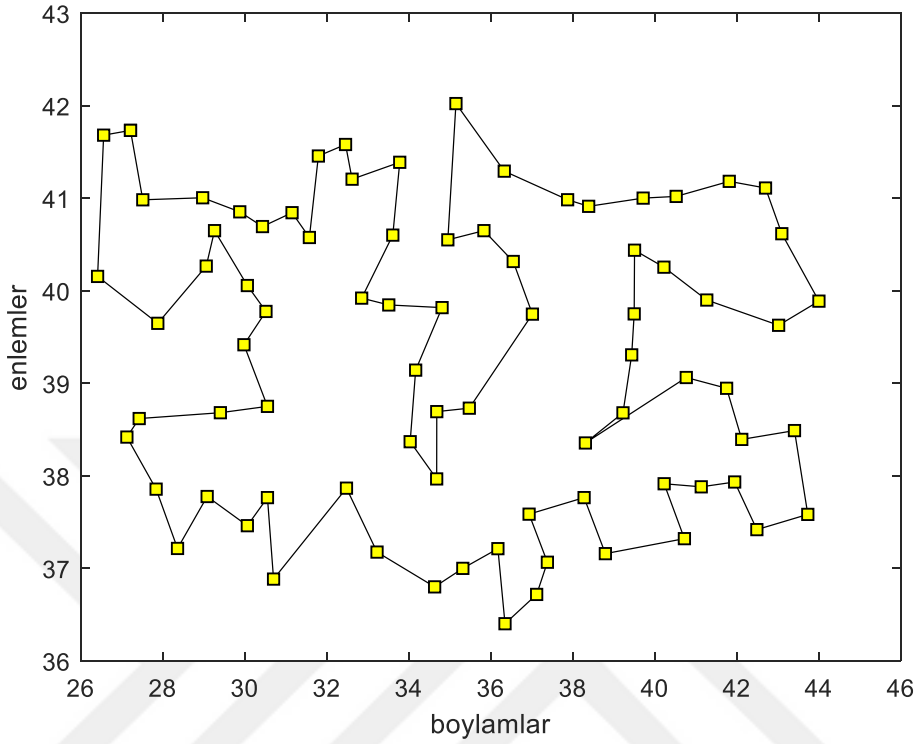
Programın Bulduğu En Kısa Mesafe (km)	**Programın Çalışma Süresi (sn)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Bulunan Mesafe (km)**	**En Kısa Yolun KGM Yol Güzergahı Üzerinde Tahmini Yolculuk Süresi**
9982	13.50	10041	125 saat 26 dakika

Çizelge 4.12. ERA sonucu bulunan en kısa mesafe il listesi.

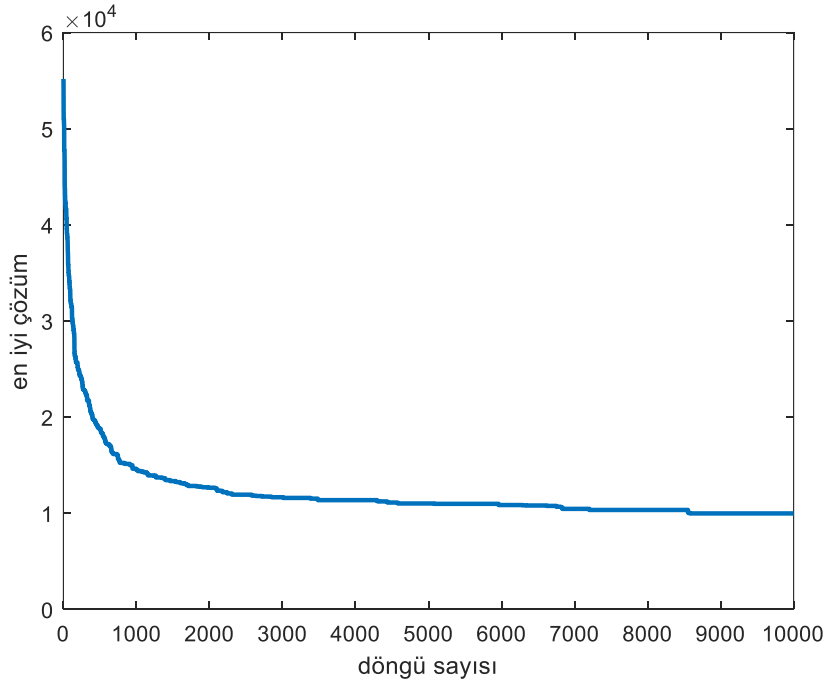
1	KIRKLARELİ	28	GİRESUN	55	KAHRAMANMARAŞ
2	TEKİRDAĞ	29	TRABZON	56	GAZİANTEP
3	İSTANBUL	30	RİZE	57	KİLİS
4	KOCAELİ	31	ARTVİN	58	HATAY
5	SAKARYA	32	ARDAHAN	59	OSMANİYE
6	DÜZCE	33	KARS	60	ADANA
7	BOLU	34	IĞDIR	61	MERSİN
8	ZONGULDAK	35	AĞRI	62	KARAMAN
9	BARTIN	36	ERZURUM	63	KONYA
10	KARABÜK	37	BAYBURT	64	ANTALYA
11	KASTAMONU	38	GÜMÜŞHANE	65	ISPARTA
12	ÇANKIRI	39	ERZİNCAN	66	BURDUR
13	ANKARA	40	TUNCELİ	67	DENİZLİ
14	KIRIKKALE	41	ELAZIĞ	68	MUĞLA
15	YOZGAT	42	MALATYA	69	AYDIN
16	KIRŞEHİR	43	BİNGÖL	70	İZMİR
17	AKSARAY	44	MUŞ	71	MANİSA
18	NİĞDE	45	BİTLİS	72	UŞAK
19	NEVŞEHİR	46	VAN	73	AFYONKARAHİSAR
20	KAYSERİ	47	HAKKÂRİ	74	KÜTAHYA
21	SİVAS	48	ŞIRNAK	75	ESKİŞEHİR
22	TOKAT	49	SİİRT	76	BİLECİK
23	AMASYA	50	BATMAN	77	YALOVA
24	ÇORUM	51	DİYARBAKIR	78	BURSA
25	SİNOP	52	MARDİN	79	BALIKESİR
26	SAMSUN	53	ŞANLIURFA	80	ÇANAKKALE
27	ORDU	54	ADIYAMAN	81	EDİRNE

Bulunan en kısa mesafeye ait program çıktısı Şekil 4.13’da, bulunan en kısa tur değeri ile döngü sayısı grafiği Şekil 4.14’de, bulunan en kısa mesafenin harita üzerinde kuş bakışı

çizimi Şekil 4.15'de, bulunan en kısa mesafenin karayolları üzerindeki yol haritası Şekil 4.16'de gösterilmiştir.



Şekil 4.13. ERA en kısa mesafe program çıktısı.



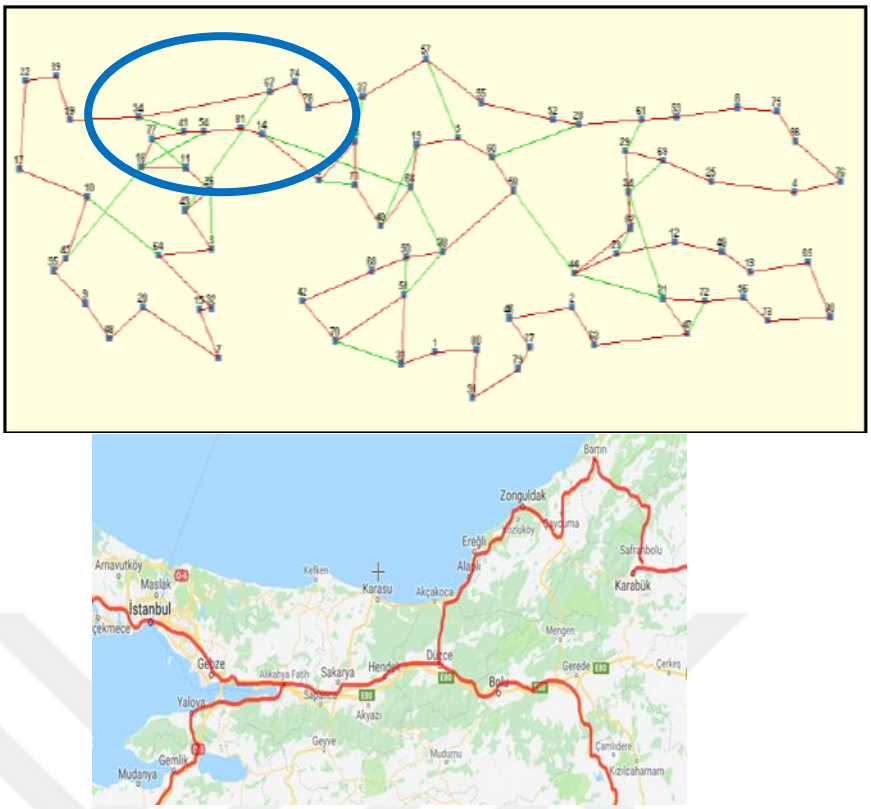
Şekil 4.14. ERA en iyi tur-döngü sayısı grafiği.

Şekil 4.15. ERA en kısa mesafe kuş bakışı gösterimi.

Şekil 4.16. ERA en kısa mesafe karayollarında gösterimi.

4.1.5. Sonuçların Karşılaştırılması

Çalışmada daha önce Türkiye haritası üzerinde yapılan GSP analizi sonucu bulunan 9966 km değerinden daha iyi bir sonuç bulunmuş hem de GSP'nin her şehre sadece bir defa uğrama şartının sağlanmadığı görülmüştür [59]. Şekil 4.17'deki program çıktısından Zonguldak şehrinden İstanbul şehrine giderken başka bir şehre uğramadığı fakat bu çıktının karayolları üzerinde çizildiğinde Düzce, Sakarya ve Kocaeil şehirlerine uğramadan bu güzergahı sağlayamayacağı görülmüştür [60].



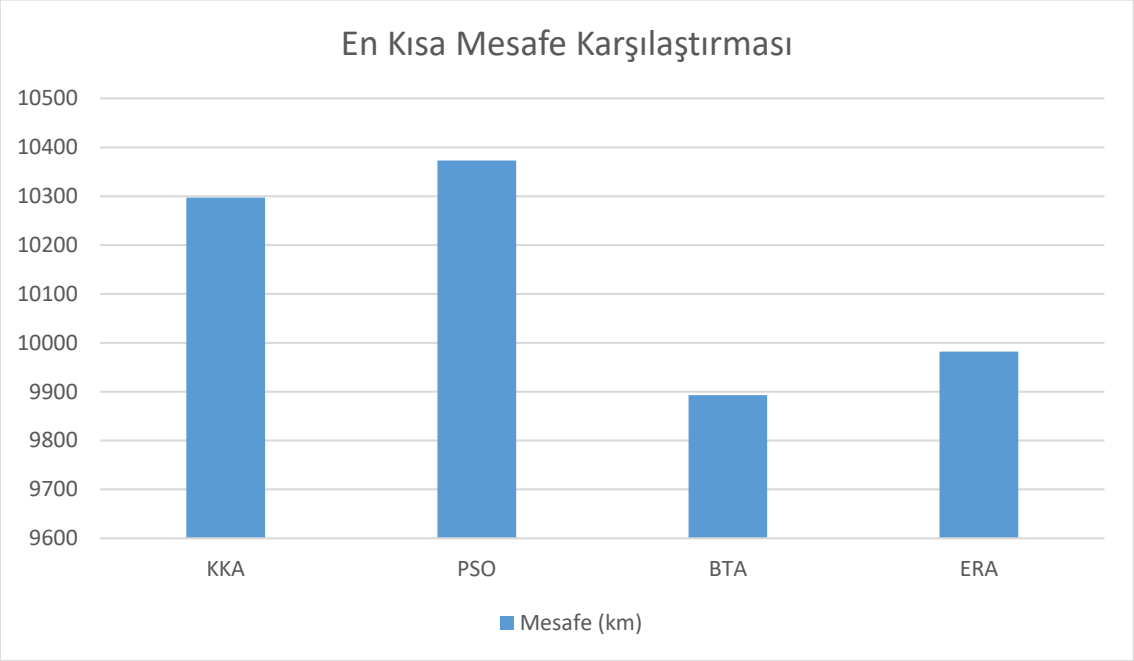
Şekil 4.17. Karşılaştırılan sonucun hatası.

Algoritmaların sonuçları üzerinden bakıldığında Türkiye'deki 81 il üzerinden yapılan GSP çözümlemesi için ERA, KKA ve PSO'ya göre hem en kısa tur mesafesi açısından hem de algoritma çalışma süresi açısından iyi değerler elde etmiştir. BTA ise ERA'ya göre daha güzel sonuçlar çıkarmıştır. BTA'nın özellikle algoritma çalışma süresi diğer üç algoritmaya göre kayda değer sonuçlar çıkarmıştır. Program çalışma süresi özellikle fazla noktanın olduğu veri topluluğunda önem arz etmektedir. Bulunan değerler Çizelge 4.13'de gösterilmiştir.

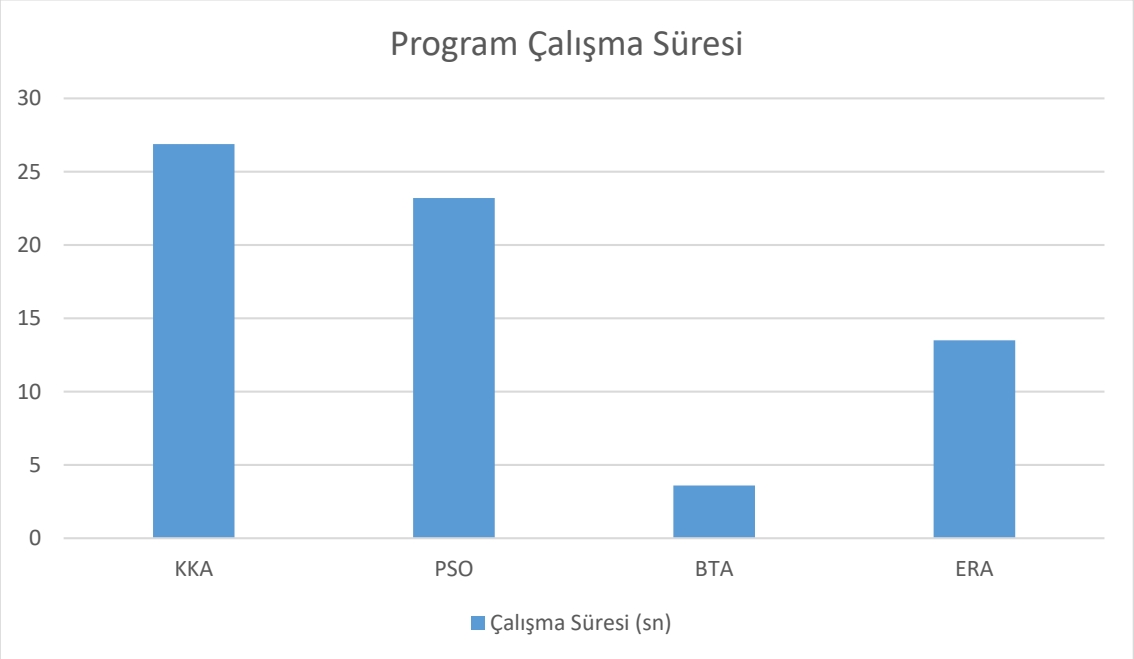
Çizelge 4.13. PSO, KKO, BTA ve ERA sonuçlarının karşılaştırılması.

Kullanılan Algoritma	Döngü Sayısı	En Kısa Mesafe (km)	Programın Çalışma Süresi (sn)
KKA	2500	10297	26.88
PSO	20000	10373	23.21
BTA	15000	9893	3.59
ERA	10000	9982	13.50

Algoritmaların en kısa mesafe ve çalışma sürelerinin birbirleri ile karşılaştırması ayrıca Şekil 4.18 ve Şekil 4.19'da gösterilmiştir.



Şekil 4.18. Algoritmaların en kısa mesafe karşılaştırması.



Şekil 4.19. Algoritmaların çalışma süresi karşılaştırması.

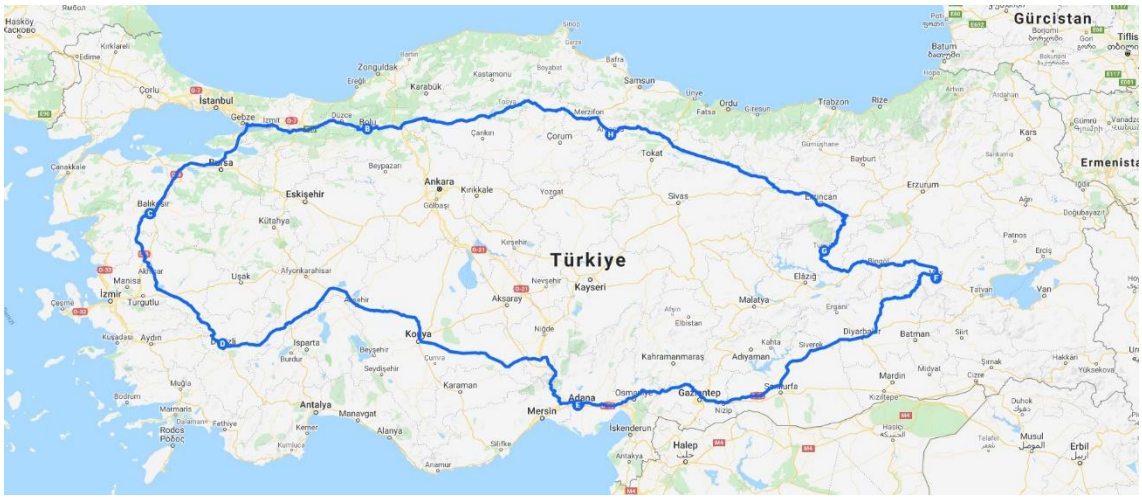
4.2. K-ORTALAMALAR KÜMELEME ANALİZİNİN KULLANILMASI

Gelişen dünyada zamanın ne kadar önemli olduğu asla reddedilemez. Bu yüzden de bütün haritayı bir satıcının gezmesi gerçekçi bir çözüm değildir. GSP problemi lojistik yönden çağımızın önem arz eden konularından birdir. Tek bir gezgin satıcı yerine gezilecek iller k adet kümeye bölünerek her kümeye ait gezgin satıcı ve kümelere ulaşım sağlayacak gezgin satıcı olmak üzere k+1 satıcı ile çözümü ulaşmak daha az maliyetli ve özellikle zaman açısından daha avantajlı olacaktır. Formüllerle duruma bakıldığında n şehirli bir GSP probleminin çözüm kümesi $(n-1)!$ tur içerirken, k-ortalamalar algoritması ile k sayıda alt kümelere ayırır ve küme eleman sayılarını eşit varsaydığımızda $k\left(\left(\left(\frac{n}{k}\right)-1\right)!\right)$ tur içerir. Örneklediğimizde, 81 il için hesaplama yaparsak tek GSP için $7{,}1569 * 10^{118}$ tur içerirken, $\mathrm{k}=9$ için her kümesin 9 elemanı olduğunu varsayarsak $9\left(\left(\left(\frac{81}{9}\right)-1\right)!\right)=362880$ tur içerir.

Bir satıcının tüm noktaları tek başına ziyaret etmesinin yerine, haritayı bölgelere ayırıp bölge merkezlerini bir kişinin ve bölge turlarını da bölge sayısı kadar kişinin ziyaret etmesi özellikle süre açısından avantaj sağlayacaktır.

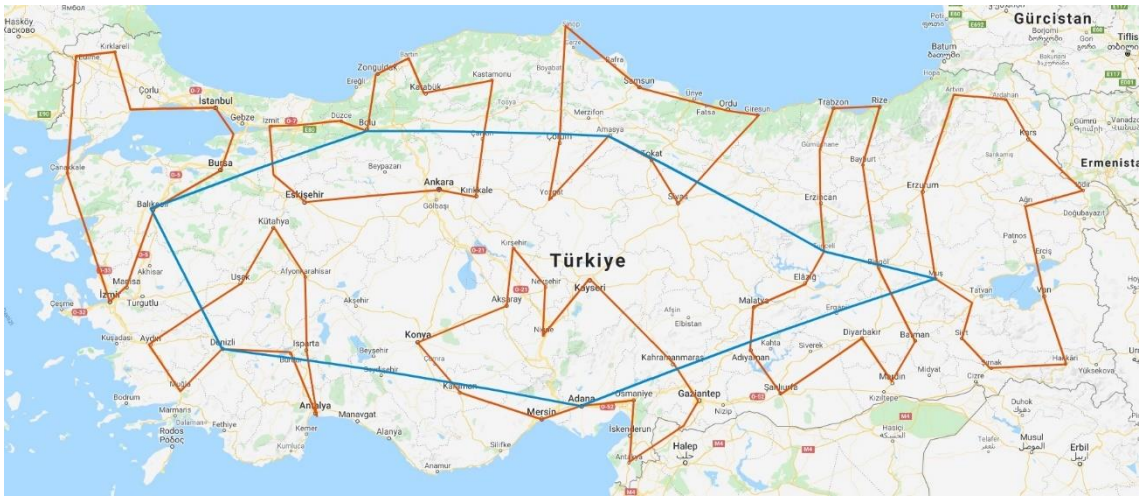
Bu amaçla tezin bu kısmında 81 il k-ortalamalar algoritması kullanılarak 7, 8 ve 9 kümeye ayrılmıştır. Kümeleme işlemi yapılırken her k değeri için 5 kez çalıştırılarak merkez koordinat bilgileri hesaplanmış ve en kısa tura ait bilgiler seçilmiştir. Seçilen koordinat bilgilerine en yakın koordinata sahip il küme merkezi olarak belirlenmiştir. Bu merkezlere göre de KGM sayfasından kilometre ve yolculuk süreleri bulunarak karşılaştırmalar yapılmıştır.

K-ortalamalar algoritması $k=7$ için çalıştırılmış, küme merkezleri ve o kümeye ait noktalar bulunmuştur. Bulunan küme merkezleri tur hesaplamaları ve her kümeye ait noktalar arası turlar ERA ile bulunmuştur. Bulunan küme merkezlerine ait tur harita üzerinde Şekil 4.20'de, oluşan kümelere ait en kısa turlar harita üzerinde Şekil 4.21'da, hem küme merkezlerinin hem de küme iç turlarının kuş bakışı çizimi haritada Şekil 4.22'de gösterilmiştir.



Şekil 4.20. k=7 için küme merkezlerine ait turun harita gösterimi.

Şekil 4.21. k=7 için kümelerin iç turlarının harita gösterimi.



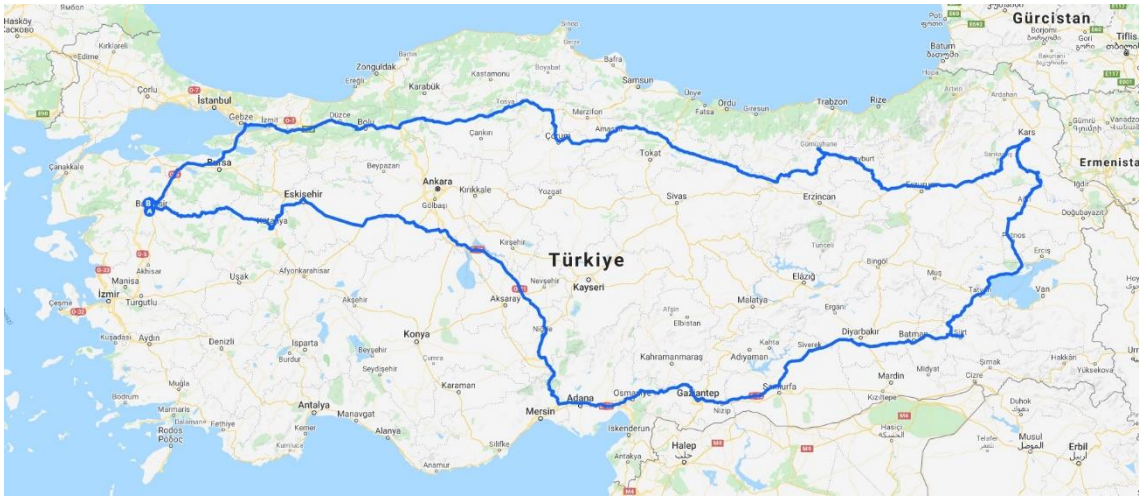
Şekil 4.22. k=7 için küme merkezleri ve iç turların kuşbakışı gösterimi.

Kümeleme işlemi sonucunda bulunan küme merkezleri ve küme içindeki iller arasında en kısa tura ait bilgiler KGM'ye ait yol güzergahı oluşturma sayfasına girilerek merkezler arası ve kümelerin iç turlarına ait mesafe ve yolculuk süresi bilgileri bulunmuştur. Bulunan bilgiler Çizelge 4.14'da gösterilmiştir.

Çizelge 4.14. k=7 için bulunan mesafe ve süreler.

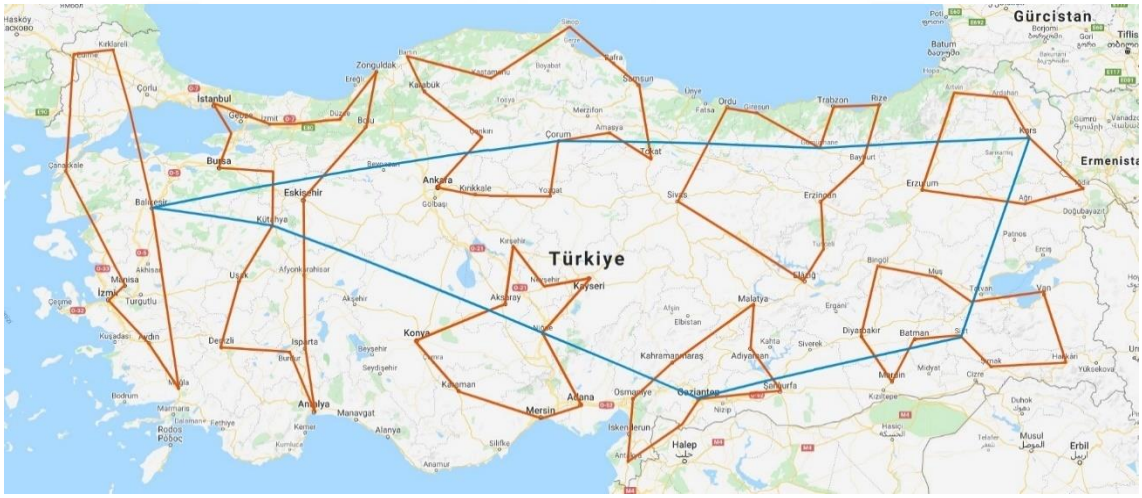
Değerler	**1. Küme**	**2. Küme**	**3. Küme**	**4. Küme**	**5. Küme**	**6. Küme**	**7. Küme**	**Kümeler Toplamı**	**Küme Merkezleri**	**Genel Toplam**
Mesafe (km)	1540	1234	2052	1400	1784	1371	1765	11146	3353	14499
Yolculuk Süresi	20 saat 39 dakika	14 saat 42 dakika	28 saat 33 dakika	15 saat 56 dakika	25 saat 18 dakika	16 saat 34 dakika	21 saat 50 dakika	38 saat 18 dakika	143 saat 32 dakika	181 saat 50 dakika

K-ortalamalar algoritması k=8 için çalıştırılmış, küme merkezleri ve o kümeye ait noktalar bulunmuştur. Bulunan küme merkezleri tur hesaplamaları ve her kümeye ait noktalar arası turlar ERA ile bulunmuştur. Bulunan küme merkezlerine ait tur harita üzerinde Şekil 4.23'de oluşan kümelere ait en kısa turlar harita üzerinde Şekil 4.24'da, hem küme merkezlerinin hem de küme iç turlarının kuş bakışı çizimi haritada Şekil 4.25'de gösterilmiştir.



Şekil 4.23. k=8 için küme merkezlerine ait turun harita gösterimi.

Şekil 4.24. k=8 için kümelerin iç turlarının harita gösterimi.



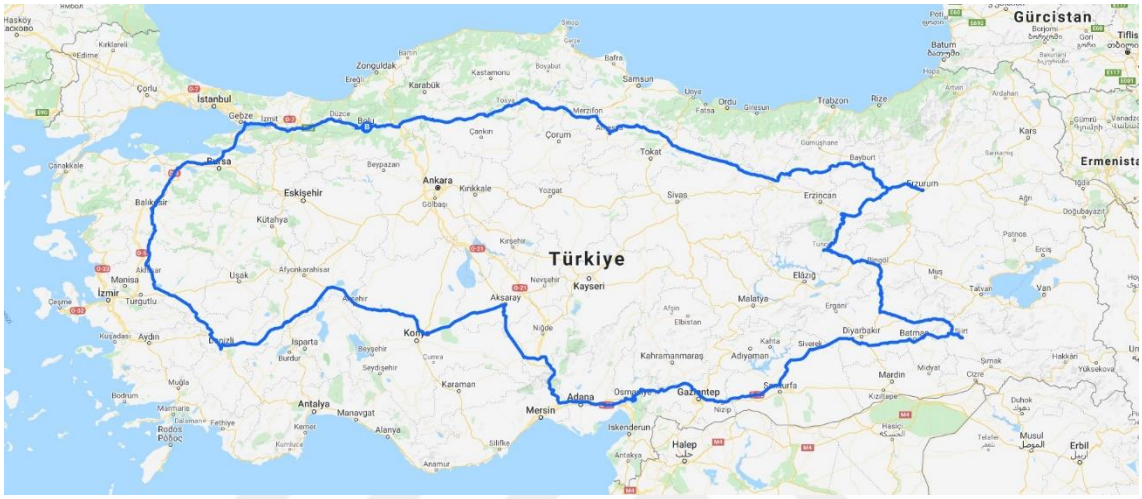
Şekil 4.25. k=8 için küme merkezleri ve iç turların kuşbakışı gösterimi.

Kümeleme işlemi sonucunda bulunan küme merkezleri ve küme içindeki iller arasında en kısa tura ait bilgiler KGM'ye ait yol güzergahı oluşturma sayfasına girilerek merkezler arası ve kümelerin iç turlarına ait mesafe ve yolculuk süresi bilgileri bulunmuştur. Bulunan bilgiler Çizelge 4.15'de gösterilmiştir.

Çizelge 4.15. k=8 için bulunan mesafe ve süreler.

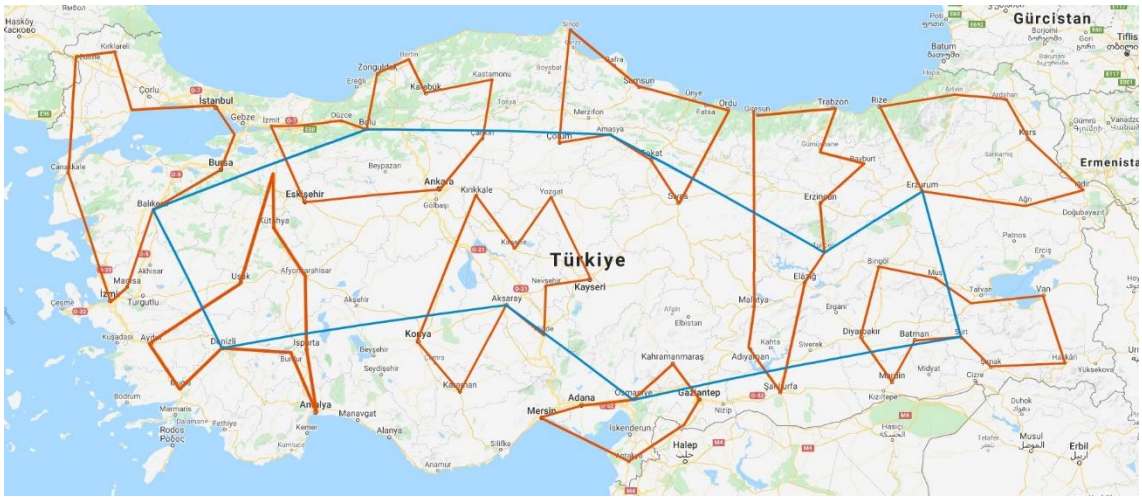
Değerler	**1. Küme**	**2. Küme**	**3. Küme**	**4. Küme**	**5. Küme**	**6. Küme**	**7. Küme**	**8. Küme**	**Kümeler Toplamı**	**Küme Merkezleri**	**Genel Toplam**
Mesafe (km)	863	2226	1531	1816	1316	1673	1187	1035	11647	3966	15613
Yolculuk Süresi	11 saat 42 dakika	25 saat 46 dakika	22 saat 35 dakika	19 saat 42 dakika	18 saat 7 dakika	20 saat 47 dakika	14 saat 10 dakika	13 saat 15 dakika	148 saat 4 dakika	45 saat 6 dakika	193 saat 10 dakika

K-ortalamalar algoritması k=9 için çalıştırılmış, küme merkezleri ve o kümeye ait noktalar bulunmuştur. Bulunan küme merkezleri tur hesaplamaları ve her kümeye ait noktalar arası turlar ERA ile bulunmuştur. Bulunan küme merkezlerine ait tur harita üzerinde Şekil 4.26'de, oluşan kümelere ait en kısa turlar harita üzerinde Şekil 4.27'de, hem küme merkezlerinin hem de küme iç turlarının kuş bakışı çizimi haritada Şekil 4.28'de gösterilmiştir.



Şekil 4.26. k=9 için küme merkezlerine ait turun harita gösterimi.

Şekil 4.27. k=9 için kümelerin iç turlarının harita gösterimi.



Şekil 4.28. k=9 için küme merkezleri ve iç turların kuşbakışı gösterimi.

Kümeleme işlemi sonucunda bulunan küme merkezleri ve küme içindeki iller arasında en kısa tura ait bilgiler KGM'ye ait yol güzergahı oluşturma sayfasına girilerek merkezler arası ve kümelerin iç turlarına ait mesafe ve yolculuk süresi bilgileri bulunmuştur. Bulunan bilgiler Çizelge 4.16'de gösterilmiştir.

Çizelge 4.16. k=9 için bulunan mesafe ve süreler.

Değerler	**1. Küme**	**2. Küme**	**3. Küme**	**4. Küme**	**5. Küme**	**6. Küme**
Mesafe (km)	1227	1281	1066	1504	1400	1316
Yolculuk Süresi	14 saat 49 dakika	15 saat 15 dakika	14 saat 27 dakika	17 saat 43 dakika	15 saat 56 dakika	18 saat 7 dakika
Değerler	**7. Küme**	**8. Küme**	**9. Küme**	**Kümeler Toplamı**	**Küme Merkezleri**	**Genel Toplam**
Mesafe (km)	1179	828	1712	11513	3861	15374
Yolculuk Süresi	15 saat 16 dakika	10 saat 8 dakika	24 saat 17 dakika	147 saat 58 dakika	44 saat 7 dakika	192 saat 5 dakika

K-ortalamalar algoritması ile 7, 8 ve 9 kümeleme işlemi yaparak illeri grupladığımızda çıkan total değerlerin karşılaştırmasını yapacak olursak, bulunan değerler birbirinden çok uzak olmasa da 7 bölge ideal seçim olarak görünmektedir. Çıkan mesafe sonuçları Çizelge 4.17'de, yolculuk süreleri Çizelge 4.18'de gösterilmiştir.

Çizelge 4.17. K-ortalamalar ile bulunan mesafelerin karşılaştırılması.

K Değerleri	Küme Merkezleri	Küme Toplamaları	Genel Toplam
7	3353	11146	14499
8	3966	11647	15613
9	3861	11513	15374

Çizelge 4.18. K-ortalamalar ile bulunan yolculuk sürelerinin karşılaştırılması.

K Değerleri	Küme Merkezleri	Küme Toplamaları	Genel Toplam
7	38 saat 18 dakika	143 saat 32 dakika	181 saat 50 dakika
8	45 saat 6 dakika	148 saat 4 dakika	193 saat 10 dakika
9	44 saat 7 dakika	147 saat 58 dakika	192 saat 5 dakika

5. SONUÇ VE ÖNERİLER

Bu çalışmada GSP'nin çözümü için yeni bulunmuş bir algoritma olan ERA ile performansı ölçülmeye çalışılmış ayrıca PSO, KKO ve BTA ile karşılaştırmalar yapılmıştır. Aynı zamanda GSP'nin güncel hayatta daha kullanışlı olabilmesi için k-ortalamalar algoritması ile değişik adetlerde kümelemeler yaparak daha uygunalabilir sonuçlar elde edilmeye çalışılmıştır.

GSP'nin çözümü için Türkiye haritası üzerinde 81 ile ait koordinatlar ve bu illerin arasındaki gerçek mesafe değerleri kullanılmıştır. Bu bilgiler üzerinden ERA, PSO, KKO ve BTA 30 kez çalıştırılarak en iyi, en kötü ve ortalama değerler hem mesafe olarak hem de program çalışma süresi olarak ortaya çıkarılmıştır.

Bu dört algoritma ile yapılan deneyler sonunda bulunan en iyi çözümlere ait mesafe değerlerinin birbirine çok yakın olduğu görülmüştür. Program çalışma süreleri açısından ERA ile bulunan değerlere bakıldığında, PSO ve KKA ile bulunan sonuçların yarısı kadarken, BTA ile bulunan sonucun 3.76 katıdır.

Bu çalışma 81 il üzerinde yapıldığından GSP için nokta sayısı açısından küçük boyutlu bir problemdir. O sebeple, bulunan sonuçların gerçek karayolları haritası üzerine yansıtıldığında da GSP şartları üzerinden uygulanabilir olması da değerlendirilmiştir. Bu durum PSO ve KKA ile bulunan sonuçlarda olumsuz olarak gözlemlenmiştir. Detaylı açıklaması yapılacak olursa, GSP'nin her şehre sadece bir defa uğrama şartı program çıktısı alındığında sağlanırken, karayolları haritasına yansıtıldığında bir şehrin üzerinden iki defa geçildiği gözlemlenmiştir. PSO ile bulunan en iyi çözümde Giresun şehrinden Rize şehrine geçerken Trabzon şehri üzerinden geçmiş tekrar Trabzon şehrine dönmüştür. KKA ile bulunan en iyi çözümde ise, aynı sorun Zonguldak şehrinden İstanbul şehrine geçerken daha önce uğradığı Düzce, Sakarya ve Kocaeli şehirleri üzerinden geçmesi ile görülmüştür. Bu sorunlar ERA ve BTA ile bulunan en iyi çözümde görülmemiştir.

GSP için k-ortalamalar analizini 7, 8 ve 9 kümeleme yaparak 81 il için uygulandığında 7 kümelemenin ideal kümeleme olduğu görülmüştür. Bunun yanında küme sayısının arttıkça kümelerin kendi iç turlarında kat edilen mesafe ve yolculuk süresinde azalma görülmüştür. Buda özellikle devamlılık arz eden bir çalışma alanında büyük bir avantaj sağlayacaktır.

Kümeleme işlemleri ile bulunan sonuçlar da gerçek hayata uyarlanabilirlik açısından KGM güzergâh analizi sayfası üzerinde yolculuk süreleri bazında da değerlendirilmiştir. Yolculuk süreleri özellikle kümeleme açısından değerlendirildiğinde önem arz etmektedir. Çünkü kümeleme de her kümenin iç turu aynı anda başlayacağından yolculuk süreleri karşılaştırması toplanarak değerlendirilmemelidir.

Bir kargo şirketi üzerinden örneklendiğinde daha net görünecektir. GSP'nin ERA ile tek dolaşımlı çözümü için KGM sayfasından yolculuk süresi hesaplandığında 125 saat 26 dakika görünmektedir. Buda bir kargo arabasının her ile uğrayarak bir kargo bırakması olarak değerlendirildiğinde tüm kargoların yerine ulaşması için net süredir. Aynı işlemi k-ortalamalar analizi ile yaptığımızda ise merkezler arası turun süresi ile, tüm küme turları arasındaki en uzun sürenin toplamının alınması seyahat süresinin totalini oluşturacaktır. k=9 alarak bulunan sürelere bakıldığında merkezler arası seyahat süresi 44 saat 7 dakika, kümelerin iç turlarına bakıldığında ise en uzun süreye sahip 9. kümeye ait olan 24 saat 17 dakikadır. Bu hesaplama ile tüm kargoların yerine ulaşma süresi olarak ikisinin toplamı olan 68 saat 24 dakika alınabilir.

Ahmet Yekta KAYMAN'ın 2015 yılında yaptığı çalışmada; GSP için standart PSO kullanarak ve bulanık c-ortalamalar kümeleme yöntemini PSO ile birleştirip 3 varyant oluşturarak bir karşılaştırma çalışması yapmıştır. Bu karşılaştırmaları ise küçük ölçekli, orta ölçekli ve büyük ölçekli problemler şeklinde üç grupta değerlendirmiştir. Değerlendirmeler sonucunda; küçük ölçekli GSP'ler için çözüm kalitesi yönünden standart PSO'dan daha iyi sonuçlar elde edilirken çalışma süresi kriteri açısından standart PSO'ya göre kötü sonuçlar elde edilmiştir. Orta ve büyük ölçekli GSP'ler için ise geliştirilmiş varyantlar hem çözüm kalitesi hem de çalışma süresi açısından standart PSO'ya göre daha iyi sonuçlar vermiştir. Böylece bulanık c-ortalamalar kümeleme yöntemi ile oluşturduğu varyantların her boyuttaki problemlerin çözüm kalitesi açısından, orta ve büyük ölçekli problemlerin hem çalışma süresi hem de çözüm kalitesi açısından daha uygun ve avantajlı olduğunu göstermiştir.

Bu çalışmada ise GSP, ERA ile çalıştırılarak tek gezgin satıcı sonuçları PSO, KKA ve BTA ile karşılaştırması yapılmıştır. İkinci bölümde ise k-ortalamalar kümeleme analizi ile harita 7, 8 ve 9 kümeye bölünmüş ve k-gezgin satıcı performansı değerlendirilmiştir. Yapılan testler sonucunda k-gezgin satıcının, tek gezgin satıcıya göre ulaşım süreleri açısından daha gerçekçi ve uygulanabilir sonuçlar verdiği görülmüştür. Türkiye haritası için 7 bölgenin daha stabil

sonuçlar verdiği gözlemlenmiştir. Daha uygulanabilir ve ulaşım sürelerinin azaltılması adına kümelerin birde alt kümeleri oluşturulması önerilmektedir.

Bu çalışmada hedeflenen GSP çözümünün performansının değerlendirilmesinin yanında Türkiye haritasındaki 81 için uygulanabilirlik değerlendirilmesi yapılmıştır. Hem tek gezgin satıcı için gerçekçilik şartı aranmıştır hem de kümeleme analizi ile yolculuk süresi değerlendirilmesi yapılmıştır. Sonuç olarak da kümelemenin gerçek hayata uyarlanabilirlik açısından uygun olduğu görülmüştür.

Bu çalışma kapsamında öneriler k-ortalamalar kümeleme analizinin hiyerarşik kümeleme işlemi yapılarak, kümelerin alt kümeleri oluşturulmasının performansı ve uygulanabilirliği arttıracağı beklenmektedir.

6. KAYNAKLAR

[1] A. Y. Kayman, "Gezgin satıcı probleminin çözümünde parçacık sürü optimizasyonu algoritması performansının bulanık c-ortalamalar yöntemi ile iyileştirilmesi", Doktora tezi, Endüstri Mühendisliği, Fen Bilimleri Enstitüsü, Kocaeli Üniversitesi, Kocaeli, Türkiye, 2015.

[2] G. Filiz, "Elektromanyetik algoritmanın gezgin satıcı problemine uyarlanması", Yüksek lisans tezi, Endüstri Mühendisliği, Fen Bilimleri Enstitüsü, İstanbul Üniversitesi, İstanbul, Türkiye, 2012.

[3] S. Kuzu, O. Önay, U. Şen, M. Tunçer, B. F. Yıldırım, ve T. Keskintürk, "Gezgin satıcı problemlerinin metasezgiseller ile çözümü", *İstanbul Üniversitesi İşletme Fakültesi Dergisi.*, c. 43, sayı 1, ss. 1–27, 2014.

[4] M. Held ve R. Karp, "The traveling salesman problem and minimum spanning trees", *Operations Research*, c. 18, sayı 6. ss. 1138–1162, 1969.

[5] S. Lin ve B. W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem", *Operations Research*, c. 21, sayı 2, ss. 498–516, 1973.

[6] N. Christofides, "Technical note-bounds for the traveling salesman problem", *Opererations Research,* c. 20, ss. 1044–1056, 1972.

[7] J. L. Bentley, "Experiments on traveling salesman heuristics", *Proceedings of the First Annual ACM-SIAM Symposium*, San Francisco, California, USA: Societs for Industrial and Applied Mathematics, 1990, ss. 91–99.

[8] G. Reinelt, "TSLIB-a traveling salesman problem library", *Informs Journal on Computing*, c. 3, sayı 4, ss. 376–384, 1991.

[9] A. Joines, M. G. Kay, M. F. Karabacak, K. Karagül, ve S. Tokat, "Performance analysis of genetic algorithm optimization toolbox via traveling salesman problem", *Contemporary Issues In Social Sciences and Humanities,* London, UK, 2017, ss. 213–221.

[10] K. Karagül, "Gezgin satıcı probleminin çözümü için macar algoritması esaslı yeni bir çözüm yaklaşımı", *Mühendislik Bilim. ve Tasarım Dergisi*, c. 7, sayı 3, ss. 561–571, 2019.

[11] C. Rego, D. Gamboa, F. Glover, ve C. Osterman, “Traveling salesman problem heuristics: Leading methods, implementations and latest advances”, *European Journal of Operational Research*, c. 211, sayı 3, ss. 427–441, 2011.

[12] M. Chatting, “A comparison of exact and heuristic algorithms to solve the travelling salesman problem”, *Plymouth Student Scientist*, c. 11, sayı 2, ss. 53–91, 2018.

[13] E. Önder, “Araç rotalama problemlerinin parçacık sürü ve genetik algoritma ile optimizasyonu”, Doktora tezi, İşletme, Sosyal Bilimler Enstitüsü, İstanbul Üniversitesi, İstanbul, Türkiye, 2011.

[14] K. Akşehir, “Gezgin satıcı probleminin karınca kolonisi algoritması ile çözüm performansının arttırılmasında parametre optimizasyonu”, Yüksek lisans tezi, İstatistik, Fen Bilimleri Enstitüsü, Ondakuz Mayıs Üniversitesi, Samsun, Türkiye, 2019.

[15] P. S. Yalçın, “Seçici gezgin satıcı problemi için yeni matematiksel modeller”, Yüksek lisans tezi, Endüstri Mühendisliği, Fen Bilimleri Enstitüsü, Başkent Üniversitesi, Ankara, Türkiye, 2014.

[16] M. I. Hosny ve C. L. Mumford, “The single vehicle pickup and delivery problem with time windows: Intelligent operators for heuristic and metaheuristic algorithms”, *Journal of Heuristics*, c. 16, sayı 3, ss. 417–439, 2010.

[17] S. K. Yadlapalli ve S. Darbha, “3-Approximation algorithm for a two depot, heterogeneous traveling salesman problem”, *Optimization Letters*, c. 6, sayı 1, ss. 141–152, 2012.

[18] C. A. Gencel, “Otel seçimli gezgin satıcı problemi için yeni matematiksel modeller”, Yüksek lisans tezi, Endüstri Mühendisliği, Fen Bilimleri Enstitüsü, Başkent Üniversitei, Ankara, Türkiye, 2019.

[19] M. B. Abdulrazaq, Y. S. Tahir, S. Sha’Aban, ve M. S. Jibia, “Polynomial reduction of TSP to freely open-loop TSP”, *2019 2nd International Conference of the IEEE Nigeria Computer Chapter, Nigeria Comput Conference*, 2019, ss. 3–6.

[20] Z. Wang, J. Guo, M. Zheng, ve Y. Wang, “Uncertain multiobjective traveling salesman problem”, *European Journal Operational Research*, c. 241, sayı 2, ss. 478–489, 2015.

[21] L. M. Mou, “The continuous selective generalized traveling salesman problem: An efficient ant colony system”, *Proceedings - International Conference on Natural Computation*, 2012, sayı Icnc, ss. 1242–1246.

[22] K. Helsgaun, “An effective implementation of the Lin-Kernighan traveling salesman heuristic”, *European Journal Operational Research*, c. 126, sayı 81, ss. 106–130, 1998.

[23] E. Erdem, “Olumsuz hava koşullarını dikkate alan uçuş çizelgeleme problemi için metasezgisel yaklaşımlar”, Yüksek lisans tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, Atatürk Üniversitesi, Erzurum, Türkiye, 2019.

[24] B. Satar, “Çok amaçlı optimizasyon temelli genel atama problemlerinin metasezgisel yöntemlerle çözümü”, Yüksek lisans tezi, Elektrik-Elektronik Mühendisliği, Fen Bilimleri Enstitüsü, Ankara Üniversitesi, Ankara, Türkiye, 2015.

[25] H. Ranjbar, S. F. Zarei, ve S. H. Hosseini, “Imperialist competitive algorithm based optimal power flow”, *22nd Iranian Conference on Electrical Engineering, ICEE 2014*, 2014, ss. 746–750.

[26] J. Kennedy ve R. Eberhart, “Particle swarm optimization”, *Proceedings of ICNN’95 - International Conference on Neural Networks*, 1995, ss. 1942–1948.

[27] P. Erdoğmuş, “Particle swarm optimization performance on special linear programming problems”, *Scientific Research and Essays*, c. 5, sayı 12, ss. 1506–1518, 2010.

[28] M. Kasap, “Parçacık sürüsü optimizasyonu algoritmasının başarım analizi”, Yüksek lisans tezi, Elektrik ve Elektronik Mühendisliği, Hacettepe Üniversitesi, Ankara, Türkiye, 2015.

[29] Y. E. Demirtaş, “Dïnamïk araç rotalama problemïne parçacık sürü optimizasyonu algoritmasi çözüm önerisi”, Doktora tezi, İşletme, Sosyal Bilimler Enstitüsü, İstanbul Üniversitesi, İstanbul, Türkiye, 2015.

[30] S. Öztürk, C. Karakuzu, M. Kuncan, ve A. Erdil, “Fuzzy Neural Network Controller As a Real Time Controller Using Pso”, *Akademik Platform Mühendislik ve Fen Bilimleri Dergisi*, c. 5, sayı 1, ss. 15–22, 2017.

[31] H. A. Alhasan, “Parçacık sürü optimizasyonu (PSO) kullanarak öz ayarlamalı PID

kontrolör tasarım", Yüksek lisans tezi, Elektrik Elektronik Mühendisliği, Fen Bilimleri Enstitüsü, Kahramanmaraş Sütçü İmam Üniversitesi, Kahramanmaraş, Türkiye, 2018.

[32] S. Goss, S. Aron, J. L. Deneubourg, ve J. M. Pasteels, "Self-organized shortcuts in the argentine ant", *Naturwissenschaften*, c. 76, ss. 579–581, 1989.

[33] S. Aslan, "Ders çizelgeleme probleminin paralel karınca kolonisi algoritması ile çözümü: Mersin Üniversitesi örneği", Yüksek lisans tezi, Elektrik Elektronik Mühendisliği, Fen Bilimleri Enstitüsü, Mersin Üniversitesi, Mersin, Türkiye, 2018.

[34] K. Alaykıran ve O. Engin, "Karınca kolonileri metasezgiseli ile ve gezgin satıcı problemleri üzerinde bir uygulaması", *Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi*, c. 20, sayı 1, ss. 69–76, 2005.

[35] K. Loubna, B. Bachir, ve Z. Izeddine, "Optimal gigital IIR filter design using ant colony optimization", *2018 4th International Conference on Optimization and Applications (ICOA)*, 2018, ss. 1–5.

[36] R. S. Jadon ve U. Datta, "Modified ant colony optimization algorithm with uniform mutation using self-adaptive approach for travelling salesman problem", *2013 4th International Conference on Computing, Communications and Networking Technologies*, 2013, ss. 1–4.

[37] T. Keskintürk ve H. Söyler, "Global karınca kolonisi optimizasyonu", *Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi*, c. 21, sayı 4, ss. 689–698, 2006.

[38] H. H. Dodurgalı, "Karınca kolonisi optimizasyonu ile eğitilmiş çok katmanlı yapay sinir ağı ile sınıflandırma", Yüksek lisans tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, İstanbul Teknik Üniversitesi, İstanbul, Türkiye, 2010.

[39] E. Çelik, "Bulut görev çizelgelemesi için benzetilmiş tavlama tabanlı bir optimizasyon yaklaşımı", Yüksek lisans tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, Atatürk Üniversitesi, Erzurum, Türkiye, 2018.

[40] Pakize Erdoğmuş, "Doğadan esinlenen optimizasyon algoritmaları ve optimizasyon algoritmalarının optimizasyonu", *Düzce Üniversitesi Bilim ve Teknoloji Dergisi*, c. 4, ss. 293–304, 2016.

[41] S. Kirkpatrick, C. D. Gelatt, ve M. P. Vecchi, "Optimization by simulated annealing", *Science*, c. 220, sayı 4598, ss. 671–680, 1983.

[42] R. W. Eglese, "Simulated annealing : a tool for operational research", *European Journal Operation Reserach*, c. 46, ss. 271–281, 1990.

[43] B. Haznedar, "Benzetilmiş tavlama algoritması ile adaptif ağ tabanlı bulanık mantık çıkarım sisteminin (ANFIS) eğitilmesi", Doktora tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, Erciyes Üniversitesi, Kayseri, Türkiye, 2017.

[44] R. A. Zeineldin, "An improved simulated annealing approach for solving the constrained optimization problems", *2012 8th International Conference on Informatics and Systems (INFOS)*, s. 27-31.

[45] E. Atashpaz-Gargari ve C. Lucas, "Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition", *2007 IEEE Congress on Evolutionary Computation*, 2007, ss. 4661–4667.

[46] A. Yelghi, "Değiştirilmiş ateş böceği algoritması ve veri yoğunluğu kümelemesine uygulanması", Doktora tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, Karadeniz Teknik Üniversitesi, Trabzon, Türkiye, 2018.

[47] S. Hosseini ve A. Al Khaled, "A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research", *Applied Soft Computing*, c. 24, ss. 1078–1094, 2014.

[48] M. Baygan ve M. Baygan, "A new method for solving the open shop scheduling using imperialist competitive algorithm and tabu search with regard to maintenance of machine", *2015 2. International Cnoference On Knowledge-Based Engineering And Innovation(KBEI)*, 2015, ss. 3–8.

[49] M. Sarıkoç, "Dinamik sistemlerin ANFIS ile modellenmesinde yayılımcı rekabetçi optimizasyon (ICA) algoritmasının kullanılması", Yüksek lisans tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, Bilecik Şeyh Edebali Üniversitesi, Bilecik, Türkiye, 2014.

[50] M. H. Sayadnavard, A. T. Haghighat, ve M. Abdechiri, "Wireless sensor network localization using imperialist competitive algorithm", *2010 3rd IEEE International Conference Computer Science Information Technology ICCSIT 2010*, c. 9, ss. 818–

822, 2010.

[51] Y. Wang, Y. Wei, M. Zhu, ve Y. Zhang, "Solar sail spacecraft trajectory optimization based on improved imperialist competitive algorithm", *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)*, 2012, sayı 4, ss. 191–195.

[52] S. J. Mousavirad ve H. Ebrahimpour-Komleh, "Feature selection using modified imperialist competitive algorithm", içinde *Proceedings of the 3rd International Conference on Computer and Knowledge Engineering, ICCKE 2013*, 2013, ss. 400–405.

[53] M. Demirkan, "Kategorik ve kategorik olmayan verilerden oluşan veri setleri için k-ortalama tabanlı bir yaklaşım", Yüksek lisans tezi, Bilgisayar Mühendisliği, Fen Bilimleri Enstitüsü, Beykent Üniversitesi, Ankara, Türkiye, 2014.

[54] K. Koşuta, "K-ortalamalar algoritması ile ileriye dönük modellemeler", Yüksek lisans tezi, Matematik Mühendisliği, Fen Bilimleri Enstitüsü, Yıldız Teknik Üniversitesi, İstanbul, Türkiye, 2005.

[55] M. Yousefikhoshbakht ve M. Sedighpour, "New imperialist competitive algorithm to solve the travelling salesman problem", *International Journal of Computer Mathematics*, c. 90, sayı 7, ss. 1495–1505, 2013.

[56] K. G. Müdürlüğü, (2021, 15 Şubat). *İller arası mesafe tablosu*. Erişim: https://www.kgm.gov.tr/SiteCollectionDocuments/KGMdocuments/Root/Uzakliklar/ilmesafe.xls.

[57] M. Göçenoğlu, (2021, 27 Ocak). *İl merkezleri koordinatları*. Erişim: http://mgocenoglu.blogspot.com/2014/01/illerin-koordinatlar-ondalk-cinsinden.html.

[58] K. G. Müdürlüğü, (2021, 1 Mart). *Güzergah Analizi*. Erişim: https://yol.kgm.gov.tr/guzergahanalizi.

[59] H. Dikmen, H. Dikmen, A. Elbir, Z. Ekşi, ve F. Çelik, "Gezgin satıcı probleminin karınca kolonisi ve genetik algoritmalarla eniyilemesi ve karşılaştırılması", *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, c. 18, sayı 1, ss. 8–13, 2014.

[60] O. Köse ve P. Erdoğmuş, "K-gezgin satıcı probleminin emperyalist rekabetçi

algoritması ile kümeleme tabanlı optimizasyonu", *Bartın Üniversitesi Mühendislik ve Teknoloji Bilimleri Dergisi*, c. 3, sayı 2, ss. 42–50, 2015.

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Oktay KÖSE

Yabancı Dili : İngilizce

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Müh.	Karabük Üniversitesi	2013
Lisans	Bilgisayar Sis. Öğrt.	Süleyman Demirel Üniversitesi	2005
Lise	Bilgi İşlem	Karabük Anadolu Ticaret Meslek Lisesi	2000

YAYINLAR

O. Köse ve P. Erdoğmuş, "K-gezgin satıcı probleminin emperyalist rekabetçi algoritması ile kümeleme tabanlı optimizasyonu", *Bartın Üniversitesi Mühendislik ve Teknoloji Bilimleri Dergisi*, c. 3, sayı 2, ss. 42–50, 2015.