T.C.

YAŞAR ÜNİVERSİTESİ

SOSYAL BİLİMLER ENSTİTÜSÜ

İŞLETME ANABİLİM DALI

YÜKSEK LİSANS TEZİ

YEREL ARAMALI KESİKLİ FARKSAL EVRİM ALGORİTMASI VE KESİKLİ PARÇACIK SÜRÜ EN İYİLEME ALGORİTMASI KULLANARAK GEZGİN SATICI PROBLEMİNİN ÇÖZÜMÜ

Yeliz KOCAMAN

Danışmanlar

Prof. Dr. Mehmet Edip Teker

Doç. Dr. Mehmet Fatih Taşgetiren

İzmir, 2010

T.C.

YAŞAR ÜNİVERSİTESİ

SOSYAL BİLİMLER ENSTİTÜSÜ

İŞLETME ANABİLİM DALI

YÜKSEK LİSANS TEZİ

YEREL ARAMALI KESİKLİ FARKSAL EVRİM ALGORİTMASI VE KESİKLİ PARÇACIK SÜRÜ EN İYİLEME ALGORİTMASI KULLANARAK GEZGİN SATICI PROBLEMİNİN ÇÖZÜMÜ

Yeliz KOCAMAN

Danışman

Prof. Dr. Mehmet Edip Teker

Doç. Dr. Mehmet Fatih Taşgetiren

İzmir, 2010

T.C.

YAŞAR ÜNİVERSİTESİ

SOSYAL BİLİMLER ENSTİTÜSÜ

YÜKSEK LİSANS TEZ SINAV TUTANAĞI

Öğrencinin Adı ve Soyadı :

Tez Konusu:

Anabilim Dalı/ Programı :

Sınav Tarihi ve Saati :

Yukarıda kimlik bilgileri belirtilen öğrenci Enstitü Yönetim Kurulu'nun ……...…….. tarih ve ….....……. sayılı toplantısında oluşturulan jürimiz tarafından Yaşar Üniversitesi Sosyal Bilimler Enstitüsü Lisansüstü Yönetmeliğinin 23/37.maddesi gereğince yüksek lisans tez sınavına alınmıştır.

Adayın kişisel çalışmaya dayanan tezini … dakikalık süre içinde savunmasından sonra jüri üyelerince gerek tez konusu gerekse tezin dayanağı olan Anabilim dallarından sorulan sorulara verdiği cevaplar değerlendirilerek tezin,

BAŞARILI O OY ÇOKLUĞU O

OY BİRLİĞİ ile O

DÜZELTME O

RED edilmesine O ile karar verilmiştir.

Jüri teşkil edilmediği için sınav yapılamamıştır. O

Öğrenci sınava gelmemiştir. O

	Evet
Tez burs, ödül veya teşvik programlarına (Tüba, Fullbrightht vb.) aday olabilir.	O
Tez mevcut hali ile basılabilir.	O
Tez gözden geçirildikten sonra basılabilir.	O
Tezin basımı gerekliliği yoktur.	O

JÜRİ ÜYELERİ		İMZA
……	□Başarılı □Düzeltme □Red	….……..……
……	□Başarılı □Düzeltme □Red	….……..……
……	□Başarılı □Düzeltme □Red	….……..……

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum “Yerel Aramalı Kesikli Farksal Evrim Algoritması ve Kesikli Parçacık Sürü En İyileme Algoritması Kullanarak Gezgin Satıcı Probleminin Çözümü” adlı çalışmanın, tarafımdan bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın yazıldığını ve yararlandığım eserlerin bibliyografyada gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve bunu onurumla doğrularım.

..../..../2010

Yeliz KOCAMAN

ÖZET
Yüksek Lisans
YEREL ARAMALI KESİKLİ FARKSAL EVRİM ALGORİTMASI VE KESİKLİ PARÇACIK SÜRÜ EN İYİLEME ALGORİTMASI KULLANARAK GEZGİN SATICI PROBLEMİNİN ÇÖZÜMÜ
Yeliz Kocaman
Yaşar Üniversitesi
Sosyal Bilimler Enstitüsü
İşletme Yüksek Lisans Programı

Bu tez çalışmasında gezgin satıcı problemi (GSP), farksal evrim algoritması ve doğadaki balık kuş gibi sürü halinde hareket eden canlılardan esinlenilerek tasarlanmış parçacık sürü en iyileme algoritması ile çözülmüş ve performansları karşılaştırılmıştır. Algoritmalar, TSBLIB' den alınan 48 ile 1084 arasında şehir sayısına sahip 54 test problemi ile çalıştırılmıştır.

FEA ve PSE algoritmaları sürekli değerlerden oluşan problemlerin çözümü için tasarlanmıştır. Gezgin satıcı problemi ise kesikli değerlerden oluşan bir problemdir. Bu nedenle algoritmalar yerel arama teknikleri ile birtakım değişikliğe uğratılarak GSP ye uygulanmıştır. Yerel arama teknikleri olarak NEH sezgiseli, ekleme sezgiseli, boz yap sezgiseli, PTL çaprazlama işlemcisi ve 2-opt kullanılmıştır. Algoritmalar çalıştırılmadan önce sonuca etkisi olan faktörleri belirlemek için deney tasarımı yapılmıştır. Deney sonucunda problemin sonucuna etkisi olan faktörler ve seviyeleri belirlenmiş ve test problemlerine uygulanmıştır.

Sonuç olarak, GSP çözümünde en iyi çözümü bulmada her iki algoritmanın eşdeğer olduğu sonucuna varılmıştır ancak hesaplama süresi açısından kesikli parçacık sürü en iyileme algoritmasının kesikli farksal evrim algoritmasına göre daha iyi performans gösterdiği sonucuna varılmıştır.

Anahtar Kelimeler: Gezgin satıcı problemi, en iyileme, kesikli farksal evrim algoritması, kesikli parçacık sürü algoritması.

ABSTRACT

Master Thesis

SOLVING TRAVELING SALESMAN PROBLEM BY DISCRETE DIFFERENTIAL EVOLUTION ALGORITHM AND DISCRETE PARTICLE SWARM OPTIMIZATION ALGORITHM WITH LOCAL SEARCH

Yaşar University

Institute of Social Sciences

Master of Business Administration

In this paper, traveling salesman problem is solved by differential evolution algorithm and particle swarm optimization algorithm that is generated by the inspiration of the animals that live as a swarm such as bird and fish. Then the performance of these methods are compared. The comparison is iplemented to the well-known 54 test problem with 48 and 1084 cities.

Differential evolution algorithm and particle swarm optimization algorithm is designed in order to solve the continuous problems howewer TSP is a discrete problem so that the algorithms are modified to TSP by some local search techniques. NEH, insertion, destruction-construction, PTL crossover operator and 2-opt techniques are used for local search. Before the algorithms are performed, the factors that impacts the results are determined by design of experiment. According to the conclusion of experimental design, the factors and the levels are determined and then they are implemented to the test problems.

As a conclusion, discrete differential evolution algorithm and discerete particle swarm algorithm are equivalent while obtaining the optimal solution but DPSO algorithm have better performance than DDEA in terms of processing (CPU) time.

Keywords: Traveling salesman problem, optimization, discrete differantial evolution algorithm, discerete particle swarm algorithm.

TEŞEKKÜR

Bu tez çalışması süresince ve yüksek lisans eğitimim boyunca, bilgi ve tecrübeleriyle bana daima yol gösteren danışmanım Sayın Prof. Dr. Mehmet Edip Teker'e, yazılım konusundaki bilgilerini benimle paylaşan, tez çalışmamın her aşamasında yardımlarını esirgemeyen, ve her türlü olanağı sağlayan ikinci danışmanım Sayın Doç. Dr. Mehmet Fatih Taşgetiren'e, derslerimde emeği geçen tüm hocalarıma, çalışmam esnasında desteklerini esirgemeyen aileme ve arkadaşlarıma içten ve sonsuz teşekkürlerimi sunarım.

İÇİNDEKİLER

TABLO DİZİNİ

ŞEKİLLER DİZİNİ

SEMBOLLER VE KISALTMALAR DİZİNİ

AAÇ: Açgözlü Alt Tut Çaprazlaması(GSX: Greedy Subtour Crossover)
BY: Boz-Yap (DC: destruction construction)
$BozYap_{dp}$: bozulma derecesinin d, perturbasyon derecesinin p olduğu boz-yap işlemi
$b_{j,U}$: Değişkenin Üst Sınır Değeri
$b_{j,L}$: Değişkenin Alt Sınır Değeri
CR: Çaprazlama Oranı
D: Parametre Sayısı (değişken sayısı(gen sayısı) (1, 2, 3, ., *j*))
DKA: Değişken Komşuluk Araması (VNS: Variable Neighborhood Search)
EPD: Enküçük Pozisyon Değeri (SPV: Smallest Position Value)
FEA: Farksal Evrim Algoritması (DEA: Differantial Evolution Algorithm)
F: Ölçeklendirme Faktörü (mutasyon faktörü)
g_{best}:küresel en iyi (bir nesilde ulaşılan en iyi parçacık konumu)
GA: Genetik Algoritma (GA: Genetic Algorithm)
GFE: Genetik Farksal Evrim (GDE: Genetic Differential Evolution)
GSP: Gezgin Satıcı Problemi (TSP: Traveling Salesman Problem)
GGSP: Genelleştirilmiş Gezgin Satıcı Problemi (GTSP:Generalized Traveling Salesman Problem)
PSE: Parçacık Sürü Eniyileme (PSO: Particle Swarm Optimization)
j_{rand}: yeni kromozomun üretilmesinde kullanılacak rassal seçilmiş kromozomlar
k :Nesil indeksi
KFEA: Kesikli Farksal Evrim Algoritması (DDEA:Discrete Differensial Evolution Algorithm)
KPSE: Kesikli Parçacık Sürü Eniyileme (DPSO: Discerete Particle Swarm Optimization)
LKH Lin-Kernighan-Heulsgaun
LS (Local Search): Yerel Arama
MR: Mutasyon Oranı
NP: Popülasyon Büyüklüğü
P_m: Mutasyon (bozulma) Olasılığı
P_c : Çaprazlama Olasılığı
p_{best}: kişisel en iyi (parçacığın içinde bulunduğu nesile kadar ulaşmış olduğu en iyi konumu)
SÇ: Sıralı Çaprazlama (OC: Ordered Crossover)
USÇ: Uyarlanmış Sıralı Çaprazlama (MOX: Modified Ordered Crossover)
u: Deneme vektör
v: Mutant vektör
π: Hedef Vektör
π_g: Hedef popülasyondaki en iyi birey

BÖLÜM 1

1. GİRİŞ

Gezgin satıcı problemi (GSP), aralarındaki uzaklıkların bilindiği nokta ya da düğüm olarak da adlandırılan *n* tane şehrin her birine bir kez mutlaka uğrayarak başlangıç şehrine geri döndüğü şekilde, satıcının en kısa ya da en az maliyetli turu bulmasını hedefleyen bir problemdir. Problem kombinasyonel en iyileme (optimizasyon) problemleri kapsamına girmektedir. Problemin kombinasyonel olması, olası çözümlerinin hem sonlu, hem de kesikli (discrete) yani sayılabilir (enumerable) olması demektir. GSP de olası çözümler yani şehirlerin permütasyonu sonlu kesikli bir set oluşturmaktadır.

Problemin çözümünde kullanılan yöntemlerinden dal ve sınır yöntemi (branch and bound), dal ve kes (branch and cut) yöntemi kesin çözüm veren yöntemlerdir. Ancak problemin çözümünde satıcı n! tur arasından seçim yapmaktadır. Bu durumda satıcının ziyaret edeceği şehir sayısı arttıkça çözüme ulaşmak da uzun zaman alacak ve dolayısı ile zorlaşacaktır. Bu nedenle araştırmacılar tarafından sezgisel yöntemler geliştirilmiştir. Sezgisel yöntemler kesin sonucu vermez ancak makul sürede kesin çözüme yakın bir çözüm elde etmeyi sağlamaktadır.

Genetik algoritma (genetic algorithm), benzetimli tavlama (simulated annealing), yasak arama (tabu search), karınca kolonisi en iyileme (ant colony optimization), yapay sinir ağları (neural networks) yöntemleri GSP çözümünde kullanılan sezgisel yöntemlerdir.

Problemde uzaklıklar öklit mesafesi (kuş bakışı mesafe) olarak alınır. Trafikte tek yön olması ya da gidiş-dönüş yollarının farklı olması gibi bazı durumlarda A şehrinden B şehrine gitme maliyeti ile B şehrinden A şehrine gitme maliyeti farklı olabilmektedir. Bu durumda problem asimetrik (non-symetric) GSP olarak tanımlanmaktadır aksi durumda ise problem simetrik GSP olarak tanımlanmaktadır. Bu tezde simetrik GSP çalışılmıştır.

GSP detaylı olarak bölüm 2 de anlatılmıştır. Ayrıca bu bölümde en iyileme, en iyileme teknikleri, çözüm için kullanılan yöntemler yani evrimsel algoritmalar, farksal evrim algoritması, parçacık sürü en iyileme algoritması ve bu algoritmaların

daha iyi sonuç verebilmesi için geliştirilen yerel arama sezgiseli olan NEH sezgiseli, ekleme sezgiseli, boz-yap (destruction construction) sezgiseli, PTL çaprazlama işlemcisi ve 2- opt sezgisel yöntemleri anlatılmıştır. Dahası farksal evrim algoritması ve parçacık sürü en iyilemesi algoritması ile çözülmüş GSP konusunda daha önce yapılmış çalışmalardan bahsedilmiştir.

3. bölümde çözüm yöntemlerinin parametrelerini belirlemek için kullanılan deney tasarımı yönteminden bahsedilmiştir. 4. bölümde ise GSP kütüphanesinden (TSPLIB) alınan 54 ayrı test problemi üzerinde deneysel çalışmalar yapılmış ve sonuçları değerlendirilmiştir.

1.1. Amaç

Gerçek yaşam problemleri olan dağıtım, araç rotalama, iş planlama, bütünleşik devre tasarımı, çizelgeleme, fiziksel haritalama, şehir boru hattı döşeme, malzeme akış gibi pek çok problem tipi gezgin satıcı problemi olarak formüle edilebilmektedir. Tezin amacı, GSP olarak formüle edilmiş bu tip problemleri yeni bir yaklaşım olan yerel arama sezgiseli ile birleştirilmiş kesikli farksal evrim ve kesikli parçacık sürü en iyilemesi yaklaşımları ile çözüp en kısa turu belirlememizi sağlayan yani en iyi (optimal) sonucu veren yöntemi belirlemektir.

1.2. Yöntem

Bu çalışmada GSP çözümü için kullanılan yöntemler farksal evrim algoritması ve parçacık sürü en iyileme algoritmasıdır. Ancak bu yaklaşımlar çözüm kümeleri sürekli uzayda olan problemler için tasarlanmıştır. GSP ise çözüm kümesi kesikli uzayda olan bir problemdir. Bu nedenle çözümde kullanılan yaklaşımlar öncelikle kesikli problem olan GSP ye uyarlanmıştır. Yöntemlerin uygulama aşamasında olası çözümler zaman zaman yerel minimumlara takılabilmektedir. Yani sadece bir bölgeye mahsus en küçük uygunluk değerini veren çözüm bulunabilmektedir. Bu durum da karşılaşabileceğimiz daha iyi çözümler ile karşılaşamamamıza neden olmaktadır. Bu nedenle çalışmada çözümün bazı aşamalarında NEH, ekleme, boz-yap, PTL çaprazlama işlemcisi, 2-opt gibi yerel arama uygulamaları yapılmaktadır. Bu şekilde yerel değil küresel en iyi çözüm elde edilmektedir.

1.3. Literatür Taraması

1.3.1. Farksal Evrim Algoritması ile ilgili Literatür Taraması

Sauer ve Coelho (2008) çalışmalarında Gezgin Satıcı Problemini çözmek için Kesikli Farksal evrim algoritması kullanmıştır. Çalışmada öncelikle 4 farklı yöntemden bahsedilmiş ve problemler bu dört yöntem ile çözülüp değerlendirme yapılmıştır.
Yöntemler:

i) Yerel aramasız farksal evrim algoritması,
ii) LKH (Lin-Kernighan-Heulsgaun) yöntemine dayanan yerel aramalı farksal evrim algoritması,
iii) Değişken Komşuluk Araması (DKA) yöntemine dayanan yerel aramalı farksal evrim algoritması,
iv) LKH ve DKA yöntemine dayanan yerel aramalı farksal evrim algoritması şeklindedir.

Yapılan çalışmada FE algoritmasının *DE/rand/1/bin* varyasyonu uygulanmıştır. Notasyondaki rand: mutasyona uğramış vektörü oluşturan kromozom çiftinin seçilme yönteminin rassal olduğunu göstermekte,
1: Başlangıç kromozomları bozmak için kullanılan farklılık vektörlerinin sayısının 1 olduğunu göstermekte,
Bin: Çocuk popülasyonunun yaratılması için kullanılan çaprazlama mekanizmasının bir dizi bağımsız binom dağılımı ile kontrol edildiğini göstermektedir.

Sauer ve Coelho, kesikli farksal evrim algoritmasının yapısındaki süreklilikten dolayı başarılı sonuçlara zor ulaşıldığını belirtmiştir. Bu engeli ortadan kaldırmak için en küçük pozisyon değeri (EPD) yöntemi kullanılmaktadır. Yerel aramalı farksal evrim algoritması ise simetrik GSP' lerde etkin sonuçlar vermektedir.

GSP kütüphanesinden alınan test problemleri ile sayısal çalışmalar yapılmış ve literatürdeki 317 ve 7197 şehirli problemler ile karşılaştırılmıştır. Sonuç olarak, LKH ve DKA yöntemine dayanan yerel aramalı farksal evrim algoritması en iyi sonuçlardan ortalama nısbi sapma yüzdesi klasik farksal evrim algoritmasına göre daha iyi vermiştir ve çalıştırma zamanı da klasik algoritmaya göre daha kısa olarak belirlenmiştir. Bu sonuçlar farksal evrim algoritmasının tek başına GSP çözümü için

yeterli olmadığını LKH yöntemi ile birlikte kullanıldığında en iyi sonuca (optimale) daha yakın çözümler elde edildiğini hatta DKA yöntemi ile de birleştirildiğinde literatürdeki en iyi çözümlere dahi ulaşılabildiğini göstermektedir. Sauer ve Coelho, çok amaçlı GSP problemlerinde de aynı yöntemleri denemenin faydalı olabileceğini öne sürmektedirler.

Jian, Peng ve Zhiming (2008) GSP nin çözümü için genetik yeniden üretme mekanizmasıyla birleştirilmiş farksal evrim algoritmasından türetilmiş genetik farksal evrim algoritması kullanılmıştır. Çocuk bireylerin üretilmesi için açgözlü alt tur çaprazlaması (AAÇ) uygulanmıştır. 3 farklı bireye dayanan mutasyon ve çaprazlama uygulamak için uyarlanmış sıralı çaprazlama (USÇ) uygulanmıştır ayrıca yerel arama performansını artırmak için 2-opt yerel arama uygulanmıştır. Çalışmada 52, 100, 200 şehirlik test problemleri kullanılmıştır.

Genetik farksal evrim algoritması adımları şu şekilde belirtilmiştir.

Kodlama: $X_i^t = (x_{i,1}^t, x_{i,2}^t, x_{i,D}^t)$ d tüm şehir sayısını göstermekte.

$x_{i,d}^t$ d. şehri göstermektedir.

Uyarlanmış sıralı çaprazlama (USÇ): USÇ, sıralı çaprazlamadan türetilmiş 2 vektörü çaprazlamak için uygulanır. USÇ da esas turun sıralı bir bölümü hedef vektöre eklenir ve takrarlanan şehirler turdan çıkartılır.

Mutasyon: Genetik farksal evrim algoritmasında mutant vektör, hedef vektör ve AAÇ ile çaprazlanmış 2 vektörün çaprazlanması ile elde edilir. AAÇ da açgözlü sezgiseli uygulanmaktadır.

Çaprazlama: Mutasyondan sonra USÇ ya dayalı bir çaprazlama kullanılır. Len_2 kullanılır ve CR gibi çalışır. Hedef vektör ile mutant vektörün oranlarını belirler.

Yerel Arama: Turu kısaltmak için 2-opt yerel arama yöntemi uygulanmaktadır.

Seçim: Farksal evrim algoritmasın daki ile aynı şekilde uygulanmaktadır.

Çalışmada popülasyon ölçüsü 30 olan, 52, 100 ve 200 şehirli 100 üretme yapılmıştır. 52 şehirli uygulamada GFE tutarlı bulunmuştur. Len_1 ve Len_2 nin %25 olduğu durumda etkin üretim sağlanmıştır.

100 şehirli uygulamada yine Len_1 ve Len_2 nin %25 olduğu durumda tüm denemeler en iyiye ulaşmıştır. 200 şehirli uygulamada her Len_2 için en düşük Len_1 değerine sahip deneme en iyi sonucu vermiştir.

Aynı problemler farklı Len değerleri ile takrar çözülmüştür. Sonuç olarak yapılan analizlere göre GDE algoritmasının GSP i için etkin bir çözüm yöntemi olduğu saptanmıştır. Ancak farksal evrim algoritmasının aksine F ve CR değerlerinin geniş bir aralıkta olmasının aksine Len_1 ve Len_2 değerleri [0,05;0,25] gibi bir aralıkta olduğunda daha iyi sonuçlar elde edilebilmektedir.

Keskintürk çalışmasında, (2006a) GSP yi çözmek için farksal gelişim algoritması kullanmıştır. Çalışmada şehir sayısı olarak 5, 10, 20, 40 ve 50 li olmak üzere 5 test probleminde deneme yapılmıştır. Problemin parametreleri için ek bir çalışma yapılmamış olup literatür deki değerler kullanılmıştır. Her bir problem 100 kez çalıştırılmış ve ortalama tur uzunlukları ve ortalama yüzde sapmalar hesaplanmıştır.
Sonuç olarak, farksal evrim algoritmasının küçük ölçekli problemlerde optimale yakın sonuçlar verdiği ancak büyük problemlerde iyi sonuçlar vermediği saptanmıştır. Bunun için çalışmada öneri olarak, algoritmada arama sürecini etkinleştirmek için değişiklik yapılması gerektiği öne sürülmüştür. Ayrıca Keskintürk, parametre seçiminin bir en iyileme yöntemi ile yapılması çalışmanın daha iyi sonuç verebileceğini belirtmiştir.

Keskintürk (2006b) yaptığı başka bir çalışmada doğrusal olmayan en iyileme problemlerinin çözümünde farksal evrim algoritması ve genetik algoritmayı karşılaştırmıştır. Problem parametrelerinin doğru seçilmesi durumunda farksal evrim algoritmasının sürekli değerlerin kullanıldığı eniyileme problemlerinde genetik algoritmadan daha iyi sonuçlar verdiğini öne sürmüştür.

Karaboğa ve Ökdem (2004) çalışmalarında, her ikisi de popülasyon temelli algoritmalar olan farksal evrim algoritması ile genetik algoritmanın performanslarını kıyaslamıştır. Çalışmada De Jong'un test fonksiyonları kullanılmıştır. Kontrol parametrelerini: popülasyon büyüklüğünü küçük ölçekli problemler için 10, büyük ölçekli problemler için 20, ölçeklendirme faktörünü [-2,2] aralığında rassal olacak

şekilde, çaprazlama oranının 0,8, yeniden üretme katsayısını 0,5 olarak belirlenmiştir.

Yapılan analizler ile farksal evrim algoritmasının yakınsama hızının genetik algoritmanın yakınsama hızından daha iyi olduğu, ayrıca farksal evrim algoritmasının en iyi sonuca daha az üretme ile ulaştığı sonucu ortaya çıkmıştır.

1.3.2. Parçacık Sürü En İyileme Algoritması ile ilgili Literatür Araması

Wang ve diğerleri (2003) çalışmalarında GSP yi çözmek için parçacık sürü eniyileme yöntemi kullanmışlardır. Yöntemde PSE algoritmasına *yer değiştirme* işlemi (swap operator) ve *yer değiştirme* sıralaması (swap sequence) ekleyerek 14 şehirli problem çözülmeye çalışılmıştır. Çalışmanın sonucunda küçük ölçekli problemlerde iyi sonuç alındığı yani en iyiye (optimale) yakınsamanın hızlı olduğu ve en iyi sonuca ulaşılabilindiği saptanmıştır. Algoritmanın iyi çalışması daha sonra yapılacak çalışmalar için tetikleme yapmıştır.

Shi ve diğerleri (2007) çalışmalarında PSE ile GSP ve Genelleştirilmiş GSP (GGSP) çözümü ile ilgili çalışmışlardır. Çalışmada PSE algoritması biraz değiştirilerek probleme uygulanmıştır. Örneğin çözülebilir problem sayısını artırmak için yer değiştirme işleminden faydalanılmış ve permütasyon kavramı algoritmaya eklemişlerdir. Ayrıca yakınsama hızını artırmak için belirsizlik arama stratejisi eklenmiştir algoritmaya. Algoritma problem üzerinde 100 kez denenmiştir. Algoritmanın 50-80 şehirli problemler için çalıştırıldığında iyi sonuçlar verdiği gözlemlenmiştir.

Wang ve diğerleri (2005) çalışmalarında PSE algoritmasını yeniden tasarlamışlardır. Yer değiştirme işlemi ve yer değiştirme sıralamasından farklı olarak ayarlama işlemi (adjustment operator) ve ayarlama sıralaması (adjustment sequence) eklenmiştir algoritmaya. Ayrıca daha önce parçacık sürü eniyilemesi algoritmasında adı geçmeyen mutasyon işlemi de eklenmiştir algoritmaya. Algoritmanın etkinliğini ölçmek için 14 şehirli ve 29 şehirli iki problem çözülmüştür. Sonuç olarak algoritmanın üstün başarısı gözlemlenmiştir.

Shi ve diğerleri (2004) çalışmalarında kesikli PSE metodunu, genelleştirilmiş GSP (GGSP) problemi çözümünde kullanmışlardır. Problemin yakınsama hızını artırmak için de yerel arama yöntemi kullanmışlardır. PSE sürekli problemleri çözmek için tasarlanan bir yöntem olduğu için GGSP çözümünde birtakım değişikliğe uğratılmıştır. Bunun için Kennedy ve Eberhart'ın öne sürdüğü parçacıkların konum ve hızları ile tanımlandıkları kesikli iki terimli PSE versiyon (discerete binary PSO version) yöntemi kullanılmıştır. Yönteme göre, π_{id} nin 1 olma olasılığının v_{id} olduğu parça [0,1] aralığında hareket edebilir sadece. Yöntem GSP kütüphanesinden alınan 21 test problemi ile test edilmiştir. Deneyler sonucunda, önerilen yöntemin GGSP'nde kullanılmasının etkin olduğu gözlemlenmiştir. En büyük nısbi hata oranı % 9,76 olduğu ortalama nısbi hatanın %2,55 olduğu belirlenmiştir. Karşılaştırma yapılan diğer çalışmalara göre oldukça iyi sonuçlar vermiştir.

BÖLÜM 2

2. GEZGİN SATICI PROBLEMİ VE ÇÖZÜM YÖNTEMLERİ

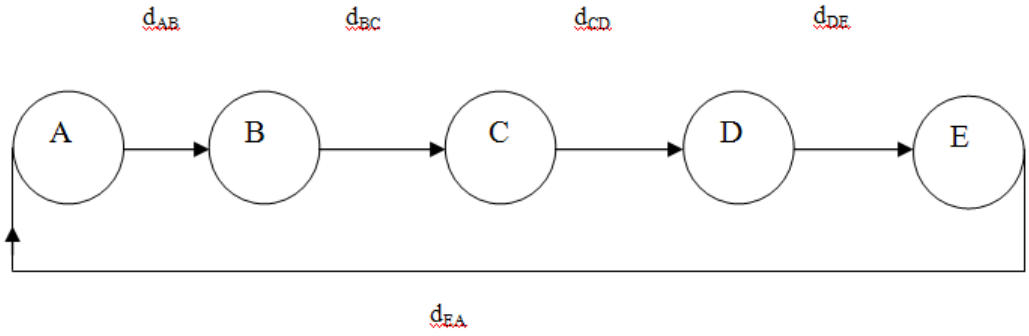
2.1. Gezgin Satıcı Problemi

Gezgin satıcı problemi zor kombinasyonel en iyileme problemlerinin tipik örneğidir. Problemde *i.* ve *j.* şehir arasındaki mesafenin d_{ij} olduğu *n* tane şehir bulunmaktadır ve amaç gezgin satıcının bu şehirlere en fazla bir kez uğrayarak şehirler arasındaki en kısa turu oluşturmasını ve başlangıç noktasına geri dönmesini sağlamaktır.

Şehirler düzlemdeki konumları ile ifade edilir, örneğin şehir 1; (x_1,y_1), şehir 2, (x_1,y_2) şeklinde ifade edilmektedir. Şehirlerarasındaki mesafe $d_{ij,}$ öklit mesafesi yani kuşbakışı mesafe olarak varsayılır, denklem 2.1 ile hesaplanır.

$$d_{ij}= \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{2.1}$$

Toplam mesafe, şehirlerarasındaki mesafeler denklem 2.1 ile hesaplanıp birbirleri üzerine eklenerek bulunmaktadır. Örneğin satıcı A şehrinden başlarsa ve turundaki şehirler B,C,D,E şeklindeyse toplam mesafe Şekil 2.1 deki gibi şehirlerin mesafelerinin tek tek eklenmesi ile hesaplanmaktadır (Erentürk, 2003).



Şekil 2.1 Şehirlerin Tur Gösterimi

Problem de şehir sayısı *n* ile gösterilmektedir. Satıcının ziyaret ettiği şehirlerin sıralamasına tur adı verilmektedir. Turda her şehir en az bir kez ziyaret edilir ve başlangıç şehrine tekrar dönülür. Problemin amacı en az maliyeti sağlayan ya da en

kısa turu sağlayan şehir sıralamasını belirlemektir. Bu amaç doğrultusunda problemin matematiksel ifade edilişi şu şekildedir:

$$\min z(\pi) = \sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)} \tag{2.2}$$

$$\sum_{j=1}^{N} \pi_{ij} = 1; \forall_i \tag{2.3}$$

$$\sum_{i=1}^{N} \pi_{ij} = 1; \forall_j \tag{2.4}$$

$$\pi_{ij} = 0 \text{ or } 1 \tag{2.5}$$

1 numaralı denklem problemin amaç fonksiyonudur. Amaç en kısa turu bulmak olduğu için problem en küçükleme problemi olarak tasarlanmıştır. d_{ij}, değişkeni *i* şehrinden *j* şehrine gitme maliyetini ya da uzaklığı ifade etmektedir. Denklem 2.2, i şehrinden *j* şehrine gidildiğinde toplam maliyetin en küçüklendiğini ifade etmektedir. Denklem 2.3 problemin ilk kısıtıdır. Her şehrin mutlaka bir kez terk edildiğini ifade etmektedir. Denklem 2.4 problemin 2. kısıtıdır. Bu kısıt da her şehrin mutlaka bir kez ziyaret edildiğini ifade etmektedir. Denklem 2.5 ise problemin son kısıtıdır. Son kısıt π_{ij} nin 0 ya da 1 olduğunu yani ziyaret edilme ya da edilmemeyi ifade etmektedir (Taşgetiren v.d., 2009a).

Eğer, $1<i, j>n'i$ sağlayan $d_{ij}=d_{ji}$ koşulu var ise problem simetrik GSP olarak nitelendirilir. Koşul, *i* şehrinden *j* şehrine olan mesafe ile *j* şehrinden *i* şehrine olan mesafenin eşit olduğunu belirtmektedir. Koşulun sağlanmadığı durum yani, $d_{ij} \neq d_{ji}$ durumu da *ij* uzaklığı ile *ji* uzaklığının eşit olmadığını belirtmektedir. Bu tip GSP problemi de asimetrik GSP olarak nitelendirilir (Sauer ve Coelho, 2008).

Problemde gezgin satıcı başlangıçta *n* tane değişik şehir arasından seçim yapıp turuna başlamaktadır. İlk şehirde ise *n-1* tane şehir arasından seçim yaparak turuna devam etmektedir. İkinci şehirde ise *n-2* tane şehir arasından seçim yapmaktadır. Yani satıcı için *n!* tane değişik tur seçeneği mevcuttur. Problem 10 şehirli ise satıcı 10!= 3628800 değişik tur arasından seçim yapacaktır, 50 şehirli ise 50!= $3{,}04*10^{64}$ değişik tur arasından seçim yapacaktır, 100 şehirli ise 100!=$9{,}33*10^{157}$ değişik tur arasından seçim yapacaktır. Görüldüğü üzere problem şehir sayısı arttıkça çözümü zorlaşmaktadır. Sayım tekniği ile her olası turun tek tek incelenip, en az maliyeti

sağlayan ya da en kısa turu sağlayan en iyi (optimal) turun polinom zamanda bulunması imkansıza yakındır. Bu nedenle gezgin satıcı problemi çok olasılıklı NP-zor problemlere örnek teşkil etmektedir. Problemin çözümünde, NP doğasından dolayı sadece yaklaşık sonuçlar beklenmektedir (Özsağlam, 2009).

GSP matematikçiler ve istatistikçiler arasında "Messenger Problemi" ismi ile bilinmekte idi. Hakkında ilk yazılı çalışma Menger tarafından yapılmıştır. Menger çalışmasında sayım (enumeration) yöntemi ile en iyi çözüme ulaşılabileceğini fakat en yakın komşu (nearest neighbour) yönteminin en iyi çözümü garanti etmediğini öne sürmüştür (Taşgetiren v.d., 2009b).

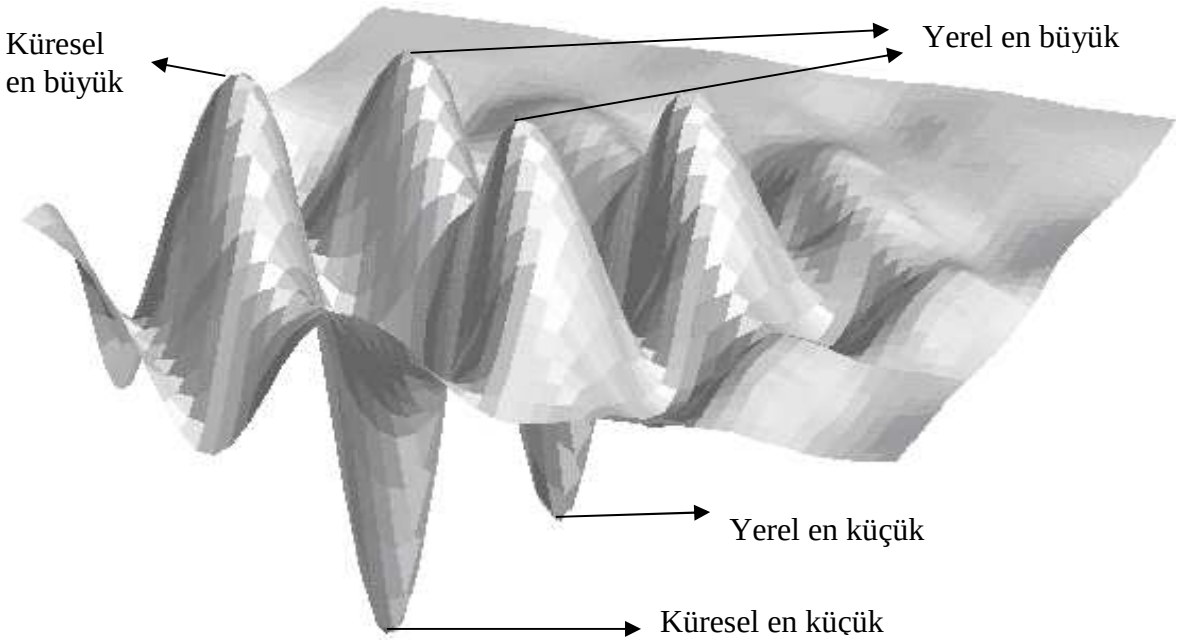
Araç rotalama, iş planlama, bütünleşik devre tasarımı, çizelgeleme, esnek imalat sistemleri, fiziksel haritalama problemleri, X-ray kristallografi gibi çeşitli problemler GSP olarak formüle edilebilir (Sauer ve Coelho, 2008). Ayrıca, şehir boru hattı döşeme en iyilemesi, malzeme akış endüstrisinde araç rotalama problemleri de GSP olarak formüle edilebilmektedir (Wang v.d., 2005).

Gerçek yaşam problemlerinin pek çoğu GSP veya GSP çeşitleri olarak formüle edilebilmektedir. GSP'nin çeşitleri aşağıda verilmiştir:
MAX GSP (The MAX TSP)
Darboğaz GSP (The Bottleneck TSP)
Çok ziyaretli GSP (TSP with multiple visits)
Taşıyıcı problemi (Messenger problem)
Kümelenmiş GSP(Clustered TSP)
Genelleştirilmiş GSP (Generalized TSP)
Çok satıcılı GSP (The m-salesman TSP)

GSP çözümü için kullanılan yöntemler kesin çözüm yöntemleri, sezgisel yöntemler ve modern sezgisel yöntemler olmak üzere 3 e ayrılır. Dinamik Programlama (Dynamic Programming), Tam Sayı Programlama (Integer Programming), Dal ve Sınır Yöntemi (Branch and Bound), Dal ve Kesme (Branch and Cut) yöntemleri kesin çözüm yöntemleri olarak tanımlanmaktadır.

Son 20 yılda ise GSP çözümü için, 2-opt yöntemi, 3-opt yöntemi, benzetimli tavlama (simulated annealing) yöntemi, yapay sinir ağları (artificial neural network) yöntemi, yasak araması (tabu search) yöntemi gibi pek çok yöntem kullanılmıştır. Bu yöntemler de kesin çözümleri değil, en iyi çözüme (optimale) yakın çözümleri vermektedir. Son zamanlarda ise GSP de genetik algoritma, farksal evrim algoritmaları başarı ile uygulanmaktadır. (Sauer ve Coelho, 2008).

2.1.1. En İyileme (Optimizasyon)



Şekil 2.2 Birkaç En büyük ve En küçük Değerli Fonksiyon

En iyileme bir dizi çözüm arasından en iyiyi bulma işlemidir yani istenen özelliği artırırken istenmeyen özellikleri azalma işlemidir. Küresel (global) ve yerel (local) olmak üzere 2 tür en iyileme işlemi vardır. Küresel en iyileme tüm çözüm kümesinden küresel en iyi olanı seçer, yerel en iyileme ise çözüm kümesinden birbirlerine yakın çözümler arasından en iyiyi seçer. Yerel en iyileme de başlangıç noktası önemlidir, başlangıç noktasına göre elde edilen çözüm değişiklik gösterir. Küresel en iyilemede ise başlangıç noktası önemli değildir hep aynı sonuç bulunur ancak bunun için daha fazla zamana gereksinim duyulmaktadır. Küresel en iyiyi bulmak neredeyse imkansızdır.

Yavaş bir şekilde küresel en iyiyi (best) bulmaktansa hızlı şekilde yerel en iyiyi (good) bulmak zaman açısından tercih edilebilmektedir. Örneğin $f(x)=x^2$ fonksiyonunda, fonksiyon değerini en küçük (min) yapan değerin x=0 noktası olduğunu bulmak oldukça kolaydır ancak Şekil 2.2 deki gibi daha karmaşık fonksiyonlarda sonuca bu kadar çabuk ulaşmak zorlaşmaktadır. Şekil 2.2 birden fazla yerel en büyük ve yerel en küçük göstermektedir (Erentürk, 2003).

2.1.1.1. En İyileme Teknikleri

Günümüzde pek çok bilim dalı eniyileme problemleri ile ilgilenmektedir. Problemler doğrusal ya da doğrusal olmayan eniyileme problemi olarak modellenebilir ancak gerçek hayat problemleri genellikle doğrusal olmayan nitelikdedir. Bu tip problemleri mantıksal zaman çerçevesi dahilinde çözmek zordur. Problemlerin zorluk derecesi, problemdeki değişken sayısına ve veri tipine bağlı olarak değişmektedir. Örneğin, GSP de satıcının *n!* kadar değişik tur arasından seçim yapması gerekmektedir. Bulunan en güçlü algoritma n^2*2^n zamanda çözüm yapabilmektedir. Bu da 100 şehirli bir problemin $1{,}26*10^{30}$ adımda çözülmesi anlamına gelmektedir. Bu durumda GSP yi, NP-zor olarak nitelendirmektedir (Eberhart ve Shi, 2001).

Çözüm zamanının çok uzun olmasından dolayı bu tip problemler sezgisel yöntemlerle çözülebilmektedir. Özellikle genetik algoritma, bulanık mantık, karınca kolonisi algoritması ve benzetimli tavlama gibi popülasyon temelli sezgisel yöntemler çok noktalı arama yöntemi ile hızlı sonuç verebilmektedir.

Kombinasyonel problemlerin çözümünde son yıllarda sezgisel yöntemlerin kullanımı büyük oranda artmıştır. Artışın sebebi sezgisel yöntemlerin pek çok avantajının olmasıdır. Avantajları: hesaplama gücünün iyi olması, dönüştürülebilir olması, ayrıca sayım tekniğine (enumeration) göre çözüm zamanının çok daha kısa olmasıdır. Bu avantajlarının yanı sıra optimum çözümü garanti etmeme ve iyi çözüm elde edebilmek için problem parametrelerinin iyi ayarlanması gerekliliği şeklinde dezavantajları vardır (Jones ve diğerleri, 2002).

2.2. Gezgin Satıcı Probleminin Çözümünde Kullanılan Modern Sezgisel Yöntemler

Bilinen eniyileme tekniklerinin fayda sağlayamadığı durumlar olduğundan evrimsel algoritmalar geliştirilmiştir. Evrimsel algoritmalar aşağıdaki 5 maddeye gereksinim duymaktadır:

a. Çözümün genetik temsili,
b. Başlangıç popülasyonunun oluşturulması
c. Uygunluk fonksiyonunun belirleme
d. Hesaplama işlemlerini (operatörleri) belirleme
e. Kontrol parametrelerini belirleme (popülasyon büyüklüğü, operatör uygulama oranları, ölçekleme faktörü gibi)

Genetik algoritmalar, evrimsel programlama, evrimsel stratejiler ve farksal evrim algoritması, evrimsel algoritmalarıdır..

Evrimsel hesaplama teknikleri birbirlerine benzemektedir ancak birbirinden farklılık da göstermektedir. Örneğin çözümlerin temsili farklı şekilde yapılabilmektedir ve buna bağlı olarak farklı operatörler kullanılmaktadır.

Ancak farklı evrimsel algoritmalarda adımların sıralamasında değişiklik olabilmektedir. Örneğin evrimsel stratejilerde önce popülasyon değiştirilir sonra yeni popülasyon seçme işlemi ile oluşturulur. Aynı algoritmada dahi birden fazla işlem seçeneği mevcuttur. Örneğin farksal evrim algoritmasında bireylerin seçiminde kullanılacak çok sayıda seçme yöntemi mevcuttur (Koyuncu, 2006).

2.2.1. Farksal Evrim Algoritması (FEA)

Sürekli uzayda küresel eniyileme içeren problemler bilimsel toplumda oldukça yaygındır. Genellikle amaç sistem parametrelerini belirleyerek belirli özellikleri eniyilemektir. Sistem parametreleri vektör olarak ifade edilmektedir. Bir eniyileme problemine standart yaklaşım amaç fonksiyonunu tasarlamaktır ve bu genellikle en küçükleme şeklinde olabilir. En küçüklenmek istenen maliyet, zaman veya mesafe olabilir. Amaç fonksiyonu doğrusal olmayan ya da türevlenemez (non-differentiable)

olduğu durumda doğrudan arama yöntemleri tercih edilir ki bunlar genetik algoritma ve evrimsel stratejiler şeklinde olabilir.

Genellikle en küçükleme tekniklerinden bazı gerekçeleri yerine getirmesi beklenir. Bu gerekçeler;

a. Doğrusal olmayan, türevlenemez (non-differentiable) ve çoklu maliyet fonksiyonlarını çözebilme,
b. Hesaplama yoğun maliyet fonksiyonlarını çözmek için paralelleştirebilme, (örneğin bir maliyet fonksiyonunun sadece bir kez bile değerlendirilmesi çok fazla zaman alabilir. Değerlendirilmeler aynı özelliğe sahip farklı bilgisayarlarda yapılabilecek şekilde paralelleştirme yapılabilmeli)
c. Kullanım kolaylığı,
d. İyi yakınsama özelliği.

Şeklinde sıralanabilir. Bu gerekçeleri yerine getirebilme için farksal evrim algoritması geliştirilmiştir. Farksal evrim algoritması, özellikle sürekli problemlerin çözümünde kullanılan bir modern sezgisel en iyileme yöntemidir, ilk kez Storn ve Price tarafından tasarlanmıştır. Diğer evrimsel algoritmalar gibi popülasyon temelli stokastik bir algoritmadır. FE algoritmasında genetik algoritmada olduğu gibi mutasyon, çaprazlama ve seçim işlemleri vardır ancak işlemlerin uygulama sıraları farklı şekildedir (Storn ve Price, 1997). Algoritmanın genel işleyişi Şekil 2.3 de gösterilmiştir:

```
Parametreleri belirle
Başlangıç popülasyonu oluştur
Popülasyonu değerlendir
do {
        Mutasyon uygula
        Çaprazlama uygula
        Değerlendir
        Seçim
} while(Sonlandirma kosulu)
```

Şekil 2.3 KFE Algoritmasının İşleyişi

Algoritma olası çözümleri kapsayan başlangıç (hedef) popülasyonunu oluşturmakla başlar. Başlangıç popülasyonun da her bir birey *D* boyutlu bir vektördür. Algoritmada *D* boyutlu her bir vektör yani her bir birey, kromozom olarak ifade

edilmektedir. Algoritmanın sonraki adımlarında mutasyon, çaprazlama ve seçim işlemleri gerçekleşmektedir. Algoritmada mutant çözüm elde etmek için başlangıç popülasyonundan rassal seçilmiş 2 olası çözümün yani bireyin ağırlıklandırılmış farkı 3. bir bireye eklenerek mutasyon işlemi gerçekleştirilir. Ardından mutant çözüm ile hedef çözüm çaprazlanarak deneme (trial) çözüm elde edilir. Çaprazlama işleminin ardından bir sonraki nesle kimlerin devam edeceğini belirlemek için deneme çözüm ile hedef çözümün uygunluk değerleri (amaç fonksiyonları) arasında kıyas yapılır ve en iyi olan seçilir (Taşgetiren v.d., 2009a).

2.2.1.1. Algoritmanın Temel Elemanları

NP: Popülasyon büyüklüğü (kromozom sayısı), $NP \geq 4$

n: Gen sayısı (eniyilenecek değişken sayısı) $j = 1,2,..,n$

CR: Çaprazlama oranı, $CR \in (0,1)$. Popülasyonun bir sonraki nesille olan farklılığını etkileyen sabit bir sayıdır. Genellikle 0 ile 1 arasında bir sayıdır.

k: Nesil indeksi

F: Mutasyon faktörü, $F \in (0,2)$. Ölçeklendirme faktörü olarak da adlandırılan mutasyon faktörü: 0 ile 2 arasında değişiklik gösteren gerçek bir sayıdır. İki birey arasındaki farklılığın çeşidini etkileyen bir faktördür.

π_{ij}^k: Hedef birey: *k.* nesildeki *j* boyutlu *i*. bireyi ifade etmektedir. $\pi_i^k = \left[\pi_{i1}^k, \pi_{i2}^k, ..., \pi_{in}^k\right]$, *(j=1,2,..,n)* şeklinde tanımlanır.

v_{ij}^k: Mutant birey: *k.* nesildeki *j* boyutlu mutasyona uğramış *i.* bireyi ifade etmektedir.

$v_i^k = \left[v_{i1}^k, v_{i2}^k, ..., v_{in}^k\right]$, *(j=1,2,..,n)* şeklinde tanımlanır.

u_{ij}^k: Deneme birey: *k.* nesildeki *j* boyutlu *i*. deneme bireyini ifade etmektedir.

$u_i^k = \left[u_{i1}^k, u_{i2}^k, ..., u_{in}^k\right]$, *(j=1,2,..,n)* şeklinde tanımlanır.

π^k : Hedef popülasyon : *k.* nesildeki hedef popülasyondaki *NP* tane birey dizisini ifade etmektedir. $\pi^k = \left[\pi_1^k, \pi_2^k, ..., \pi_{NP}^k\right]$ şeklinde tanımlanır.

v^k*:* Mutant popülasyon: *k.* nesildeki mutant popülasyondaki *NP* tane birey dizisini ifade etmektedir. $v^k = \left[v_1^k, v_2^k, ..., v_{NP}^k\right]$ şeklinde tanımlanır.

u^k: Deneme popülasyon: *k*. nesildeki deneme popülasyonundaki *NP* tane birey dizisini ifade etmektedir. $u^k = \left[u_1^k, u_2^k, ..., u_{NP}^k \right]$ şeklinde tanımlanır.

$f_i(\pi_i^k)$*:*Uygunluk (amaç) fonksiyonu: En küçükleme problemlerinde π_i^k bireyi için hedef fonksiyonudur. (Taşgetiren v.d., 2009a).

Başlangıç Popülasyonu

Algoritma *NP* tane *D* boyutlu vektör oluşturarak başlangıç popülasyonunu oluşturmakla başlar çözüme. Başlangıç popülasyonu oluşturulmadan önce alt ve üst limitler belirlenir. Limitler belirlendikten sonra rassal sayı üretici bir mekanizma her vektörün her parametresine bir değer atar. Örneğin başlangıç neslinde, *i* vektörlü *j* parametreli başlangıç değeri denklem 2.6 ile hesaplanır.

$$\pi_{i,j}^0 = rand_j(0,1).(b_{j,U}-b_{j,L})+b_{j,L} \quad (2.6)$$

L ve *U* indisleri başlangıç parametre sınırlarını ifade etmektedir.

Rand j(0,1), 0 ile 1 arasında uniform şekilde dağıtılmış rassal değişkeni temsil etmektedir.

Vektörün üssünün 0 olması, vektörün ait olduğu neslin başlangıç nesli olduğunu ifade etmektedir (Taşgetiren v.d., 2009a).

Mutasyon İşlemi

Algoritma başlangıç popülasyonunu oluşturduktan sonra yine *NP* büyüklüğünde bir deneme popülasyonu oluşturmak için popülasyonu mutasyona uğratır ve çaprazlar. Mutasyon mevcut popülasyonu değişikliğe uğratmak anlamına gelmektedir. Bu şekilde kromozomun temsil ettiği çözüm, çözüm uzayında hareket etmektedir. Geleneksel farksal evrim yönteminde algoritma rassal seçilmiş iki vektörün farkını ağırlıklandırıp yine rassal seçilmiş üçüncü bir vektöre ekleyerek mutasyon işlemini gerçekleştirir. Yani mutant vektör üç farklı vektörün işleme alınması ile oluşmuş yeni bir vektördür. Bu durumda popülasyon büyüklüğünün en az 3 olması gerektiğini belirtebiliriz (Price ve Storn, 2005). Ancak geleneksel FEA sürekli en iyileme problemler için tasarlanmıştır. Kesikli problemlerin çözümü için yeni bir FEA olan kesikli farksal evrim algoritması (KFEA) geliştirilmiştir. Algoritmada hedef birey $\pi_i^k = \left[\pi_{i1}^k, \pi_{i2}^k, .., \pi_{in}^k \right]$ gibi bir çözüm ile temsil edilir. Başlangıç popülasyonu rassal

olarak n şehir için permütasyonlardan NP kadar birey oluşturulur. Mutant birey hedef popülasyondaki neslinin en iyi çözümünün bozulması (perturbasyona uğraması) ile elde edilir. Farksal çeşitliliğe hedef popülasyondaki en iyi çözümün bozulmalarıyla ulaşılabilmektedir. Bozulmalar stokastik yani rassal şekilde yapılmaktadır, öyle ki mutant popülasyondaki her bir bireyin ayırıcı olması beklenmektedir. Mutant birey elde etmek için denklem 2.7 kullanılmaktadır.

$$v_i^k = \begin{cases} BozYap_{d,p}\left(\pi_g^{k-1}\right) & eger \quad r < P_m \\ Ekleme\ \left(\pi_g^{k-1}\right) & aksi \quad takdide \end{cases} \tag{2.7}$$

π_g^{k-1}, bir önceki neslin hedef popülasyonundaki en iyi çözümü,

$P_{m,}$ bozulma olasılığını,

$BozYap_{d,p}$, bozma(destruction) ölçüsünün d olduğu, perturbasyon (perturbation) derecesinin p olduğu bozulma işlemini,

Ekleme, basit bir rassal ekleme hareketini ifade etmektedir.

[0,1] aralığında r rassal sayısı üretilir. Eğer bu sayı P_m den küçük ise mutant birey oluşturmak için *Boz-Yap* işlemi uygulanır, $v_i^k = BozYap_{d,p}\left(\pi_g^{k-1}\right)$ denklemi kullanılır. Aksi taktirde, bir önceki neslin en iyi çözümü rassal bir ekleme işlemi ile pertubre edilir, $v_i^k = Ekleme(\pi_g^{k-1})$ denklemi kullanılır.

Denklem 2.7, $v_i^k = P_m \oplus BozYap_{d,p}\left(\pi_g^{k-1}\right) \quad i:=1,2,\ldots,NP$ (2.8)

olarak ifade edilir. Ve sonuç olarak mutant popülasyon denklem 2.8 ile elde edilmiş olur. (Taşgetiren v.d., 2009a).

Çaprazlama İşlemi

Mutasyon işleminden sonra algoritma çaprazlama uygular. Çaprazlama, mutasyon sonucu elde edilen mutant vektör ile hedef vektör kullanılarak yeni bir vektör olan deneme vektörü elde etme işlemidir.

$$u_i^k = \begin{cases} CR\left(v_i^k, \pi_i^{k-1}\right) & eğer \quad r < P_c \\ v_i^k & aksi \quad takdirde \end{cases} \tag{2.9}$$

Deneme vektör, *CR* nin çaprazlama işlemi, P_c nin, çaprazlama olasılığı anlamına geldiği denklem 2.9 kullanılarak hesaplanır. Başka bir deyişle, uniform şekilde üretilmiş *r* rassal sayısının P_c çaprazlama olasılığından küçük olması durumunda deneme bireyi, $u_i^k = CR(v_i^k, \pi_i^{k-1})$ elde etmek için çaprazlama işlemi uygulanmaktadır. Aksi takdirde, deneme bireyi, $u_i^k = v_i^k$ olarak belirlenmektedir. Tüm bu işlemler yapıldıktan sonra deneme bireyi ya perturbasyon işlemi sonucunda ya da çaprazlama işleminden elde edilmiş olmaktadır.

Denklem 2.9

$$u_i^k := P_c \oplus CR(v_i^k, \pi_i^{k-1}) \quad i:=1,2,\ldots,NP \tag{2.10}$$

şeklinde gösterilir (Taşgetiren v.d., 2009a).

Seçim İşlemi

Sonuç olarak uygunluğun devamı için deneme ve hedef bireyler arasında seçim işlemi yapılır. Seçim işlemi denklem 2.11 ile hesaplanır.

$$\pi_i^k = \begin{cases} u_i^k & eğer \quad f\left(u_i^k\right) < f\left(\pi_i^{k-1}\right) \\ \pi_i^{k-1} & aksi \quad takdirde \end{cases} \tag{2.11}$$

Denklem 2.11, $\pi_i^k = arg\ min\left\{f(\pi_i^k),\ f(\pi_i^{k-1})\right\} \quad i:=1,2,\ldots NP$ (2.12)

şeklinde gösterilir.

Durdurma Koşulu

Nesil sayısı, belirlenen sınırı aşarsa ya da optimal sonuca ulaşılamazsa belirlenen bir tekrarda algoritma durdurulur. İstenilen durdurma koşulu sağlanıncaya kadar işlemler tekrar edilir.(Taşgetiren v.d., 2009a).

Algoritmanın detaylı işleyişi Tablo 2.4 deki gösterim (pseudo) kodunda gösterilmiştir.

Pr*ocedure KFE _ TSP*

$\pi = \left(\pi_1^0, \pi_2^0, ..., \pi_{NP}^0\right)$

$\pi = NEH\left(\pi\right)$

$f\left(\pi_i^0\right)$
$i{:}1,2,...,NP$

$\pi_i^0 = 2\_opt(\pi_i^0)$
$i=1,2,...,NP$

$\pi_i^0 = LS(\pi_i^0)$
$i=1,2,...,NP$

$\pi_G^0 = \arg\min\left\{ f(\pi_i^0)\right\}$
$i=1,2,...,NP$

$k := 1$

while(*Not sonlandirma kosulu*) do

$v_i^k := P_m \oplus BozYap_{d,p}\left(\pi_g^{k-1}\right)$
$i=1,..,NP$

$u_i^k := P_c \oplus CR\left(v_i^k, \pi_i^{k-1}\right)$
$i=1,..,NP$

$u_i^k = 2\_opt(u_i^k)$
$i=1,2,...,NP$

$u_i^k = LS(u_i^k)$
$i=1,2,...,NP$

$\pi_i^k = \arg\min\left\{ f\left(\pi_i^{k-1}\right), f\left(u_i^k\right)\right\}$
$i=1,2,...,NP$

$\pi_g^k = \arg\min\left\{ f\left(\pi_i^k\right), f\left(\pi_g^{k-1}\right)\right\}$
$i=1,2,...,NP$

$k = k+1$

endwhile

returnπ_B

Şekil 2.4 KFE Algoritmasının Gösterim (pseudo) Kodu

2.2.1.2. Notasyon

Farksal evrim algoritmasının notasyonu *DE/x/y/z* şeklinde gösterilmektedir. Notasyondaki *x*, mutant vektörü ifade etmektedir. *x*'in in rand olması mutant vektör oluşturulurken kullanılacak bireylerin popülasyondan rassal olarak seçildiğini best olması ise mutant vektör oluşturulurken kullanılacak bireylerin popülasyondan uygunluk fonksiyonuna göre en iyisinin seçildiğini gösterir.

y, ağırlıklandırılmış fark vektörlerinden kaç tane kullanıldığını belirtir, 1 ya da 2 olarak değişiklik gösterebilir.

z, ise çaprazlama türü hakkında bilgi vermektedir, (bin) ya da (exp) olabilir. z'nin 'bin' olması çaprazlama mekanizmasının bağımsız binom dağılımı ile kontrol edildiğini ifade etmektedir.

Klasik notasyon *DE/rand/1/bin* şeklinde gösterilmektedir. Diğer stratejilerin notasyonları da şu şekildedir:

DE/best/1/bin	$u_i^k = \pi_{best}^{k-1} + F(\pi_{r1}^{k-1} - \pi_{r2}^{k-1})$
DE/rand/1/bin	$u_i^k = \pi_{r1}^{k-1} + F(\pi_{r2}^{k-1} - \pi_{r3}^{k-1})$
DE/rand-to-best/1/bin	$u_i^k = \pi_i^{k-1} + \lambda(\pi_{best}^{k-1} - \pi_{r1}^{k-1} -) + F(\pi_{r1}^{k-1} - \pi_{r2}^{k-1})$
DE/rand/2/bin	$u_i^k = \pi_5^{k-1} + F(\pi_{r1}^{k-1} - \pi_{r2}^{k-1} - \pi_{r3}^{k-1} - \pi_{r4}^{k-1})$
DE/best/2/bin	$u_i^k = \pi_{best}^{k-1} + F(\pi_{r1}^{k-1} - \pi_{r2}^{k-1} - \pi_{r3}^{k-1} - \pi_{r4}^{k-1})$

(r_i : rassal sayı $i=1,2,3,\ldots n$)

(Davendra ve Onwubolu, 2009)

2.2.2. Parçacık Sürü En İyileme Algoritması (PSE)

Parçacık sürü eniyilemesi ilk kez Kennedy ve Eberhart tarafından tasarlanmıştır. Genetik algoritma ya da farksal evrim algoritması gibi popülasyon tabanlıdır, balık ve kuş gibi sürü davranışlarından esinlenmektedir. Her denemede en iyi çözümü kullanarak daha iyiyi bulmaya çalıştığı için genetik algoritmaya göre daha zeki davranan bir algoritmadır. PSE bireyler arasındaki sosyal bilgi paylaşımını esas alır. PSE kavramı ilk başta sosyal bir sistemin benzetimi (simülasyonu) olarak ortaya çıkmıştır. İlk baştaki amaç bir kuş sürüsünün tahmin edilmesi zor koreografisinin benzetimini grafiksel olarak yapmaktı fakat algoritmanın evriminde bazı noktalarda kavramsal modelin aslında bir eniyileyici (optimizer) olduğu anlaşılmıştır. PSE de genetik algoritma gibi rassal çözümlerin yer aldığı bir popülasyonun oluşturulması ile sistem çalışmaya başlar ancak GA dan farkı başlangıçta her çözüme atanmış rassal bir hız değerine sahip olmasıdır (Eberhart ve Shi, 2001). Daha önce popülasyon olarak tanımlanan olası çözümler kümesi PSE algoritmasında sürü olarak ifade edilirken, birey ya da kromozom diye adlandırılan her olası çözüm parçacık olarak adlandırılır ve her parçacığın ayrı bir hızı vardır. Her nesil hızını popülasyondaki en iyi parçacığın hızı ile karşılaştırarak oluşturur böylece her nesil

bir öncekinden daha iyi konuma gelmiş olur bu da algoritmanın daha kısa zamanda en iyiye (optimale) yaklaşmasını sağlar (Özsağlam, 2009).

Parçacıkların her birinin, içinde bulundukları nesile gelene kadar ulaşabildikleri en iyi konumu o parçacığın kişisel en iyisi (personal best) olarak belirlenir ve p_{best} olarak ifade edilir. Popülasyondaki parçacıklardan içinde bulunduğu nesilde en iyi uygunluk değerine sahip olanı, popülasyonun en iyisi yani küresel en iyi (global best) olarak belirlenir ve g_{best} olarak ifade edilir. (Eberhart ve Shi, 2001).

PSE sürekli eniyileme problemleri için geliştirilmiştir ancak kesikli eniyileme problemleri için de kullanılabilmektedir. Ancak algoritma kesikli problemlere doğrudan uygulanmaz, öncelikle birtakım işlemlere tabi tutularak kesikli probleme uyarlanır (Shi v.d., 2007). Daha iyi çözüm yeni bir nesil üretilmeye çalışarak elde edilmektedir ve her nesil kendinden bir önceki nesil kullanılarak üretilmektedir. Bu adımlarda parçacık sürü eniyileme yönteminin farksal evrim algoritması yöntemi gibi evrimsel bir algoritma olduğunu göstermektedir (Wang v.d., 2003).

PSE, diğer evrimsel algoritmalar gibi pek çok en iyileme problemlerine uygulanabilmektedir. Algoritma, sistem tasarımı, çok amaçlı en iyileme, sınıflandırma, biyolojik sistem modelleme, çizelgeleme, işaret işleme, robot uygulamaları, karar verme, benzetim gibi uygulama alanlarında bulanık kontrolör tasarımı, atölye tipi çizelgeleme, robot güzergahı planlama, görüntü bölümlendirme, zaman frekans analizi, otomatik hedef belirleme gibi pek çok problemin çözümünde kullanılmaktadır (Eberhart ve Shi, 2001). Ayrıca algoritma, sistem tanımlama, sipariş miktarı belirleme (lot sizing) problemi (Tasgetiren ve Liang, 2003), sinir ağları (neural network) eğitimi (Van den Bergh ve Engelbecht, 2000), akış tipi çizelgeleme problemleri (Tasgetiren vd., 2004a; 2004b), güç ve voltaj kontrolü (Yoshida vd., 2000; Abido, 2002), tek makine toplam pozitif gecikme problemi (Tasgetiren vd., 2004c), tedarikçi seçimi ve sıralama problemlerinin (Yeh, 2003) ve iş atama problemlerinin (Salman vd., 2003) eniyilemesinde etkin şekilde kullanılmaktadır.

Algoritmanın adımları; popülasyonun oluşturulması, uygunluk değerinin hesaplanması, en iyi parçacığın bulunması, küresel en iyinin bulunması, konum ve

hızların yenilenmesi ve durdurma kriteri sağlanıncaya kadar devam etme şeklinde sıralanabilir (Özsağlam, 2009).

Orijinal olarak sürekli fonksiyonların en iyilenmesi için tasarlanan PSE de parçacık kendi hızına, konumuna ve uygunluk (amaç fonksiyonu) değerine sahiptir. Parçacıklar, *n*-boyutlu bir uzayda sürü popülasyonunun bireylerin kişisel en iyi ve küresel eniyi parçacıklarından yararlanarak hız yeniden hesaplanır ve hız parçacığa eklenir. Dolayısıyla algoritma kişisel ve küresel en iyi tarafından yönlendirilir ve bu yüzden hep daha iyiye yaklaşırlar.

NP, sürü büyüklüğüdür ve parçacık $\pi^k = \lfloor \pi_1^k, \pi_2^k, ...\pi_{NP}^k \rfloor$ şeklinde tanımlanır. Popülasyon sürüsündeki her parçacık şu özelliklere sahiptir:

$\pi_i^k = \lfloor \pi_{i1}^k, \pi_{i2}^k, ...\pi_{in}^k \rfloor$ şeklinde tanımlanan mevcut konum.

$V_i^k = \lfloor v_{i1}^k, v_{i2}^k, ...v_{in}^k \rfloor$ şeklinde tanımlanan mevcut hız.

$P_i^k = \lfloor p_{i1}^k, p_{i2}^k, ...p_{in}^k \rfloor$ şeklinde tanımlanan mevcut kişisel en iyi konum.

$G_i^k = \lfloor g_1^k, g_2^k, ...g_n^k \rfloor$ şeklinde tanımlanan mevcut küresel en iyi konum.

f fonksiyonunun en küçükleme olduğunu varsayarsak, *j* boyutlu *i* parçacığının hızı denklem 2.13 ile güncellenir.

$$v_{ij}^k = w^{k-1} v_{ij}^{k-1} + c_1 r_1 (p_{ij}^{k-1} - \pi_{ij}^{k-1}) + c_2 r_2 (g_j^{k-1} - \pi_{ij}^{k-1}) \quad (2.13)$$

Denklemdeki w^k; önceki hızın mevcut hıza etkisini kontrol eden bir parametredir,

c_1, c_2; ivme katsayılarıdır,

r_1, r_2; [0,1] aralığında uniform rassal sayılardır.

Önceki konum ve hız kullanılarak güncellenen, *j* boyutlu *i* parçacığının mevcut konumu denklem 2.14 ile hesaplanır.

$$\pi_{ij}^k = \pi_{ij}^{k-1} + v_{ij}^{k-1} \quad (2.14)$$

Parçacığın kişisel en iyi konumu denklem 2.15 ile belirlenir.

$$P_i^k = \begin{cases} P_i^{k-1} \ if \ f(\pi_i^k) \geq f(P_i^{k-1}) \\ \pi_i^k \ if \ f(\pi_i^k) < f(P_i^{k-1}) \end{cases} \quad (2.15)$$

Sürü popülasyonundaki küresel en iyi konum da denklem 2.16 ile belirlenir.

$$G^k = \begin{cases} \arg\min_{P_i^k} f(P_i^k) \; if \; \min f(P_i^k) < f(G^{k-1}) \\ G^{k-1} else \end{cases} \quad 1 \le i \le NP. \tag{2.16}$$

Konumlar gerçek değerli olduğu sürece, kesikli iş permütasyonlarını (job permutation) oluşturmak için standart PSE eşitlikleri kullanılamaz. Bunun yerine kesikli iş permutasyonlarına dayalı parçacıklar için Pan, Taşgetiren ve Liang (2008) yeni bir konum güncelleme yöntemi önermiştir. Parçacığın davranışı 3 farklı olası seçim arasında uzlaşmaktadır. Bunlar;

a. Parçacık kendi konumunu takip edebilir. (π_i^k)
b. Parçacık kişisel en iyi konumunu takip edebilir. (P_i^k)
c. Parçacık tüm sürü popülasyonunun en iyi konumunu takip edebilir. (G^k)

k. nesilde parçacığın konumu denklem 2.17 ile güncellenir.

$$\pi_i^k = c_2 \otimes F_2(c_1 \otimes F_1(w \otimes BozYap(\pi_i^{k-1}), P_i^{k-1}), G^{k-1}) \tag{2.17}$$

Güncellenen bu eşitlik 3 bileşen içermektedir.

a. $\lambda_i^k = w \otimes BozYap(\pi_i^{k-1})$ bileşeni, parçacığın hızını temsil etmektedir. Bileşendeki *BozYap*, *w* olasılığı ile *ekleme* mutasyon işlemcisini temsil eder. Başka bir deyişle, [0,1] arasında rassal uniform bir sayı üretilir. Eğer üretilen bu '*r*' rassal sayısı *w* olasılığından küçük ise parçacığın perturbasyon edilmiş permutasyonunu üretmek için mutasyon işlemcisi $\lambda_i^k = BozYap(\pi_i^{k-1})$ uygulanır. Aksi taktirde $\lambda_i^k = \pi_i^{k-1}$ uygulanır.
b. $\delta_i^k = c_1 \otimes F_1(\lambda_i^k, P_i^{k-1})$ bileşeni, parçacığın bilişsel kısmını temsil etmektedir. Bileşendeki F_1, c_1 olasılığı işe çaprazlama işlemcisini temsil etmektedir. Yani perturbasyana uğramış parçacık λ_i^k ve kişisel eniyi P_i^{k-1} çaprazlanması amaçlanmaktadır. Başka bir deyişle, [0,1] arasında rassal uniform bir sayı üretilir. Eğer üretilen bu '*r*' rassal sayısı c_1 olasılığından küçük ise, bileşen çaprazlanarak $\delta_i^k = F_1(\lambda_i^k, P_i^{k-1})$ şeklinde; aksi halde $\lambda_i^k = \delta_i^k$ şeklinde sonuçlanır.

c. $\pi_i^k = c_2 \otimes F_2(\delta_i^k, G^k)$ bileşeni, parçacığın sosyal kısmını yani parçacıklar arasındaki uyumu temsil etmektedir. F_2, c_2 olasılığı ile çaprazlama işlemcisini temsil etmektedir. Başka bir deyişle, [0,1] arasında rassal uniform bir sayı üretilir. Eğer üretilen bu '*r*' rassal sayısı c_2 olasılığından küçük ise, bileşen çaprazlanarak $\pi_i^k = F_2(\delta_i^k, G^{k-1})$ şeklinde; aksi halde $\pi_i^k = \delta_i^k$ şeklinde sonuçlanır.

Konum güncelleme eşitliğinde yeni bir çaprazlama işlemcisi olan PTL (Pan ve diğerleri, 2008) kullanılmıştır. PTL çaprazlama işlemcisi hakkında detaylı bilgi bölüm 2.2.6 da verilmiştir (Pan, Taşgetiren, Liang, 2008). Algoritmanın genel işleyişi Şekil 2.5 te verilmiştir. Ayrıca detaylı işleyişi Şekil 2.6 daki gösterim (pseudo) kodunda verilmiştir.

DPSO{
Parametreleri belirle
Popülasyonu oluştur
Popülasyonu değerlendir
Do{
Kişisel en iyiyi bul
Küresel en iyiyi bul
Konumu güncelle
Değerlendir
}while (sonlandırma koşulu)

Şekil 2.5 PSE Algoritmasının İşleyişi

$\Pr ocedure \;\; PSE\_TSP$

$\pi = \left(\pi_1^0, \pi_2^0, ..., \pi_{NP}^0\right)$

$\pi = NEH\left(\pi\right)$

$f\left(\pi_i^0\right)$
$i{:}1,2,...,NP$

$\pi_i^0 = 2\_opt(\pi_i^0)$
$i=1,2,...,NP$

$\pi_i^0 = LS(\pi_i^0)$
$i=1,2,...,NP$

$\pi_B = \arg\min\left\{ f(\pi_i^0)\right\}$
$i=1,2,...,NP$

$k := 1$

while(*Not sonlandirma kosulu*) do

$\lambda_i^k := w \oplus BozYap_{d,p}\left(\pi_i^{k-1}\right)$
$i=1,..,NP$

$\delta_i^k := c_1 \oplus F_1\left(\lambda_i^k, P_i^{k-1}\right)$
$i=1,..,NP$

$\pi_i^k := c_2 \oplus F_2\left(\delta_i^k, G^{k-1}\right)$
$i=1,..,NP$

$\pi_i^k = 2opt(\pi_i^k)$
$i=1,2,...,NP$

$\pi_i^k = LS(\pi_i^k)$
$i=1,2,...,NP$

$\pi_i^k = \arg\min\left\{ f\left(\pi_i^{k-1}\right), f\left(\pi_i^k\right)\right\}$
$i=1,2,...,NP$

$\pi_g^k = \arg\min\left\{ f\left(\pi_i^k\right), f\left(\pi_g^{k-1}\right)\right\}$
$i=1,2,...,NP$

$\pi_B = \arg\min\left\{ f\left(\pi_B\right), f\left(\pi_g^k\right)\right\}$

$k = k+1$

endwhile

returnπ_B

Şekil 2.6 PSE Algoritmasının Gösterim (pseudo) Kodu

2.2.3. NEH Sezgiseli

Bu bölümde yerel arama sezgisellerinden NEH sezgiseli açıklanmıştır. Sezgiselin adımları şu şekilde belirtilmiştir.

1. Şehirlerden başlangıç turu (π) elde etmek.
2. Olası turun ilk iki şehiri (π_1, π_2) alınır ve bu iki şehirin olası turu değerlendirilir ve iyi olan seçilir. Ancak tur oluşturulurken başlangıç şehiri ile bitmesine dikkat edilmelidir. Örneğin, (π_1, π_2, π_1) ya da (π_2, π_1, π_2) gibi.
3. Yukarıdaki adımlar, başlangıç turundaki tüm şehirlerin ilk iki şehirin oluşturduğu tura eklenmesi bitene kadar tekrarlanır.
Örnek:
Mevcut çözüm π= {3,1,4,2,5}olsun
İlk 2 şehirin olası turlarının {3,1,3} ve{1,3,1}değerlendirilmesi. İlk turun uygunluk fonksiyonunun daha iyi olduğunu varsayalım. Bu durumda mevcut kısmi tur {3,1} şeklinde olacaktır.
Eklemeler:
3. sıradaki şehir yani 4 şehirinin mevcut kısmi turdaki 3 olası konuma eklenmesi: {4,3,1,4},{3,4,1,3}ve{3,1,4,3} şeklinde olmaktadır. En iyi uygunluk değerini veren turun {3,4,1,3}olduğunu varsayarsak, mevcut kısmi tur {3,4,1} olarak belirlenir.

4. sıradaki şehir yani 2şehirinin mevcut kısmi turdaki 4 olası konuma eklenmesi: {2,3,4,1,2}, {3,2,4,1,3}, {3,4,2,1,3} ve {3,4,1,2,3} şeklinde olmaktadır. En iyi uygunluk değerini veren turun {3,2,4,1,3}olduğunu varsayarsak, mevcut kısmi tur {3,2,4,1} olarak belirlenir.

Son olarak 5. sıradaki şehir yani 5 şehirinin mevcut kısmi turdaki 5 olası konuma eklenmesi: {5,3,2,4,1,5}, {3,5,2,4,1,3}, {3,2,5,4,1,3}, {3,2,4,5,1,3} ve {3,2,4,1,5,3} şeklinde olmaktadır. En iyi uygunluk değerini veren turun {3,2,4,5,1,3}olduğunu varsayarsak, mevcut kısmi tur {3,2,4,5,1} olarak belirlenir. Son tam tur π = {3,2,4,5,1} şeklindedir (Taşgetiren v.d., 2009a).

2.2.4. Ekleme Sezgiseli (Insertion)

Bu bölümde yerel arama sezgisellerinden (düğüm/şehir) ekleme yöntemi açıklanmıştır. Düğüm olarak adlandırılan değişken problemde şehri temsil etmektedir. Yöntemde n tane düğümü yani şehri olan tur parçası π^D ile ifade edilmektedir. π_k^R değişkeni de herhangi bir düğümü yani şehri ifade etmektedir.

Yöntem turdan çıkartılan π_k^R şehrinin π^D turunda *n-1* tane olası konuma yerleştirilmesi ve problemin amaç fonksiyonunun sağlanması ile gerçekleştirilir. Ekleme turun başına, sonuna veya herhangi iki şehir arasına olmak üzere 3 farklı şekilde yapılabilmektedir. (Taşgetiren v.d., 2009b). Problemin kısmi çözümü Tablo 2.1 deki gibi olsun:

Tablo 2.1 Problemin Kısmi Çözümü

J	1	2	3	4	5	6	7	8	9	10
π_j^D	5	2	9	4	1	3	8	6	10	7
$d_{\pi_j \pi_{j+1}}$	$d_{5,2}$	$d_{2,9}$	$d_{9,4}$	$d_{4,1}$	$d_{1,3}$	$d_{3,8}$	$d_{8,6}$	$d_{6,10}$	$d_{10,7}$	$d_{7,5}$

a. Şehir ($x_k^R = 11$)in , kısmi turun en başına eklenmesi:

Çıkart= $d_{\pi_n^D \, \pi_1^D}$

Ekle= $d_{\pi_k^R \, \pi_1^D} + d_{\pi_n^D \, \pi_k^R}$

$F(\pi) = F(\pi^D) + Ekle - Çıkart$ (2.18)

$F(\pi)$ ve $F(\pi^D)$, tur ve kısmi turun ekleme yapıldıktan sonraki uygunluk fonksiyonlarıdır. Turun ekleme işleminden sonraki uygunluk fonksiyonu (değeri) denklem 2.18 ile hesaplanır.

A örneği:

Çıkart= $d_{\pi_n^D \, \pi_1^D}$

Çıkart= $d_{\pi_{10}^D \, \pi_1^D}$

Çıkart= $d_{7,5}$

Ekle= $d_{\pi_k^R \, \pi_1^D} + d_{\pi_n^D \, \pi_k^R}$

Ekle= $d_{\pi_1^R \, \pi_1^D} + d_{\pi_{10}^D \, \pi_k^R}$

Ekle= $d_{11,5} + d_{7,11}$

$F(\pi) = F(\pi^D) + ekle - çıkart$

$F(\pi) = d_{5,2} + d_{2,9} + d_{9,4} + d_{4,1} + d_{1,3} + d_{3,8} + d_{8,6} + d_{6,10} + d_{10,7} + \cancel{d_{7,5}} + d_{11,5} + d_{7,11} - \cancel{d_{7,5}}$

$F(\pi) = d_{5,2} + d_{2,9} + d_{9,4} + d_{4,1} + d_{1,3} + d_{3,8} + d_{8,6} + d_{6,10} + d_{10,7} + d_{11,5} + d_{7,11}$

$\pi_k^R = 11$şehrinin kısmi turun ilk konumuna eklenmesi Tablo 2.2 de gösterilmiştir.

Tablo 2.2 $\pi_k^R = 11$ Şehrinin Kısmi Çözümün İlk Konumuna Eklenmesi

J	1	2	3	4	5	6	7	8	9	10	11
π_j^D	**11**	5	2	9	4	1	3	8	6	10	7
$d_{\pi_j \pi_{j+1}}$	$d_{11,5}$	$d_{5,2}$	$d_{2,9}$	$d_{9,4}$	$d_{4,1}$	$d_{1,3}$	$d_{3,8}$	$d_{8,6}$	$d_{6,10}$	$d_{10,7}$	$d_{7,11}$

b. Şehir ($\pi_k^R = 11$)in, kısmi turun en sonuna eklenmesi:

Çıkart= $d_{\pi_n^D \pi_1^D}$

Ekle= $d_{\pi_n^D \pi_k^R} + d_{\pi_k^R \pi_1^D}$

F (π)= F (π^D) + Ekle- Çıkart

F (π) ve *F (π^D)*, tur ve kısmi turun ekleme yapıldıktan sonraki uygunluk fonksiyonlarıdır.

B örneği:

Çıkart= $d_{\pi_n^D \pi_1^D}$

Çıkart= $d_{\pi_{10}^D \pi_1^D}$

Çıkart= $d_{7,5}$

Ekle= $d_{\pi_n^D \pi_k^R} + d_{\pi_k^R \pi_1^D}$

Ekle= $d_{\pi_{10}^D \pi_1^R} + d_{\pi_1^R \pi_1^D}$

Ekle= $d_{7,11} + d_{11,5}$

F (π)= F (π^D) + ekle- çıkart

F(π)= $d_{5,2}+ d_{2,9}+d_{9,4}+d_{4,1}+ d_{1,3}+d_{3,8}+d_{8,6}+d_{6,10}+d_{10,7}+\cancel{d_{7,5}}+ d_{7,11} + d_{11,5} - \cancel{d_{7,5}}$

F(π)= $d_{5,2}+ d_{2,9}+d_{9,4}+d_{4,1}+ d_{1,3}+d_{3,8}+d_{8,6}+d_{6,10}+d_{10,7} + d_{7,11}+ d_{11,5}$

$\pi_k^R = 11$şehrinin kısmi turun son konumuna eklenmesi Tablo 2.3 de gösterilmiştir.

Tablo 2.3 $\pi_k^R = 11$ Şehrinin Kısmi Çözümün Son Konumuna Eklenmesi

J	1	2	3	4	5	6	7	8	9	10	11
π_j^D	5	2	9	4	1	3	8	6	10	7	**11**
$d_{\pi_j \pi_{j+1}}$	$d_{5,2}$	$d_{2,9}$	$d_{9,4}$	$d_{4,1}$	$d_{1,3}$	$d_{3,8}$	$d_{8,6}$	$d_{6,10}$	$d_{10,7}$	$d_{7,11}$	$d_{11,5}$

c. Şehir ($\pi_k^R = 11$)in, kısmi turda rassal seçilen iki şehrin arasına eklenmesi:

Çıkart= $d_{\pi_u^D \; \pi_v^D}$

Ekle= $d_{\pi_u^D \; \pi_k^R} + d_{\pi_k^R \; \pi_v^D}$

F (π)= F (π^D) +Ekle- Çıkart

F (π) ve *F (π^D),* tur ve kısmi turun ekleme yapıldıktan sonraki uygunluk fonksiyonlarıdır.

C örneği:

u = 6

v = 7

Çıkart= $d_{\pi_n^D \; \pi_1^D}$

Çıkart= $d_{\pi_6^D \; \pi_7^D}$

Çıkart= $d_{3,8}$

Ekle= $d_{\pi_u^D \; \pi_k^R} + d_{\pi_k^R \; \pi_v^D}$

Ekle= $d_{3,11} + d_{11,8}$

F (π)= F (π^D) + ekle- çıkart

F(π)= $d_{5,2} + d_{2,9} + d_{9,4} + d_{4,1} + d_{1,3} + \cancel{d_{3,8}} + d_{8,6} + d_{6,10} + d_{10,7} + d_{7,5} + d_{3,11} + d_{11,8} - \cancel{d_{3,8}}$

F(π)= $d_{5,2} + d_{2,9} + d_{9,4} + d_{4,1} + d_{1,3} + d_{8,6} + d_{6,10} + d_{10,7} + d_{7,5} + d_{3,11} + d_{11,8}$

$\pi_k^R = 11$şehrinin kısmi turun rassal seçilen iki şehri arasında eklenmesi Tablo 2.4 de gösterilmiştir.

Tablo 2.4 $\pi_k^R = 11$ Şehrinin Kısmi Çözümün Rassal Seçilen İki Şehri (6 ve 7) Arasına Eklenmesi

J	1	2	3	4	5	6	**7**	8	9	10	11
π_j^D	5	2	9	4	1	3	**11**	8	6	10	7
$d_{\pi_j \; \pi_{j+1}}$	$d_{5,2}$	$d_{2,9}$	$d_{9,4}$	$d_{4,1}$	$d_{1,3}$	$d_{3,11}$	$\boldsymbol{d_{11,8}}$	$d_{8,6}$	$d_{6,10}$	$d_{10,7}$	$d_{7,5}$

2.2.5. Boz-Yap (Destruction Construction) Sezgiseli

Kesikli farksal evrim algoritmasında, tekrarlı açgözlü (iterated greedy) algoritmasında kullanılan boz-yap prosedürü uygulanmaktadır. Bozma (destruction) adımında;

d kadar şehir rassal olarak belirlenip çözümden çıkartılır yani mevcut çözüm yıkıma uğratılır. Çıkartılan şehirler yeni bir kısmi tur, geride kalan şehirlerde farklı bir kısmi tur oluşturmuş olur. İlk tur, mevcut turdan çıkartılan şehirlerin çıkartılma sırasıyla oluşturduğu *d* şehirli π^R turudur. 2. tur, çıkan şehirlerden geriye kalan *(n-d)* kadar şehirin oluşturduğu π^D turudur. Bozma fazında perturbasyon işlemi gömülü olarak çalışır. Perturbasyon işleminde de π^R den *p* kadar şehir seçilir ve kısmi turdaki diğer şehirler ile yer değiştirilir.

Yapım (construction) adımında;

Yapıcı bir sezgisel prosedür olan NEH kullanılmaktadır. π^R yi yeniden π^D ye eklemek için π^R deki ilk şehir olan π_1^R , π^D deki *(n-d+1)* olası konuma eklenir ve *(n-d+1)* tane olası çözüm elde edilir. Bu *(n-d+1)* tane olası çözümden x_1^R şehrini içeren en kısa turu sağlayan çözüm alınır ve π^R deki bir sonraki şehir olan π_2^R bu çözüme eklenerek yine *n* büyüklüğündeki π^D turu elde edilmiş olur.

Örnek:

Mevcut çözüm π= {3,1,4,2,5} olsun. Mevcut çözümden *d= 2* tane şehir rassal olarak belirlenip çıkartılarak π^R = {1,5} kısmi turu, geriye kalan şehirler ile de π^D = {3,4,2} kısmi turu oluşturulmuştur.

Yapım aşamasında turdan çıkartılan şehirlerden ilki x_1^R =1, NEH sezgiselineye göre, π^D deki 4 olası konuma {1,3,4,2},{3,1,4,2},{3,4,1,2},{3,4,2,1}şeklinde yerleştirilir. Elde edilen 4 olası çözümden en kısa turu sağlayan çözüm belirlenip π^R deki 2. Şehir olan π_2^R = 5, turun yine olası konumlarına eklenerek uygunluk fonksiyonunu sağlayan tur saptanır. Elde edilen çözümlerden en iyisinin {3,4,2,1} olduğunu varsayarsak, π_2^R = 5 i de olası konumlara ekleyerek {5,3,4,2,1}, {3,5,4,2,1}, {3,4,5,2,1}, {3,4,2,5,1} ve {3,4,2,1,5}turlarını elde ederiz ve en kısa turu sağlayan çözümü olası turlar arasından seçeriz. (Taşgetiren v.d., 2009a).

2.2.6. PTL Çaprazlama İşlemcisi

PTL, Pan, Taşgetiren, Liang'ın kısaltmasıdır. Kesikli farksal evrim algoritmasında kullanılan '2 noktadan kesmeli' (2-cut) PTL çaprazlama işlemcisi Pan tarafından geliştirilmiştir. Özdeş iki ebeveyn bireyden yani özdeş 2 çözümden 2 farklı çocuk yani çözüm oluşturmak için kullanılmaktadır.

PTL çaprazlama işleminde, şehirler ebeveyn bireylerden 2 tane rassal noktadan kesilerek blok halinde belirlenmektedir. Belirlenen bu blok yeni bireyin sağ ya da sol tarafına taşınmaktadır daha sonra yeni birey 2. ebeveyn bireydeki geriye kalan şehirler ile doldurularak tamamlanmaktadır. Şekil 2.7 de görüldüğü üzere, bu işlemler özdeş ebeveyn bireylerden bile farklı 2 yeni birey oluşturmak için kullanılmaktadır.

Ebeveyn birey 1)	5	1	**4**	**2**	3
Ebeveyn birey 2)	5	1	**4**	**2**	3
Yeni birey 1)	**4**	**2**	5	1	3
Yeni birey 2)	5	1	3	**4**	**2**

Şekil 2.7 2 noktadan kesmeli PTL çaprazlama işlemi

Oluşturulan yeni bireylerden eşit olasılıkla seçim yapılmaktadır. (Taşgetiren, Liang v.d., 2009).

2.2.7. Farksal Evrim Algoritması ve Parçacık Sürü Algoritmasının Yerel Arama Sezgiseli ile Birleştirilmesi

Algoritmaların yerel arama sezgiseli ile birleştirilmesi, üretilen her deneme bireye yerel arama uygulaması yapılarak gerçekleştirilmektedir. Yerel arama sezgiseli olarak ekleme (insertion) işlemi uygulanmıştır ve her deneme bireye 2-opt işlemi uygulanmaktadır. Ekleme prosedürü şekil 2.8 de gösterilmiştir.

Pr*ocedureInsert*()
j şehrini T turundan cikar
Herbir k için

$$c^k \leftarrow \min\{d_{uk} + d_{kv} - d_{uv} / (u,v)\text{ , T turunda bir edge}\}$$

$$k^* \leftarrow \arg\min_{k \in v_j}$$

k^* şehrini T turundaki (u,v) sehirleri arasina eklenir.

Şekil 2.8 Ekleme İşleminin İşleyişi

Ekleme işleminde turdan bir şehir çıkarılır ve *n-1* pozisyona eklenir. Bu eklemelerden en iyisi uygunluk değerlerine bakılarak seçilir. Bu ekleme işlemi şekil 2.9 deki yerel arama prosedüründe n defa tekrarlanır. (Tasgetiren, Suganthan, Pan, 2010). Yerel arama sezgisel yönteminin işleyişi şekil 2.9 de verilmiştir.

procedureLS(π)
  $h := 1$
  while($h \le n$) *do*
    $\pi^* = Insert(\pi)$
    $f((\pi^*) \le f(\pi))$ then
      $\pi := \pi^*$
      $h := 1$
    else
      $h := h + 1$
    else
  endwhile
  returnπ
endprocedure

Şekil 2.9 Yerel Arama Sezgiselinin İşleyişi

2-opt işlemi, turda rassal seçilen i ve j şehirleri arasındaki mesafe ile yine rassal olarak seçilen k ve m şehirleri arasındaki mesafenin toplamı i-k mesafesi ile j-m mesafesi toplamından büyük ise j ve k şehirlerinin yer değiştirilmesi ile gerçekleştirilir.

2-opt un işleyiş şekli:

Mevcut çözüm π= {3,1,4,5,2,6,8,10,9} olsun,

i j k m

d_{ij} d_{km}

Eğer $(d_{ij}+d_{km})> (d_{ik}+d_{jm})$ ise j, k şehirlerinin yeri değiştirilir.

$d_{4,5}+d_{68} > d_{4,6}+d_{5,8}$ olduğunu varsayarsak;

Yeni çözüm π= {3,1,4,6,2,5,8,10,9} şeklinde olur.

BÖLÜM 3

3. GSP PARAMETRELERİNİN BELİRLENMESİNDE KULLANILAN DENEY TASARIMI YÖNTEMİ

Bir sistem veya sistemin bir kısmına ait ayrıntıları incelemek için yapılan faaliyetlere deney denilmektedir. Deney terimi istatistik biliminde ise yeni gerçekler meydana getirmek veya var olanı doğrulamak ya da çürütmek için kullanılan bir iş olarak tanımlanmaktadır. Deneyler kesin ve karşılaştırmalı deneyler ve tek ve çok etmenli deneyler olmak üzere iki farklı şekilde gerçekleştirilebilmektedir.

Kesin ve karşılaştırmalı deneyler, bilinmeyeni ortaya çıkarmak aynı zamanda bilineni de doğrulamak ya da çürütmek için yapılmaktadır.

Tek ve çok etmenli deneyler ise bir veya daha fazla etmenin durum ya da işlem üzerindeki etkisini belirlemek için yapılan deney türüdür ve mevcut duruma özel olarak tasarlanır.

Deney tasarımı ise, bir işlemdeki girdilerde sistematik değişiklikler yapılarak girdilerin sonuç üzerindeki etkilerini gözlemlemek için kullanılan bir tekniktir.

Deney tasarımının amaçları kısaca aşağıdaki şekilde sıralanabilir.

-Sonuç üzerinde en etkin girdi faktörlerini belirlemek.

-Performans karakteristiğini hedef değere en yakın sonuca ulaştıracak girdi faktörlerinin seviyelerini belirlemek.

-Kontrol edilemeyen girdi faktörlerinin çıktı üzerindeki etkisinin en az olacağı kontrol edilebilen girdi faktörleri kombinasyonunu oluşturmak.

Deney tasarımında, süreci etkileyen her türlü etken “faktör” olarak adlandırılır. Faktörlerin çeşitli kategorileri, dereceleri ve yoğunlukları ise “seviye“ olarak adlandırılır. Bir deney tasarımı sırasında birden fazla faktör ve seviye grupları bulunabilir (Şanyılmaz, 2006).

Denemenin planlama ve yürütülme şekli denemenin stratejisini belirlemektedir. 4 çeşit deneme stratejisi vardır, bunlar: en iyi tahmin, bir defada bir faktör, faktöriyel tasarım ve kısmi faktöriyel tasarımdır.

En iyi tahmin yaklaşımında sonuç üzerinde önemli etkisi olduğu düşünülen faktörlerin rassal bir bileşimi ele alınır ve test edilir. Ancak bu yöntemin dezavantajları vardır. Bunlardan birincisi, önemli etkisi olduğu düşünülen faktörlerin bileşimi iyi sonuç vermeyebilir. Bu durumda denemeci yeni bir bileşim oluşturmak zorunda kalır. Bu durum en iyi bileşimi elde etme garantisi olmadan devam eder. Bu da fazla zaman kaybına sebep olabilir ya da ikinci bir sakınca denemeci oluşturduğu bileşimin iyi sonuç verdiğini görünce test etmeyi bırakır. Bu durumda da daha iyi sonuçlardan vazgeçilmiş olunur.

Bir defada bir faktör yaklaşımında her bir faktör başlangıç seviyesinde, diğer faktörlerin sabit tutularak onların üzerine dağıtılması ile test edilir. Her faktör ile sabit tutulan faktörlerin değiştirilmesi ile sonucun nasıl etkilendiği gözlemlenir. Yaklaşımın dezavantajı her bir faktör tek tek incelendiğinden faktörlerin birbiri ile etkileşiminin gözlemlenemiyor olmasıdır. Etkileşim, bir faktörün diğer faktöre ait farklı seviyelerde aynı etkiyi yaratmamasıdır. Etkileşimi olan faktörlerin bulunduğu testlerde bu yaklaşımın kullanılması denemeciyi doğru sonuca ulaştıramayabilir.

Üçüncü bir yaklaşım olan faktöriyel deneme, faktörlerin tek tek değil hep birlikte ele alınması ve birbirleri ile etkileşiminin de ele alınması sebebi ile daha güvenilir bir yaklaşımdır. Faktör sayısı arttıkça deneme sayısı da artmaktadır bu da test süresinin uzaması şeklinde bir dezavantaj sağlamaktadır. Zaten faktör seviyelerinin bütün bileşimlerini kullanmak gereksiz olabilmektedir. Böyle durumlarda ise kısmi faktöriyel deneme yaklaşımı kullanılmaktadır. Kısmi faktöriyel deneme yaklaşımında faktörler içerisinde altkümelere ayrılır ve sonra işlem görür. Bu şekilde zamandan tasarruf edilir (Karakoç, 2006).

3.1. Deney Tasarım İlkeleri

Denemeciyi doğru sonuca götürecek deneylerin öncelikle iyi tasarlanması gereklidir. Deneylerin iyi tasarlanması için Fisher tarafından geliştirilmiş tekrarlama, rassallık ve bloklama ilkelerine bağlı kalınmalıdır (Şanyılmaz, 2006).

Tekrarlama

Tekrarlama deneyin birden fazla kere yapılması demektir. Aynı faktörün tekrar tekrar ölçülmesi değil esas denemenin tekrarlanması anlamında kullanılmaktadır. Tekrarlama, değişkenliğe sebep olan faktörlerin ortaya çıkardığı hataların ölçümü için kullanılmaktadır. Tekrarlama olmadan değişkenliğe sebep olan unsurlar belirlenememektedir.

Tekrarlamanın iki önemli özelliği vardır. Birincisi, deneysel hataları giderme imkanı vermesidir, ikincisi ise eğer örnek kümenin ortalaması herhangi bir faktörün etkisinin tahmin edilmesinde kullanılıyor ise sonucun daha kesin olmasını sağlamasıdır.

Rassallık

Rassallık, deneysel materyallerin herhangi bir kurala uygun olmaması, tesadüfi olması demektir. Deneysel verilerin toplanmasının da rassallık ilkesine bağlı olarak yapılması sonucun daha gerçeği yansıtmasının en önemli gerekçesidir. Deney sonuçlarını analiz etmede de rassallık kullanılacak yöntemler için gereklidir. Rassallık önyargıyı giderici bir etkidir.

Bloklama

Bloklama, deneyin hassasiyetini artıran bir ilkedir. Deneme yapılacak alanın homojen olacak şekilde veya belirli farklı özelliklere göre bölümlere ayrılması şeklinde yapılır.

3.2. Deney Tasarımının Tarihçesi

Deney tasarımı ilk olarak 1920'lerde ünlü İngiliz istatistikçi Sir A. Ronald Fisher tarafından geliştirilmiştir. Fisher, yöntemi tarım alanında ürünlerin verimliliğini artırmak amacı ile birtakım çalışmalar yaparak geliştirmiştir. Fisher ürün verimliliğini artırmak için verimliliği aynı olacak şekilde ekim yapılacak toprağı parsellere ayırmıştır. Ayırdığı parsellere ürünleri ve gübre çeşitlerini rassal olarak dağıtarak ekim yapılmasını sağlamıştır. Fisher' ın yaptığı bu çalışma deney tasarımının temellerini oluşturmaktadır. Deney tasarımı 1950'li yıllara kadar kimya

ve ilaç endüstrisinde de kullanılmıştır. 1950'li yıllardan sonra deney tasarımının, toplam kalite yöntemi ile ilgili çalışmalarda başarı göstermesi yöntemin daha yaygınlaşmasına sebep olmuştur. 1970'li yıllarda yaşanan sanayi devriminden sonra yöntem üretim sektöründe de yaygın olarak kullanılmaya başlanmıştır.

Motorola firmasının kalite grubundan K. Bhote'nin "World Class Quality" adlı kitabında kalite geliştirme de üç yöntemin çok önemli olduğunu belirtmiştir. Bunlar, geleneksel muayene, istatistiksel işlem kontrol ve deney tasarım yöntemleridir. Japonya'da kalite kavramının gelişmesinde en büyük katkıyı deney tasarımı sağlamıştır. Amerika'da ise 1980'lerden sonra deney tasarımı yöntemi gelişmeye başlamıştır (Şanyılmaz, 2006).

3.3. Deney Tasarımın Kullanım Alanları

Deney tasarımı yöntemi pek çok alanda kullanılabilmektedir. Özellikle ürün ya da işlem tasarımındaki problemleri çözmede ve üretim aşamasındaki problemleri çözmede kullanılmaktadır. Örneğin,

Tasarım bileşenlerinin değerlendirilmesi ve karşılaştırılmasında,

Malzeme seçiminde alternatiflerin değerlendirilmesinde

Tasarım parametrelerinin seçiminde

Ürün performansını belirleyici parametrelerin seçilmesinde deney tasarımı yöntemi etkin şekilde kullanılabilmektedir (Şanyılmaz, 2006).

3.4. Deney Tasarımının Adımları

Problemin tanımlanması

Faktörlerin, seviyelerin ve sıranın seçimi

Tepki değişkeninin seçimi

Deney tasarımının seçimi

Deneyi uygulama

Verilerin istatistiksel analizi

Sonuçlar (Şanyılmaz, 2006).

3.4.1. Problemin Tanımlanması

Problemin tanımlanması kolay gibi görünse de aslında tasarımın en önemli adımlarından birisidir. Tasarımdan en iyi sonucu alabilmek için problemi iyi belirlemek gerekmekte ve görünürde birden fazla problem olsa da temel problemi saptamak gerekmektedir. Bunun için konu ile ilgili veriler sağlıklı şekilde tüm birimlerden toplanmalı ve takım yaklaşımı ile problemin saptanması için beyin fırtınası yapılmalıdır.

3.4.2. Faktörlerin, Seviyelerin ve Sıranın Seçimi

Tasarımın ilk aşaması olan problem belirlemeden sonra tasarım sonucunu etkileyen faktörler ve bu faktörlere ait seviyeler belirlenmelidir. Faktörler ve seviyeler belirlenirken daha önce yapılmış çalışmalardaki saptamalardan faydalanılmalıdır.

3.4.3. Tepki Değişkeninin Seçimi

Belirlenen problem doğrultusunda, faktör ve seviyeleri kullanılarak yapılan tasarım alternatifleri denenir ve bu denemelerin sonucunda doğru tasarımın belirlenmesinde kullanılacak performans kriteri saptanır. Bu kriter ölçülen karakteristiğin ortalaması veya standart sapması ya da her ikisi birden olabilmektedir.

3.4.4. Deney Tasarımının Seçimi

İlk üç aşama gerçekleştirildikten sonra tasarımcı denemeyi uygulayacağı tasarımı seçmektedir.

3.4.5. Deneyi Uygulama

Asıl verilerin elde edildiği aşamadır.

3.4.6. Verilerin İstatistiksel Analizi

Bu adımda elde edilen asıl veriler istatistiksel yöntemler ile analiz edilmektedir. Bu yöntemler grafiksel yöntem ya da varyans analizi yöntemi olabilmektedir.

3.4.7. Sonuçlar

Yapılan istatistiksel analiz sonucunda artık kesin ve net şekilde konuşulabilir.

3.5. Faktöriyel Denemeler

Araştırmalarda çok kez sonuca etkisi olan faktörlerin etki ölçüleri inceleme konusu olabilmektedir. Bu faktörlerin tek başlarına etkileri ve birbirleri ile aynı anda sonuç üzerindeki yarattıkları etki belirlenmeye çalışılmıştır. Faktörlerin tek başlarına etkileri yeterli olmamaktadır çünkü faktörlerin başka faktörler ile bir araya geldiği zaman sonuç üzerindeki etkileri tek başlarına yarattıkları etkiden farklı olabilmektedir. Bu şekilde sonuç üzerindeki önemli etki belirlenebilmektedir ve bu faktör üzerindeki değişiklikler ile en iyi sonuç elde edilebilmektedir (Karakoç, 2006).

3.5.1. Tam Faktöriyel Deneyler

Tek Faktörlü Deneyler: Her bir faktörün sonuç üzerindeki etkisinin tek tek ölçüldüğü deney türüdür. Seçilen faktörün sonuç üzerinde etkisi olmadığı belirlendiği takdirde yeni bir faktör denenmektedir. Bu işlem beklenen sonuca ulaşılıncaya kadar sürdürülmektedir. Bu deneme türünde faktörlerin birbiri ile etkileşimi ölçülememektedir. Yani faktörlerin farklı faktörler ile farklı şekilde yan yana gelerek sonuç üzerindeki etkileri bu deneme türü ile gözlemlenememektedir. Ayrıca faktör sayısı çok olduğu zaman her denemeyi yapmak zaman kaybına ve dolayısı ile fazla maliyete sebep olabilmektedir bu nedenle faktör sayısının fazla olduğu durumlarda ve faktörlerin etkileşiminin ölçülmesi istendiği durumlarda tam faktöriyel deney türünün kullanılması deney tasarımcısına avantaj sağlayabilmektedir.

Tam faktöriyel deneyler, her bir faktörün her bir seviyesi için eşit sayıda deneme ile faktörlerin diğer faktörlerden bağımsız olarak ürün performansına etkisini ölçmede kullanılan deney türüdür. Tam faktöriyel deney stratejisi faktörlerin birleştirilmiş halini inceler ve hem tek tek her faktörün sonuç üzerindeki etkisini hem de faktörlerin birlikte sonuç üzerindeki etkisini belirlemede kullanılmaktadır. Kritik faktörleri saptamaktadır. Tam faktöriyel deneyler ürün performansına etki edecek faktör sayısının 5 ten küçük olduğu durumlarda kullanılabilmektedir çünkü daha fazla faktör olduğu zaman deneme sayısı üstel olarak artmaktadır. Bu da deneme

sayısının çok fazla olması anlamına gelmektedir. Deneme sayısı arttıkça sonuca ulaşmada zaman problemi ortaya çıkabilmektedir. Faktör sayısının 5 ten fazla olduğu durumlarda kısmi faktöriyel deneyler kullanılmaktadır.

Tam faktöriyel deneylerde, faktörlerin tek başlarına ve birlikte sonuca olan etkilerini belirleyebilmek için gerekli deneme sayısı a^k formülü ile hesaplanmaktadır.

a: faktörün seviye sayısı
k: incelenen faktör sayısı
Örneğin, 3 faktörlü 2 seviyeli bir deneyin 2^3=8 tane denemesi yapılmaktadır.

Faktörlerin seviye sayısı belirlenirken 2 yöntem kullanılabilmektedir.
Tecrübe ile ürün performansına etkisi olası seviyeler seçilerek deney yapılabilmektedir ya da seviyelerin oluşturduğu bir kümeden rassal olarak seçilip deney yapılabilmektedir (Kasap, 2006).

3.6. Parametrelerin Deney Tasarımı ile Belirlenmesi

Bu bölümde çalışmanın sonucunu etkileyecek olan faktörleri ve seviyelerini belirlemek için deney tasarımı çalışması yapılmıştır. Tez çalışmasında en kısa yolu bulmak olan amacımıza ulaşmak için farksal evrim algoritması ve parçacık sürü en iyileme algoritması kullanılmaktadır. Öncelikle algoritmalarda kullanılacak ve sonucu belirleyecek faktörlerin neler olduğa karar verilmiştir. Bunlar her iki algoritma için de popülasyon büyüklüğü, mutasyon oranı ve çaprazlama oranı iken parçacık sürü en iyileme algoritması için deneye ikinci bir çaprazlama oranı faktörü dahil olmaktadır. Faktörler kararlaştırıldıktan sonra seviyeler belirlenmiştir. Seviye olarak her bir faktör için yüksek ve düşük olmak üzere iki ölçü kullanılmıştır, literatürdeki konu ile ilgili çalışmalarda genellikle iki seviye kullanıldığı gözlemlenmiştir. Faktörler ve seviyeleri bu şekilde belirlendikten sonra tam faktöriyel deney yapılması kararlaştırılmıştır. Farksal evrim algoritması ve parçacık sürü en iyileme algoritması için olası faktör seviyeleri Tablo 3.1ve 3.2 deki gibi belirlenmiştir.

Tablo 3.1 Farksal Evrim Algoritması için Olası Faktör Seviyeleri ve Değerleri

	A(NP)	*B(MR)*	*C(CR)*
Düşük (-1)	30	0,2	0,2
Yüksek(1)	100	0,9	0,9

Tablo 3.2 Parçacık Sürü En İyileme Algoritması için Olası Faktör Seviyeleri ve Değerleri

	A(NP)	*B(CR1)*	*C(CR2)*	*D(MR)*
Düşük (-1)	30	0,2	0,2	0,2
Yüksek(1)	100	0,9	0,9	0,9

Farksal evrim algoritması çözümü için 2^3= 8 deneme yapılmıştır. Deney tasarımları Tablo 3.3. de gösterilmiştir. Parçacık sürü en iyileme algoritması çözümü için 2^4= 16 deney tasarımı yapılmıştır, Tablo 3.5 te gösterilmiştir. Denemeler şehir sayısı 200 ile 442 arasında değişen 12 test problemi üzerinde yapılmıştır. Denemeler sonucunda elde edilen ortalama nısbi sapma yüzde değerlerine bakılarak uygun deney tasarımı belirlenmiştir.

3.6.1. FEA nın Parametrelerin Deney Tasarımı ile Belirlenmesi

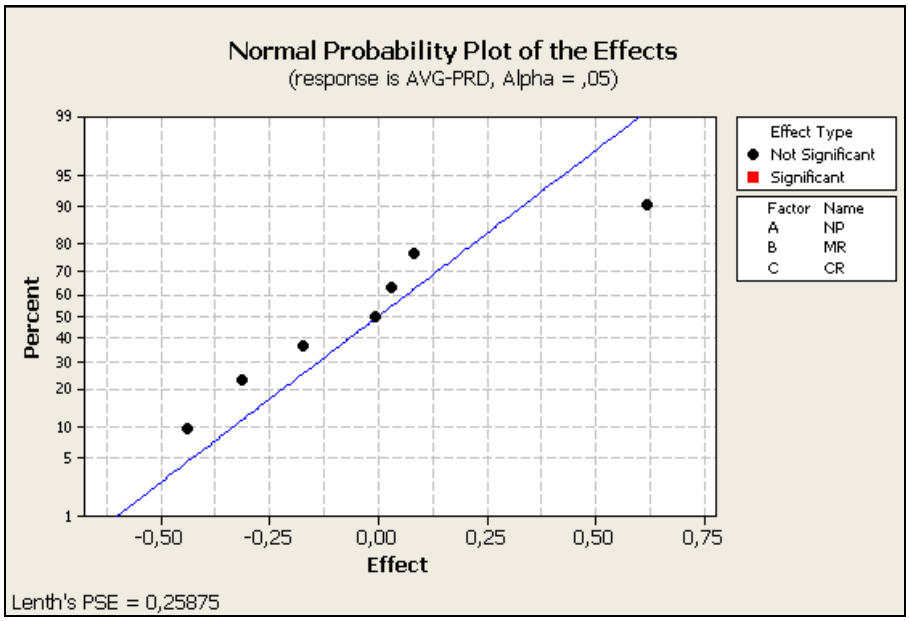
Tablo 3.3 Farksal Evrim Algoritması için Belirlenen Faktörlerin Tasarım Seçenekleri

DENEME	POPÜLASYON BÜYÜKLÜĞÜ	MUTASYON ORANI	ÇAPRAZLAMA ORANI	ORTALAMA NISBİ SAPMA (%)
1	1	1	1	1,42
2	1	-1	1	**0,95**
3	-1	-1	1	1,33
4	1	-1	-1	1
5	-1	1	-1	2,31
6	1	1	-1	1,83
7	-1	-1	-1	1,57
8	-1	1	1	1,75

Farksal evrim algoritması için yapılan denemede en iyi (optimal) çözümden ortalama nısbi sapma yüzdesi deneme 2 de en iyi değeri verdiği gözlemlenmektedir, yani popülasyon büyüklüğünün yüksek, mutasyon oranının düşük, çaprazlama oranının yüksek olarak belirlenmesinin uygun olduğu saptanmaktadır.

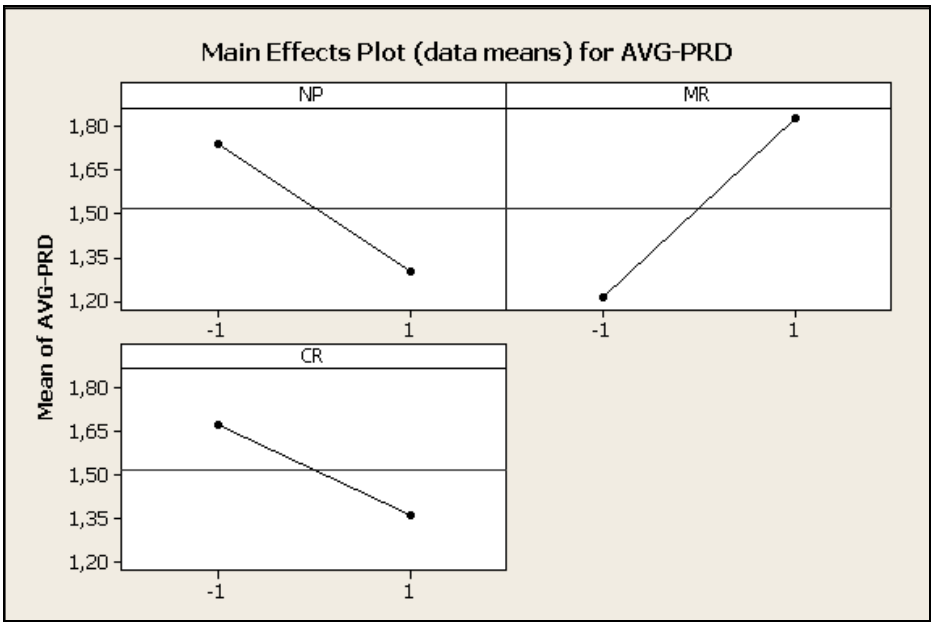
Elde edilen sonuçlara göre önemli faktörleri belirlemek için normal olasılık grafiği, ana etkiler (main effects plot) grafiği ve faktörler arasında etkileşim olup olmadığını kontrol etmek için etkileşim grafikleri minitab 14 programı ile çizilmiş ve onların da sonuçları değerlendirilmiştir.

Farksal evrim algoritması için yapılan deney tasarımını inceleyecek olursak;



Şekil 3.1 FEA için Etkilerin Normal Olasılıklı Grafiği

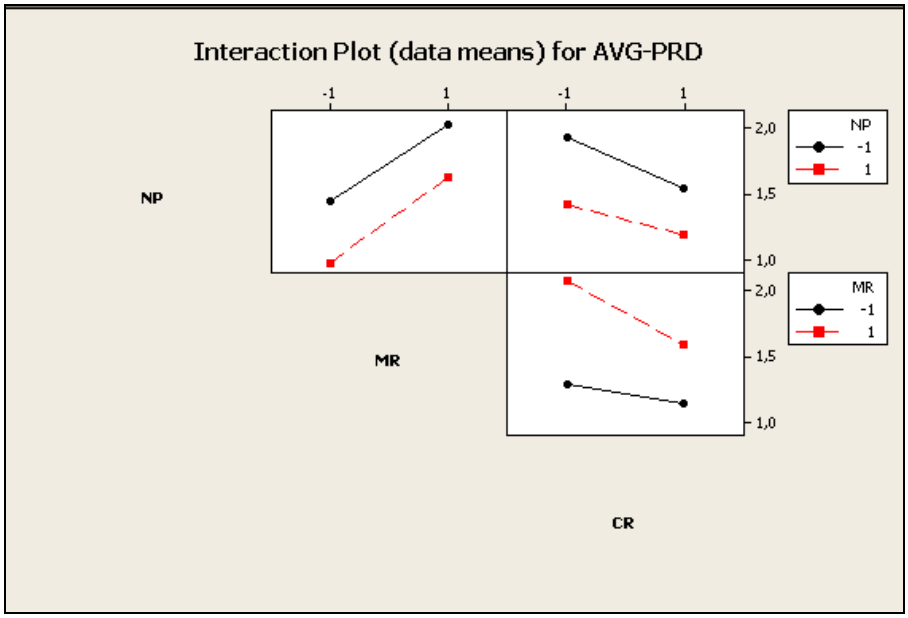
Şekil 3.1 de gösterilen etkilerin normal olasılık grafiğine göre faktörlerin hiç birinin önemli çıkmadığını yani hiçbirinin seviyesinin düşükten yükseğe çıkmasının sonuç üzerinde anlamlı bir etkisi olmadığı gözlemlenmektedir. Bu durumda uygun tasarımı seçmede Şekil 3.2 de gösterilen ana etki grafiğine bakacak olursak;



Şekil 3.2 FEA için Ana Etki Grafiği

Popülasyon büyüklüğü faktörünü yüksek, mutasyon oranı faktörünü düşük, çaprazlama faktörünü ise yüksek seçmemiz gerektiği ana etkiler grafiğinden de okunmaktadır.

Faktörler arasında etkileşim olup olmadığını belirlemek için Şekil 3.3 de gösterilen faktörlerin etkileşim grafiği çizilmiştir.



Şekil 3.3 FEA için Etkileşim Grafiği

Etkileşim grafiğine bakılarak faktörler arasında etkileşim yok denecek kadar az olduğu söylenebilmektedir. Bu durumda ana etkiler grafiğine bakarak uygun deney tasarımı, tasarım 2 yani (+1,-1,+1) olarak belirlenebilmektedir.

Yapılan denemeler sonucunda farksal evrim algoritmasındaki faktörleri ve seviyeleri Tablo 3.4 deki gibi belirlenmiştir.

Tablo 3.4 Farksal Evrim Algoritması için Belirlenen Faktör Seviyeleri ve Değerleri

Faktör	Seviye	Açıklama	Değer
A	1	Popülasyon büyüklüğü	NP=100
B	-1	Mutasyon oranı	MR=0,2
C	1	Çaprazlama oranı	CR=0,9

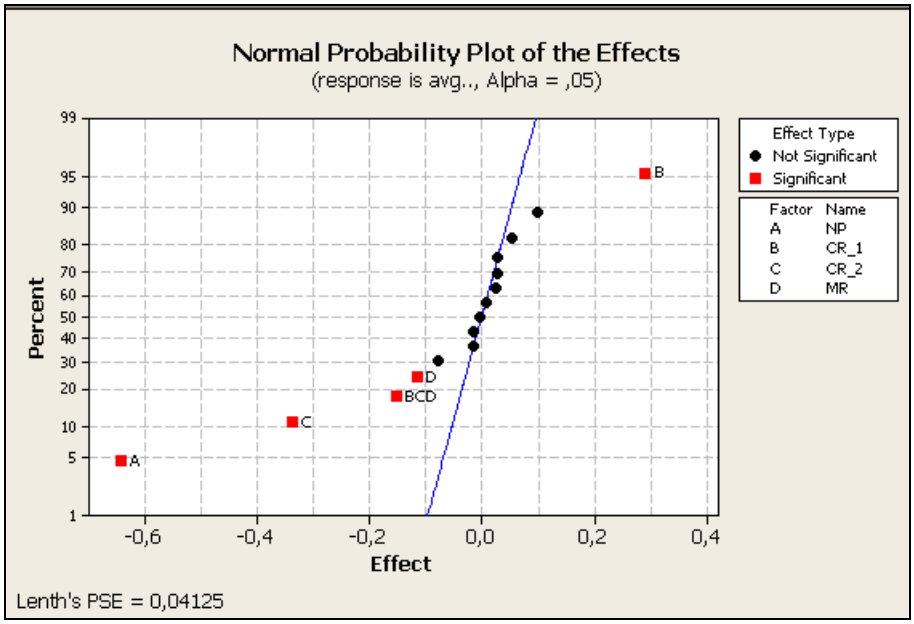
3.6.2. PSE Algoritmasının Parametrelerinin Deney Tasarımı ile Belirlenmesi

Parçacık sürü en iyileme algoritması için yapılan deney sonucunda (+1,-1,-1,+1) denemesinin en iyi sonucu verdiği gözlemlenmektedir.

Tablo 3.5 Parçacık Sürü En İyileme Algoritması için Belirlenen Faktörlerin Tasarım Seçenekleri

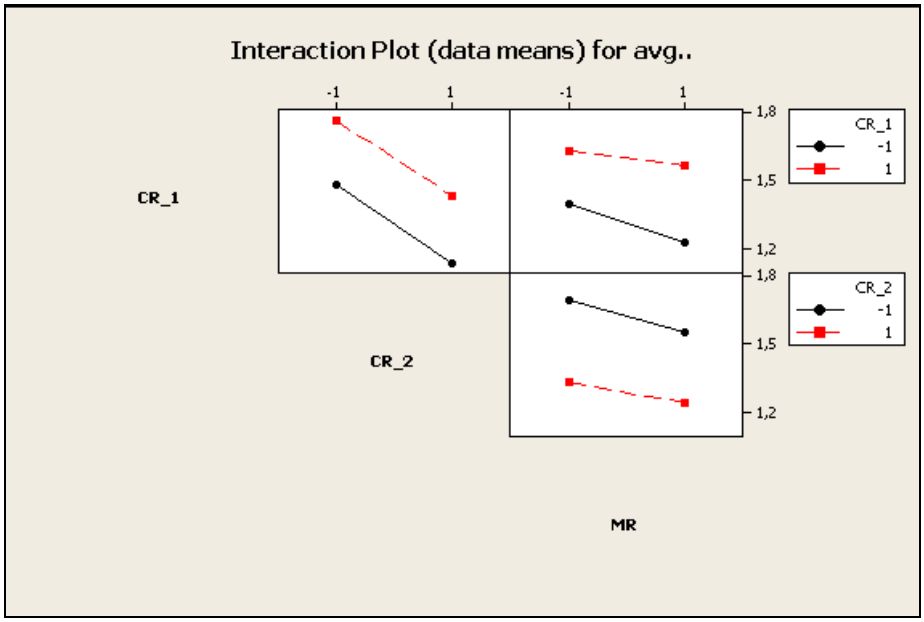
DENEME	POPÜLASYON BÜYÜKLÜĞÜ	MUTASYON ORANI	ÇAPRAZLAMA ORANI 1	ÇAPRAZLAMA ORANI 2	ORTALAMA NISBİ SAPMA (%)
1	-1	1	-1	1	1,37
2	-1	1	1	1	1,64
3	1	-1	1	1	1,23
4	-1	1	1	-1	2,19
5	-1	-1	-1	1	1,39
6	-1	-1	-1	-1	1,99
7	1	1	-1	1	0,91
8	1	-1	-1	1	**0,87**
9	1	1	1	-1	1,4
10	1	1	-1	-1	1
11	1	-1	-1	-1	1,32
12	1	-1	1	-1	1,28
13	-1	-1	1	1	1,82
14	1	1	1	1	1,04
15	-1	1	-1	-1	1,61
16	1-	-1	1	-1	2,18

Parçacık sürü en iyileme algoritması için yapılan deney tasarımı incelendiğinde; Şekil 3.4 de gösterilen etkilerin normal olasılık grafiğine göre algoritmada kullanılacak 4 faktörün her birinin anlamlı olduğu yani faktörlerin seviyelerinin düşükten yükseğe çıkmasının sonuç üzerinde anlamlı bir etki yarattığı görülmektedir.



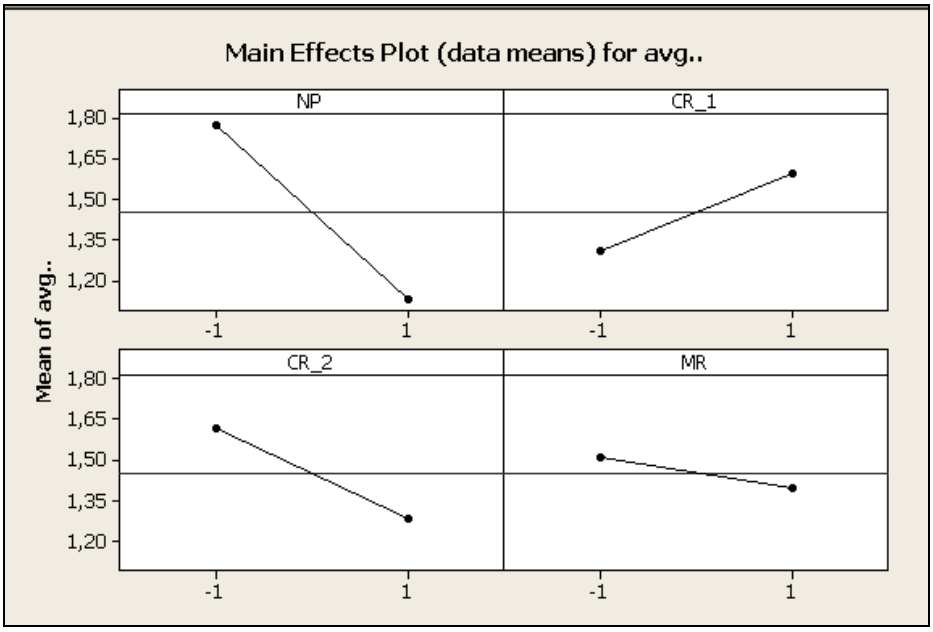
Şekil 3.4 PSE Algoritması için Etkilerin Normal Olasılıklı Grafiği

Ayrıca BCD tasarımının önemli çıktığı gözlemlenmektedir. B, C ve D faktörlerinin birbirleri ile etkileşim halinde olduğunu yani bir faktörün seviyesinin değişmesi durumunda diğerlerinin de bu durumdan etkilendiği anlaşılmaktadır.



Şekil 3.5 PSE Algoritmasının Etkileşim Grafiği

Şekil 3.5 de gösterilen etkileşim grafiğinde ise BCD faktörleri arasında etkileşim yok denecek kadar az düzeyde olduğu için yine sadece ana etkileşimler grafiğine bakarak uygun tasarımı belirlenmektedir.



Şekil 3.6 PSE Algoritması için Ana Etki Grafiği

Şekil 3.6 da gösterilen ana etkiler grafiğine bakarak popülasyon büyüklüğünü yüksek, mutasyon oranını yüksek, 1. Çaprazlama oranını düşük ve 2. Çaprazlama oranını yüksek olarak seçmemiz durumunda yani (+1,+1,-1,+1) tasarımını seçerek en iyi sonuca ulaşılacağı görülmektedir.

Parçacık sürü en iyileme algoritmasının faktörleri ve seviyeleri Tablo 3.6. daki gibi belirlenmiştir.

Tablo 3.6 Parçacık Sürü En İyileme Algoritması için Belirlenen Faktör Seviyeleri ve Değerleri

Faktör	Seviye	Açıklama	Değer
A	1	Popülasyon büyüklüğü	NP=100
B	1	Mutasyon oranı	MR=0,9
C	-1	Çaprazlama oranı 1	CR 1=0,2
D	1	Çaprazlama oranı 2	CR 2=0,9

BÖLÜM 4

4. DENEYSEL SONUÇLAR VE DEĞERLENDİRMELER

Bu tez çalışmasında gezgin satıcı problemi çözümünde farksal evrim algoritması ve parçacık sürü en iyileme algoritması kullanılmıştır. Algoritma durdurma kriteri sağlanıncaya kadar çalıştırılmıştır. Durdurma kriteri, popülasyondaki en iyi çözüm 10 defa iyileşmediği takdirde algoritmanın durdurulması şeklinde belirlenmiştir. Algoritmaların performanslarının kıyaslanabilmesi için her ikisinde de 5 tekrarlama yapılarak en iyi çözümden ortalama nisbi sapma yüzdeleri istatistiksel olarak aşağıdaki gibi hesaplanmıştır:

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(H_i - B) \times 100}{B} \right) / R \qquad (4.1)$$

Burada H_i: algoritma tarafından üretilen çözüm değeri,

B: en iyi (optimal) değeri,

R: tekrarlama sayısını ifade etmektedir. CPU zamanı karşılaştırmaları için t_{ort}, ortalama CPU zamanını göstermektedir.

4.1. Kullanılan Problemler

Algoritmaların sonuçları GSP kütüphanesinden alınan, bilinen en iyi çözümler ile karşılaştırılmıştır. Şehir sayısı 48 ile 1084 arasında değişen 54 test problemi üzerinde çalışma yapılmıştır. Kullanılan problemler:

att48.txt, gr48.txt, hk48.txt, eil51.txt, brazil58.txt, st70.txt, eil76.txt, pr76.txt, kroa100.txt, krob100.txt, kroc100.txt, krod100.txt, kroe100.txt, rat99.txt, rd100.txt, eil101.txt, lin105.txt, pr107.txt, gr120.txt, pr124.txt, bier127.txt, pr136.txt, pr144.txt, kroa150.txt, krob150.txt, pr152.txt, u159.txt, rat195.txt, d198.txt, kroa200.txt, krob200.txt, ts225.txt, pr226.txt, gil262.txt, pr264.txt, pr299.txt, lin318.txt, rd400.txt, fl417.txt, pr439.txt, pcb442.txt, d493.txt, att532.txt, att532.txt, si535.txt, pa561.txt, rat575.txt, p654.txt, u724.txt, rat783.txt, pr1002.txt, si1032.txt, u1060.txt, vm1084.txt

Örneğin, att48.txt problemi 48 şehirli problemdir. gr48.txt problemi ise yine 48 şehirli ancak başka bir araştırmacı tarafından çözülen başka bir problemdir. Problemlerle ilgili veriler TSPLIB adındaki GSP kütüphanesinden alınmıştır (http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp/). Örneğin gr48.txt probleminin verileri Tablo 4.1 de verilmiştir.

Tablo 4.1 gr48.txt Probleminin Verileri

48	5046				
0	593	409	566	633	257...
593	0	258	331	586	602...
409	258	0	171	723	522…
566	331	171	0	874	679…
633	586	723	874	0	390…
257	602	522	679	390	0...
…	…	…	…	…	…
…	…	…	…	…	…

Tablo 4.1 de ilk satırdaki rakamlar şehir sayısını ve en iyi (optimal) sonucu göstermektedir. 2. satırdaki rakamlar sırası ile $d_{11,}d_{12,}d_{13,}d_{14,}d_{15,}d_{16...}$uzaklıklarını göstermektedir. 3. satırdaki rakamlar ise sırası ile $d_{21,}d_{22,}d_{23,}d_{24,}d_{25,}d_{26...}$uzaklıklarını göstermektedir.

Farksal evrim algoritması ve parçacık sürü en iyileme algoritması Visual C++ da programlanmıştır ve Intel P IV 3.00 GHz 512 MB belleği olan bilgisayarda çalıştırılmışlardır. Algoritmaların parametreleri deney tasarımı bölümünde belirlenmiştir.

Algoritmaların performansları Tablo 4.2 ve tablo 4.3 de gösterilmiştir. Tabloların 1. sütunu test problemlerini, 2. sütunu problemlere ait bilinen en iyi sonuçları, 3. sütun mevcut çözüm yöntemi ile 5 tekrarın kaçında en iyi değere ulaşıldığını, 4. sütun en iyi çözümden ortalama nısbi sapma yüzdesini, 5. sütun en küçük sapma yüzdesini, 6. sütun en büyük sapma yüzdesini, 7. sütun tez çalışmasındaki algoritmanın en iyi sonucunu, son sütun ise ortalama işlemci (CPU) zamanını göstermektedir.

4.2. Kesikli Farksal Evrim Algoritmasının Performansı

Kesikli farksal evrim algoritması ile çözülmüş 54 test probleminin sonuçları Tablo 4.2 de gösterilmiştir. Problemler, algoritmaların 5 kez çalıştırılması (run) ile çözülmüştür. Test edilen 54 problemin 8 inde 5 çalıştırmanın tamamında en iyi çözüm elde edilmiştir, 24 ünde 5 çalıştırmanın en az birinde en iyi çözüm elde edilmiştir. Bilinen en iyi çözümlerden ortalama nısbi sapma yüzdesi 0,81 olarak bulunmuştur. Ortalama işlemci zamanı, t_{ort} (CPU) 255,56 olarak bulunmuştur.

Tablo 4.2 KFEA Algoritmasının Performansı

PROBLEM	En İyi	n_{opt}	Δ_{ort}	Δ_{min}	Δ_{max}	KFEA	t_{ort}
att48.txt	10628	5	0	0	0	10628	0,48
gr48.txt	5046	4	0,13	0	0,65	5046	0,35
hk48.txt	11461	4	0,17	0	0,83	11461	0,48
eil51.txt	426	0	0,28	0,23	0,47	427	0,48
brazil58.txt	25395	5	0	0	0	25395	0,54
st70.txt	675	5	0	0	0	675	0,91
eil76.txt	538	3	0,71	0	2,6	538	1,21
pr76.txt	108159	5	0	0	0	108159	1,14
kroa100.txt	21282	4	0,02	0	0,11	21282	1,97
krob100.txt	22141	1	0,39	0	0,65	22141	2,62
kroc100.txt	20749	5	0	0	0	20749	2,29
krod100.txt	21294	1	0,41	0	0,8	21294	2,54
kroe100.txt	22068	0	0,39	0,17	0,79	22106	2,88
rat99.txt	1211	2	0,18	0	0,5	1211	2,83
rd100.txt	7910	3	0,09	0	0,43	7910	2,4
eil101.txt	629	2	1,21	0	2,86	629	2,75
lin105.txt	14379	5	0	0	0	14379	3,38
pr107.txt	44303	3	0,08	0	0,19	44303	3,35
gr120.txt	6942	0	0,77	0,27	1,43	6961	4,42
pr124.txt	59030	3	0,03	0	0,1	59030	3,83
bier127.txt	118282	0	0,3	0,03	0,69	118313	4,93
pr136.txt	96772	0	0,71	0,15	1,85	96920	7,36
pr144.txt	58537	5	0	0	0	58537	4,97
kroa150.txt	26524	0	0,55	0,27	1,09	26596	6,91
krob150.txt	26130	0	0,36	0,02	1,3	26136	7,84
pr152.txt	73682	1	0,18	0	0,27	73682	6,72
u159.txt	42080	3	0,48	0	1,46	42080	7,63
rat195.txt	2323	0	1,64	0,82	2,58	2342	13,68
d198.txt	15780	0	0,41	0,29	0,72	15826	18,01
kroa200.txt	29368	0	0,82	0,04	2,7	29380	17,12
krob200.txt	29437	0	1,61	0,37	2,99	29546	17,5
ts225.txt	126643	5	0	0	0	126643	17,11
pr226.txt	80369	2	0,02	0	0,06	80369	24,18
gil262.txt	2378	0	3,21	1,51	4,54	2414	30,38
pr264.txt	49135	2	0,28	0	0,72	49135	37,77
pr299.txt	48191	0	0,49	0,29	0,88	48330	58,32
lin318.txt	42029	0	1,09	0,76	1,61	42347	70,57
rd400.txt	15281	0	1,7	1,07	2,24	15444	142,6
fl417.txt	11861	0	0,2	0,13	0,31	11877	127,02
pr439.txt	107217	0	0,76	0,15	2,13	107374	171,45
pcb442.txt	50778	0	1,26	0,79	1,88	51178	154,38
d493.txt	35002	0	1,98	1,35	2,39	35473	313,34
att532.txt	27686	0	1,61	1,26	2,23	28034	403,4
si535.txt	48450	0	0,07	0,04	0,13	48471	287,72
pa561.txt	2763	0	1,94	1,56	2,42	2806	378,74
rat575.txt	6773	0	2,5	1,98	3,35	6907	362,78
p654.txt	34643	0	0,18	0,1	0,27	34677	523,06
d657.txt	48912	0	2,06	1,58	3,13	49684	623,32
u724.txt	41910	0	2,45	1,11	3,37	42375	642
rat783.txt	8806	0	3,03	2,35	4,08	9013	984,12
pr1002.txt	259045	0	2,71	2,37	3,12	265197	1984,73
si1032.txt	92650	1	0,08	0	0,14	92650	1144,64
u1060.txt	224094	0	2,26	1,7	2,7	227900	2414,59
vm1084.txt	239297	0	1,8	1,24	2,53	242263	2750,29
ORTALAMA		**1,46**	**0,81**	**0,44**	**1,34**	**47115,61**	**255,56**

4.3. Kesikli Parçacık Sürü En İyileme Algoritmasının Performansı

Kesikli parçacık sürü en iyileme algoritması ile çözülmüş 54 test probleminin sonuçları Tablo 4.3 de gösterilmiştir. Problemler, algoritmaların 5 kez çalıştırılması (run) ile çözülmüştür. Test edilen 54 problemin 9 unda 5 çalıştırmanın tamamında en iyi çözüm elde edilmiştir, 25 inde 5 çalıştırmanın en az birinde en iyi çözüm elde edilmiştir. Bilinen en iyi çözümlerden ortalama nısbi sapma yüzdesi 0,88 olarak belirlenmiştir. Ortalama işlemci zamanı, t_{ort} (CPU) 223,45 olarak belirlenmiştir.

Tablo 4.3 KPSE Algoritmasının Performansı

PROBLEM	En İyi	n_{opt}	Δ_{ort}	Δ_{min}	Δ_{max}	KPSE	t_{ort}
att48.txt	10628,00	5	0,00	0,00	0,00	10628,00	0,48
gr48.txt	5046,00	3	0,22	0,00	0,55	5046,00	0,40
hk48.txt	11461,00	5	0,00	0,00	0,00	11461,00	0,48
eil51.txt	426,00	3	0,09	0,00	0,23	426,00	0,70
brazil58.txt	25395,00	5	0,00	0,00	0,00	25395,00	0,57
st70.txt	675,00	5	0,00	0,00	0,00	675,00	1,05
eil76.txt	538,00	3	0,37	0,00	0,93	538,00	1,46
pr76.txt	108159,00	5	0,00	0,00	0,00	108159,00	1,30
kroa100.txt	21282,00	5	0,00	0,00	0,00	21282,00	2,05
krob100.txt	22141,00	2	0,42	0,00	1,00	22141,00	2,92
kroc100.txt	20749,00	5	0,00	0,00	0,00	20749,00	2,42
krod100.txt	21294,00	0	0,44	0,38	0,54	21374,00	2,68
kroe100.txt	22068,00	2	0,10	0,00	0,24	22068,00	3,52
rat99.txt	1211,00	2	0,38	0,00	1,73	1211,00	2,77
rd100.txt	7910,00	2	0,48	0,00	1,29	7910,00	2,27
eil101.txt	629,00	1	0,64	0,00	1,43	629,00	3,11
lin105.txt	14379,00	3	0,16	0,00	0,67	14379,00	3,03
pr107.txt	44303,00	2	0,22	0,00	0,71	44303,00	3,76
gr120.txt	6942,00	0	0,69	0,13	1,74	6951,00	4,17
pr124.txt	59030,00	3	0,07	0,00	0,26	59030,00	4,23
bier127.txt	118282,00	2	0,09	0,00	0,38	118282,00	5,74
pr136.txt	96772,00	0	0,87	0,09	1,24	96861,00	7,28
pr144.txt	58537,00	5	0,00	0,00	0,00	58537,00	4,98
kroa150.txt	26524,00	1	0,27	0,00	0,95	26524,00	8,22
krob150.txt	26130,00	0	0,51	0,07	1,33	26148,00	7,27
pr152.txt	73682,00	4	0,04	0,00	0,18	73682,00	7,42
u159.txt	42080,00	4	0,15	0,00	0,75	42080,00	9,04
rat195.txt	2323,00	0	1,82	0,77	2,67	2341,00	15,96
d198.txt	15780,00	0	0,51	0,23	0,89	15816,00	16,62
kroa200.txt	29368,00	0	0,62	0,05	2,02	29382,00	19,41
krob200.txt	29437,00	0	0,83	0,39	1,41	29551,00	16,98
ts225.txt	126643,00	5	0,00	0,00	0,00	126643,00	18,26
pr226.txt	80369,00	0	0,03	0,00	0,07	80373,00	20,93
gil262.txt	2378,00	0	1,49	0,80	2,35	2397,00	43,91
pr264.txt	49135,00	2	0,29	0,00	0,77	49135,00	35,33
pr299.txt	48191,00	0	1,59	0,83	2,34	48591,00	51,03
lin318.txt	42029,00	0	1,80	0,66	2,41	42305,00	61,28
rd400.txt	15281,00	0	1,48	1,28	1,67	15477,00	122,01
fl417.txt	11861,00	0	0,36	0,30	0,42	11896,00	132,53
pr439.txt	107217,00	0	0,95	0,56	2,18	107813,00	177,99
pcb442.txt	50778,00	0	1,50	1,04	1,90	51306,00	163,69
d493.txt	35002,00	0	2,11	1,39	2,95	35487,00	289,37
att532.txt	27686,00	0	1,71	1,14	2,17	28002,00	322,18
si535.txt	48450,00	0	0,11	0,08	0,15	48487,00	276,47
pa561.txt	2763,00	0	2,43	1,56	2,93	2806,00	282,93
rat575.txt	6773,00	0	3,00	2,55	3,43	6946,00	369,08
p654.txt	34643,00	0	0,34	0,28	0,39	34739,00	479,53
d657.txt	48912,00	0	2,44	1,60	3,17	49696,00	613,06
u724.txt	41910,00	0	2,89	2,47	3,28	42947,00	650,30
rat783.txt	8806,00	0	3,87	3,08	4,63	9077,00	774,28
pr1002.txt	259045,00	0	3,12	2,15	4,09	264615,00	1721,11
si1032.txt	92650,00	0	0,15	0,09	0,17	92735,00	1176,68
u1060.txt	224094,00	0	3,48	2,83	4,01	230434,00	1877,81
vm1084.txt	239297,00	0	2,63	2,33	2,95	244881,00	2244,03
ORTALAMA		**1,56**	**0,88**	**0,54**	**1,33**	**47228,65**	**223,45**

4.4. KFEA ile KPSE Algoritmalarının Performanslarının Karşılaştırılması

Çalışmanın bu bölümünde KFEA ile KPSE algoritmasının performansları en iyi çözümden ortalama nısbi sapma yüzdesine ve ortalama işlemci zamanına göre karşılaştırılmıştır. Tablo 4.4 te farksal evrim algoritmasının optimal çözümden ortalama nısbi sapma yüzdesi parçacık sürü en iyileme algoritmasına göre daha küçüktür. Ancak ortalama işlemci zamanı ise farksal evrim algoritmasının ortalama işlemci zamanından daha büyüktür. Bu durumda elde edilen sonuçlara çift taraflı eşlenik *t*- testi uygulayarak algoritmaların eşdeğer olup olmadığını incelenmiştir.

H_0: eşittir ($h_{KFE} = h_{PSE}$)
H_1 : eşit değildir ($h_{KFE} \neq h_{PSE}$)
α = 0,05 (güven aralığı 0,95)
p değeri < α ise H_0 ı reddet
p değeri > α ise H_0 ı kabul et.

0 hipotezi iki algoritmanın eşdeğer olduğunu belirtmektedir. 1 hipotezi ise iki algoritmanın eşdeğer olmadığını belirtmektedir. Çift taraflı eşlenik *t*- testi uyguladığım sonuçlarda elde ettiğim p değerinin, 0,05 olarak belirlediğim anlamlılık düzeyinden küçük olması durumunda 0 hipotezi reddedilmektedir. Aksi takdirde kabul edilmektedir.

54 test probleminin ortalama nısbi sapma yüzdeleri ve ortalama işlemci zamanları kullanılarak Minitab 14 programı ile %95 (α=0,05) anlamlılık düzeyinde çift taraflı eşlenik *t*-testi uygulanmıştır. Ortalama nısbi sapma yüzdelerine göre yapılan çift taraflı eşlenik *t*- testinde p değeri 0,205 bulunmuştur. P değeri α değerinden büyük olduğu h_0 kabul edilir. Yani optimal sonuçların ortalama nısbi sapma yüzdesine göre iki algoritma eşittir. Dolayısıyla optimal sonuçtan ortalama nısbi sapma yüzdesine göre iki algoritmanın performansları arasında bir fark olmadığı ve eşdeğer olduklarına karar verilir.

Ortalama işlemci zamanlarına göre yapılan çift taraflı eşlenik *t*- testinde p değeri 0.034 bulunmuştur. P değeri α değerinden küçük olduğu h_0 reddedilir. Yani optimal sonuçları elde etmek için harcanan ortalama işlemci zamana göre iki algoritma eşit değildir. Dolayısıyla ortalama işlemci zamana göre iki algoritmanın performansları arasında bir fark olduğu ve eşdeğer olmadıkları sonucuna varılır. Dolayısıyla, KPSE algoritması, KFE algoritmasına göre daha hızlı bir algoritmadır.

Tablo 4.4 KFEA ve KPSE Algoritmasının Performanslarının Karşılaştırılması

	KFEA		KPSE	
PROBLEM	Δ_{ort}	t_{ort}	Δ_{ort}	t_{ort}
att48.txt	0,00	0,48	0,00	0,48
gr48.txt	0,13	0,35	0,22	0,40
hk48.txt	0,17	0,48	0,00	0,48
eil51.txt	0,28	0,48	0,09	0,70
brazil58.txt	0,00	0,54	0,00	0,57
st70.txt	0,00	0,91	0,00	1,05
eil76.txt	0,71	1,21	0,37	1,46
pr76.txt	0,00	1,14	0,00	1,30
kroa100.txt	0,02	1,97	0,00	2,05
krob100.txt	0,39	2,62	0,42	2,92
kroc100.txt	0,00	2,29	0,00	2,42
krod100.txt	0,41	2,54	0,44	2,68
kroe100.txt	0,39	2,88	0,10	3,52
rat99.txt	0,18	2,83	0,38	2,77
rd100.txt	0,09	2,40	0,48	2,27
eil101.txt	1,21	2,75	0,64	3,11
lin105.txt	0,00	3,38	0,16	3,03
pr107.txt	0,08	3,35	0,22	3,76
gr120.txt	0,77	4,42	0,69	4,17
pr124.txt	0,03	3,83	0,07	4,23
bier127.txt	0,30	4,93	0,09	5,74
pr136.txt	0,71	7,36	0,87	7,28
pr144.txt	0,00	4,97	0,00	4,98
kroa150.txt	0,55	6,91	0,27	8,22
krob150.txt	0,36	7,84	0,51	7,27
pr152.txt	0,18	6,72	0,04	7,42
u159.txt	0,48	7,63	0,15	9,04
rat195.txt	1,64	13,68	1,82	15,96
d198.txt	0,41	18,01	0,51	16,62
kroa200.txt	0,82	17,12	0,62	19,41
krob200.txt	1,61	17,50	0,83	16,98
ts225.txt	0,00	17,11	0,00	18,26
pr226.txt	0,02	24,18	0,03	20,93
gil262.txt	3,21	30,38	1,49	43,91
pr264.txt	0,28	37,77	0,29	35,33
pr299.txt	0,49	58,32	1,59	51,03
lin318.txt	1,09	70,57	1,80	61,28
rd400.txt	1,70	142,60	1,48	122,01
fl417.txt	0,20	127,02	0,36	132,53
pr439.txt	0,76	171,45	0,95	177,99
pcb442.txt	1,26	154,38	1,50	163,69
d493.txt	1,98	313,34	2,11	289,37
att532.txt	1,61	403,40	1,71	322,18
si535.txt	0,07	287,72	0,11	276,47
pa561.txt	1,94	378,74	2,43	282,93
rat575.txt	2,50	362,78	3,00	369,08
p654.txt	0,18	523,06	0,34	479,53
d657.txt	2,06	623,32	2,44	613,06
u724.txt	2,45	642,00	2,89	650,30
rat783.txt	3,03	984,12	3,87	774,28
pr1002.txt	2,71	1984,73	3,12	1721,11
si1032.txt	0,08	1144,64	0,15	1176,68
u1060.txt	2,26	2414,59	3,48	1877,81
vm1084.txt	1,80	2750,29	2,63	2244,03
ORTALAMA	**0,81**	**255,56**	**0,88**	**223,45**

4.5. Sonuçlar

Bu tez çalışmasında Gezgin Satıcı Problemi, farksal evrim algoritması ve parçacık sürü en iyileme algoritması ile çözülmüş ve algoritmaların performansları değerlendirilmiştir. Ancak algoritmalar sürekli problemlere göre tasarlandığı için öncelikle kesikli problem olan GSP ye uyarlanmıştır. Algoritmaların yerel en küçük çözüme takılmaması için NEH, ekleme, boz-yap sezgiselleri ve PTL çaprazlama işlemcisi kullanılmıştır. Çözüme etkisi olan faktörler ve seviyeleri deney tasarımı yöntemi ile belirlenmiştir. Algoritmalar GSP kütüphanesinden alınmış 48 ile 1084 arasında değişen şehir sayısına sahip 54 test problemine uygulanmıştır.

Sonuç olarak, GSP çözümünde en iyi çözümü bulmada her iki algoritmanın eşdeğer olduğu sonucuna varılmıştır ancak hesaplama süresi açısından kesikli parçacık sürü en iyileme algoritmasının kesikli farksal evrim algoritmasına göre daha iyi performans gösterdiği sonucuna varılmıştır.

KAYNAKLAR DİZİNİ

Davendra, D., Onwubolu, G.(2009).Motivation for Differential Evolution for Permutative-Based Combinatorial Problems. C.G. Onwubolu, ve D. Davendra, (Ed.), *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization.* (V.175) içinde (s:9). Berlin: Springer.

Eberhart, R.C., Shi, Y. (2001). Particle Swarm Optimization: Developments, Applications and Resources. *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*. Güney Kore. s: 81-86.

Erentürk, M. (2003). Performance Analysis of Meta-Heuristic Approaches for Traveling Salesperson Problem. *Yayınlanmamış Yüksek Lisans Tezi*. Yeditepe Üniversitesi. Fen Bilimleri Enstitüsü. s:3-7.

Jian L., Peng C., Zhiming L.(2008). Solving Traveling Salesman Problems by Genetic Differential Evolution with Local Search. *Workshop on Power Electronics and Intelligent Transportation System, Computer Society*, s:454-457.

Jones, D. F., Mirrazavi, S.K., Tamiz M., (2002). Multi-objective meta-heuristics:An overview of the current state-of-the-art, *Europen Journal of Operational Research*, 137, 1-9

Karaboğa, D., Ökdem, S. (2004). A Simple and Global Optimization Algorithm for Engineering Problems: Diferential Evolution Algorithm. *Turk J Elec Engin*, Vol.12, No.1. © Tübitak. s:53-60.

Karakoç, Ö. (2006). Deneylerin Faktöriyel Tasarımı. *Yayınlanmamış Yüksek Lisans Tezi*. Marmara Üniversitesi. s:3-14.

Kasap, Ş. (2006). F-4 Uçaklarında Yakıt Tüketimini Etkileyen Faktörlerin 2^k Deneyleri ve Taguchi Yöntemiyle Belirlenmesi. *Yayınlanmamış Yüksek Lisans Tezi*. Marmara Üniversitesi. Anadolu Üniversitesi. s:12-15.

Keskintürk, T.(2006a). Gezgin Satıcı Probleminin Farksal Gelişim Algoritması ile Çözümü. *VI. Ulusal Üretim Araştırmaları Sempozyumu*, İstanbul Kültür Üniversitesi. s:651-657.

Keskintürk, T.(2006b). Farksal Gelişim Algoritması. İstanbul Ticaret Üniversitesi *Fen Bilimleri Dergisi* Yıl: 5 Sayı: 9. s.85-99.

Koyuncu, C. A. (2006). Farksal Gelişim Algoritmasının İncelenmesi ve İşaret Kestiriminde Kullanılması. *Yayınlanmamış Yüksek Lisans Tezi*. Erciyes Üniversitesi.

Özsağlam, M.Y.(2009). Parçacık Sürü Optimizasyonu Algoritmasının Gezgin Satıcı Problemine Uygulanması ve Performansının İncelenmesi. *Yayınlanmış Yüksek Lisans Tezi*. Selçuk Üniversitesi. s:8-76.

Pan, Q., Taşgetiren, M.F., Liang, Y.C.(2008). A Discrete Particle Swarm Optimization Algorithm for the No-Wait Flowshop Scheduling Problem. *Computers & Operations Research 35*. s:2809-2811.

Price, K.V., Storn, R.M., Lampinen J.A.(2005). The Differential Evolution Algorithm. *Differential Evolution: A Practical Approach to Global Optimization*. Springer.s:37-40. Berlin:Springer.

Salman, A., Ahmad, I., Al-Madani, S., (2003). Particle Swarm Optimization For Task Assignment Problem, *Microprocessors and Microsystems*, 26, 363-371.

Sauer, J.G. ve Coelho L.S. (2008). Discrete Differencial Evolution with Local Search to Solve the Traveling Salesman Problem: Fundamentals and Case Studies. *IEEE Xplore Digital Library*, s:1-6.

Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., Wang, Q.X.(2007). Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters* 103. s: 169–176.

Shi, X.H., Xing, X.L., Wang, Q.X, Zhang, L.,H., Liang, Yang, X.W., Zhou, C.,G., Liang, Y.,C.(2004). A Discrete PSO Method For Generalized TSP Problem. Third *International Conference on Machine Learning and Cybernetics*, Shanghai. S:2378-2383.

Storn R., Price K.,V. (1997). Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, *Journal of Global Optimization* 11: 341–359.

Synder, L. V., Daskin, M. S.,(2006). A random key genetic algorithm for the generalized travelling salesman problem, European Journal of Operation Research 174, 38-53

Şanyılmaz, M. (2006). Deney Tasarımı ve Kalite Geliştirme Faaliyetlerinde Taguçhi Yöntemi ile Bir Uygulama. *Yayınlanmamış Yüksek Lisans Tezi*. Dumlupınar Üniversitesi. s:2-9.

Tasgetiren, M. F. Liang, Y. C., (2003). A Binary Particle Swarm Optimization Algorithm For Lot Sizing Problem, *Journal of Economic and Social Research*, 5, 2, 1-20.

Tasgetiren, M. F., Liang, Y. C., Sevkli, M., Gencyilmaz. G., (2004a). Particle Swarm Ptimization Algorithm For Makespan And Maximum Lateness Minimization İn Permutation Flowshop Sequencing Problem, *Proceedings, 4th International Symposium on Intelligent Manufacturing Systems IMS2004*, Sakarya, Turkey, s:431-441.

Tasgetiren, M. F., Sevkli, M., Liang, Y. C., Gencyilmaz. G., (2004b). Particle Swarm Optimization Algorithm For Permutation Flowshop Sequencing Problem, *Proceedings, 4th International Workshop on Ant Colony Optimization and Swarm Intelligence* (ANTS2004), LNSC 3172, Brussels, s:382-390, Belgium.

Tasgetiren, M. F., Sevkli, M., Liang, Y. C., Gencyilmaz. G., (2004c). Particle Swarm Optimization Algorithm For Single Machine Total Weighted Tardiness Problem, *Proceedings, 2004 Congress on Evolutionary Computation (CEC'04)*, s:1412-1419, Portland, Oregon.

Taşgetiren, M.F., Liang, Y.C., Pan, Q.K., Suganthan, P.(2009a). Discrete/Binary Approach. C.G. Onwubolu, ve D. Davendra, (Ed.), *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*. (V.175) içinde (s:139-162). Berlin: Springer

Taşgetiren, M.F., Chen, A., Gençyılmaz G., Gattoufi, S.(2009b). Smallest Position Value. C.G. Onwubolu, ve D. Davendra, (Ed.), *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization.* (V.175) içinde (s:121-138). Berlin: Springer.

Tasgetiren, M., Suganthan, P., Pan, Q.K., (2010). An ensemble of discrete differantial evolution algorithms for solvining the generalized traveling salesman problem, *Applied Mathematics and Computation*, 215 (9), 3356-3368.nbg

Tsai H.K., Yang J.M., Tsai Y.F., and Kao C.Y.(2004). An Evolutionary Algorithm for Large Traveling Salesman Problems, *IEEE Transactions On Systems, Man, And Cybernetics—Part B: Cybernetics*, Vol. 34, No. 4, s:1718.

Van den Bergh, F., Engelbecht, A. P., (2000). Cooperative Learning İn Neural Networks Using Particle Swarm Optimizers, *South African Computer Journal*, vol.26, s:84-90.

Yoshida, H., Kawata, K., Fukuyama, Y., Nakanishi, Y., (2000). A Particle Swarm Optimization For Reactive Power And Voltage Control Considering Voltage Security Assessment, *IEEE Transactionson Power Systems*, vol.15, s:1232-1239.

Wang, K., Huang, L., Zhou, C.,G., Pang,W.(2003). Particle Swarm Optimization For Traveling Salesman Problem. *Second International Conference on Machine Learning and Cybernetics*. s: 1583-1585.

Wang, C., Zhang, J., Yang, J., Hu,C., Liu,J.(2005). A Modified Particle Swarm Optimization Algorithm and its Application For Solving Traveling Salesman Problem. *Neural Networks and Brain, ICCN&B'05 International Conference*. s:689-694.