BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ENDÜSTRİ MÜHENDİSLİĞİ ANABİLİMDALI
ENDÜSTRİ MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

OTEL SEÇİMLİ GEZGİN SATICI PROBLEMİ İÇİN KARŞIT ÇÖZÜM TEMELLİ DEĞİŞKEN KOMŞU İNİŞ SEZGİSELİ

YÜKSEK LİSANS TEZİ

HAZIRLAYAN

İPEK DAMLA AKPINAR

ANKARA-2020

BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ENDÜSTRİ MÜHENDİSLİĞİ ANABİLİMDALI
ENDÜSTRİ MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI

OTEL SEÇİMLİ GEZGİN SATICI PROBLEMİ İÇİN KARŞIT ÇÖZÜM TEMELLİ DEĞİŞKEN KOMŞU İNİŞ SEZGİSELİ

YÜKSEK LİSANS TEZİ

HAZIRLAYAN

İPEK DAMLA AKPINAR

TEZ DANIŞMANI

DR. ÖĞR. ÜYESİ BARIŞ KEÇECİ

ANKARA-2020

BAŞKENT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

Endüstri Mühendisliği Anabilim Dalı Endüstri Mühendisliği Tezli Yüksek Lisans Programı çerçevesinde İpek Damla AKPINAR tarafından hazırlanan bu çalışma, aşağıdaki jüri tarafından Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Savunma Tarihi: 14 / 01 / 2020

Tez Adı: Otel Seçimli Gezgin Satıcı Problemi İçin Karşıt Çözüm Temelli Değişken Komşu İniş Sezgiseli

Tez Jüri Üyeleri (Unvanı, Adı - Soyadı, Kurumu) **İmza**

Prof. Dr. Fulya ALTIPARMAK, Gazi Üniversitesi .....

Dr. Öğr. Üyesi Barış KEÇECİ, Başkent Üniversitesi .....

Dr. Öğr. Üyesi Tusan DERYA, Başkent Üniversitesi .....

ONAY

Prof. Dr. Ömer Faruk ELALDI
Fen Bilimleri Enstitüsü Müdürü

Tarih: ... / ... / .....

BAŞKENT ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZ ÇALIŞMASI ORJİNALLİK RAPORU

Tarih: 20 / 01 / 2020

Öğrencinin Adı, Soyadı : İpek Damla AKPINAR

Öğrencinin Numarası : 21710301

Anabilim Dalı : Endüstri Mühendisliği Ana Bilim Dalı

Programı : Endüstri Mühendisliği Tezli Yüksek Lisans Programı

Danışmanın Unvanı/Adı, Soyadı : Dr. Öğr. Üyesi Barış KEÇECİ

Tez Başlığı : Otel Seçimli Gezgin Satıcı Problemi İçin Karşıt Çözüm Temelli Değişken Komşu İniş Sezgiseli

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 44 sayfalık kısmına ilişkin, 20 / 01 / 2020 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %4'tür.

Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrencinin İmzası : .....

Onay

20 / 01 / 2020

Dr. Öğr. Üyesi Barış KEÇECİ

TEŞEKKÜR

Çalışmalarım süresince değerli yardımlarını hiçbir zaman esirgemeyen, fikir ve yönlendirmeleriyle akademik kariyerime büyük katkı sağlayan tez danışmanım değerli hocam Sayın Dr. Öğr. Üyesi Barış KEÇECİ'ye sonsuz teşekkürlerimi sunarım.

Başkent Üniversitesi Endüstri Mühendisliği Bölümü öğretim üyeleri ve araştırma görevlisi arkadaşlarıma ve çalışmamın her aşamasında manevi destekleriyle yanımda olan tüm arkadaşlarıma çok teşekkür ederim.

Hayatımın her aşamasında olduğu gibi tez çalışmalarım boyunca da her zaman beni destekleyen ve yanımda olan aileme, yüksek lisans eğitimim süresince bana her zaman destek olan sevgili büyükbabam Prof. Dr. Mustafa KURU'ya teşekkürlerimi sunarım.

ÖZET

İpek Damla AKPINAR

OTEL SEÇİMLİ GEZGİN SATICI PROBLEMİ İÇİN KARŞIT ÇÖZÜM TEMELLİ DEĞİŞKEN KOMŞU İNİŞ SEZGİSELİ

Başkent Üniversitesi Fen Bilimleri Enstitüsü

Endüstri Mühendisliği Anabilim Dalı

2020

Bu tez kapsamında Gezgin Satıcı Problemi (GSP)'nin bir versiyonu olan Otel Seçimli Gezgin Satıcı Problemi (OSGSP) ele alınmıştır. GSP, aralarındaki uzaklıkları bilinen noktaların her birine yalnız bir kez uğramak şartıyla, başlangıç noktasına dönen en az maliyetli turun bulunması problemidir. OSGSP ise günlük çalışma süresi/mesafesi kısıtı içermektedir. Bu kısıt, tüm noktaların bir seferde ziyaret edilmesini mümkün kılmamaktadır. Bu nedenle gezgin, gün (kısıtlı çalışma süresi/mesafesi) sonunda uygun bir bekleme (otel) noktasında duraklar (konaklar) ve bir sonraki gün, tura kaldığı otel noktasından başlayarak devam eder. Her otele uğrama zorunluluğu bulunmamakla birlikte bir otele birden fazla kez de uğranabilmektedir. Bir otelde başlayıp yine bir otelde son bulan sıralı noktalar kümesine "gezi", tüm müşterileri kapsayan sıralı geziler kümesine "tur" denmektedir. Problemde birincil amaç tur içinde yapılan gezi sayısını en küçüklemektir. İkincil amaç ise gezi başına geçen süre/mesafenin günlük çalışma süresi kısıtını aşmaması şartı ile turun toplam mesafesini en küçüklemektir. Tez çalışmasında, bu problemin çözümü için Değişken Komşu İniş algoritmasına dayalı bir algoritma geliştirilmiştir. En Yakın Komşu Prensibi (EYKP) ile başlangıç çözüm elde edilmiş ve bu çözümü iyileştirmek amacıyla Karşıt Çözüm Temelli Değişken Komşu İniş Sezgiseli (KÇTDKİS) uygulanmıştır. Elde edilen sonuçlar literatürde var olan diğer sezgisel algoritmalardan elde edilen sonuçlar ile karşılaştırılmıştır. Yapılan analizler sonucunda 120 problemin 35 tanesinde daha iyi sonuçlar elde edilmiştir. Ayrıca sonuçlar, yine literatürde var olan Cplex çözücüsü ile elde edilen sonuçlarla karşılaştırılmış ve 78 problemin 17'sinde ya optimal sonuçlar elde edilmiş ya da bilinen en iyi sonuçlardan daha iyi sonuçlara ulaşılmıştır.

ANAHTAR KELİMELER: Gezgin Satıcı Problemi, Otel Seçimi, Sezgisel Algoritmalar, Değişken Komşu Arama, Karşıt Çözüm Temelli Arama.

ABSTRACT

İpek Damla AKPINAR

OPPOSITION-BASED VARIABLE NEIGHBORHOOD DESCENT ALGORITHM FOR THE TRAVELLING SALESPERSON PROBLEM WITH HOTEL SELECTION

Başkent University, Instutute of Science and Engineering

Department of Industrial Engineering

2020

In this dissertation, The Travelling Salesperson Problem with Hotel Selection (TSPHS), which is a variant of The Travelling Salesperson Problem (TSP), is considered. TSP consist of a salesman and set of nodes known distances between them. The salesman has to visit each one of the nodes starting from a certain one and returning to the same node. The goal is to have the least cost tour during the salesman's trip. TSPHS includes daily working time / distance restriction. This constraint does not allow to visit all points at once. For this reason, the traveler stops (roosts) at a suitable waiting (hotel) point at the end of the day (limited working time / distance) and continues on the next day's trip from the hotel point where he stayed. There is no obligation to visit each hotel, and a hotel can be visited more than once. A set of ordered nodes starting and ending at a hotel is called a trip, and a set of ordered trips covering all nodes is called a tour. The primary objective of the problem is to minimize the number of trips. The secondary objective is to minimize the total distance of the tour, provided that the time / distance per trip does not exceed the daily working time limit. In the dissertation, an algorithm based on Variable Neighborhood Descent algorithm is developed for the solution of this problem. The starting solution with The Nearest Neighbor Principle (NNP) is obtained. Opposition Based Variable Neighborhood Descent (OBVND) Algorithm is applied to improve this solution. The results are compared with the other heuristic algorithms in the literature. As a result of the analysis, better results are obtained in 35 of 120 problem. In addition, the results are compared with those obtained with Cplex solvent, which is also available in the literature. In 17 of 78 problems, either optimal results are obtained or better results are obtained than the best known results.

KEYWORDS: Travelling Salesperson Problem, Hotel Selection, Heuristic Algorithms, Variable Neighborhood Descent, Opposition Based Search.

İÇİNDEKİLER

Sayfa

EKLER

TABLOLAR LİSTESİ

Sayfa

ŞEKİLLER LİSTESİ

Sayfa

SİMGELER VE KISALTMALAR LİSTESİ

m	gezi sayısı
s	otel sayısı
n	müşteri sayısı
$c_{k,l}$	*k*. düğümden *l*. düğüme gitme süresi (mesafesi)
C	hergezi için belirlenen süre kısıtı
T_l	*l.* müşteride geçen zaman
$x_{k,l,d}$	*d.* gezide *k*. düğümden *l*. düğüme gidilmesi durumunda 1, değilse 0 değerini alan karar değişkeni
$x_{i,j}^{d}$	*d.* gezide *i*. düğümden *j*. düğüme gidilmesi durumunda 1, değilse 0 değerini alan karar değişkeni
y^{d}	*d*. gezi gerçekleşmiş ise 1, değilse 0 değerini alan karar değişkeni
ARP	Araç Rotalama Problemi
AT	Ara Tesis
ATPARP	Ara Tesislerle Periyodik Araç Rotalama Problemi
AYTARP	Ara Yenileme Tesisleri ile Araç Rotalama Problemi
CSVG	Castro, Sörensen, Vansteenwegen, Goos
CSVGGG	Gavish and Graves Alt Tur Eleme Kısıtları İle CSVG
ÇDARP	Çoklu Depolu Araç Rotalama Problemi
ÇDDAARP	Depolar Arası Rotalar ile Çoklu Depolu Araç Rotalama Problemi
ç-TSP	Çoklu Gezgin Satıcı Problemi
ÇZP-GSP	Çoklu Zaman Pencereli Gezgin Satıcı Problemi
DAR-ÇDARP	Depolar Arası Rotalar ile Çoklu Depolu Araç Rotalama Problemi
DKİ	Değişken Komşu İniş Sezgiseli
EYKP	En Yakın Komşu Prensibi
FGG	The Fox, Gavish and Graves
GKÇY	Genelleştirilmiş KÇY
GSP	Gezgin Satıcı Problemi
HDM	Hybrid Dynamic Programming and Memetic Search
İDKİ	İndirgenmiş DKİ
KÇTDKİS	Karşıt Çözüm Temelli Değişken Komşu İniş Sezgiseli
KÇY	Karşıt Çözüm Yöntemi
MTÖ	Merkez Tabanlı Öğrenme
OSGSP	Otel Seçimli Gezgin Satıcı Problemi
P-LS	Perturbation Local Search
VSS	Vansteenwegen, Souffriau, Sörensen
VSSGG	Gavish and Graves Alt Tur Eleme Kısıtları İle VSS
YRP	Yer Seçimi-Rotalama Problemi (YRP) (Location-Routing Problem)
ZPATARP	Zaman Pencereli Atık Toplama Aracı Rotalama Problemi

1. GİRİŞ

Günümüzde üretim ve hizmet olmak üzere iki ana sektör vardır. Üretim sektörü inşaat, otomotiv, gıda gibi alanları; hizmet sektörü ise ulaştırma, haberleşme, turizm, bankacılık gibi alanları kapsamaktadır. Her iki sektörde de rekabet koşulları ile baş edebilmek için çalışmalar yapılmaktadır. İhtiyaçlar dahilinde yapılan çalışmalarda zaman ve/veya maliyet açısından avantaj sağlamak hedeflenmektedir.

Araştırmacılar bu amaçla birçok alanda çalışma yapmaktadır. Bu alanlardan biri de rotalama problemleridir. Rotalama problemlerinin temelinde ise Gezgin Satıcı Problemi (GSP) yer almaktadır. Rotalama probleminin çözümünde zaman/maliyet açısından avantaj elde edebilmek için GSP alanında çalışılmıştır. GSP, aralarındaki uzaklıkları bilinen noktaların her birinden yalnız bir kez geçerek, başladığı noktaya dönen en az maliyetli turun bulunması problemidir.

Bu tezin konusu olan Otel Seçimli Gezgin Satıcı Problemi (OSGSP), GSP'nin genişletilmiş bir varyantıdır. GSP'nden farklı olarak çalışma süresi/mesafesi kısıtını içermektedir ve bu kısıt ile birlikte çözümlenmesi daha zor olan bir problem haline gelmiştir.

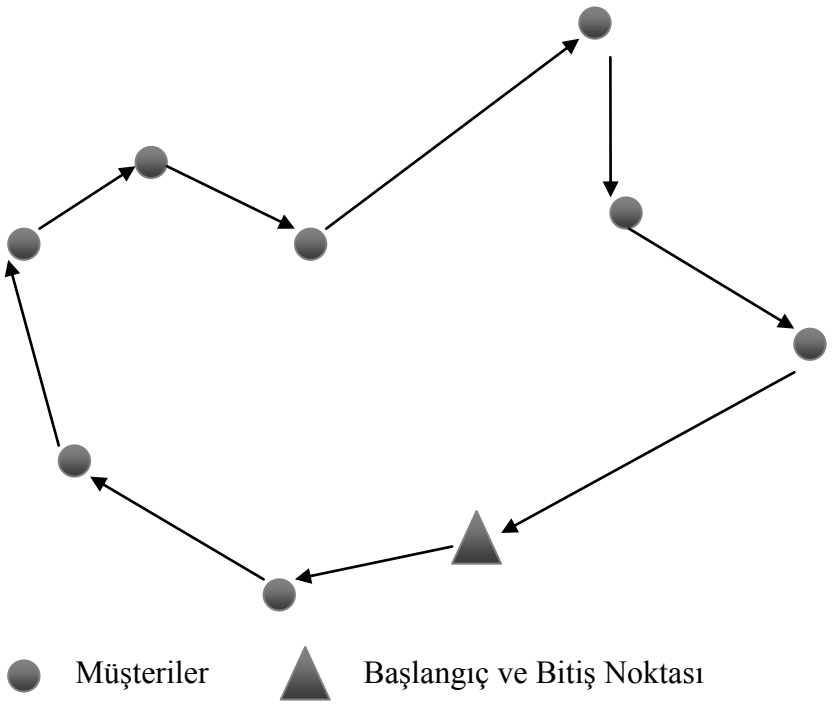
Bu tez kapsamında, OSGSP çözümü için sezgisel bir algoritma geliştirilmiş ve sonuçları literatürde var olan diğer sezgisel algoritmalar ile yine literatürde var olan dört veri seti üzerinden karşılaştırılmıştır. Yapılan analizler sonucunda set 1'de ortalama %3.80, set 2'de ortalama %1.11, set 3'de ortalama %5.50 ve set 4'de ortalama %10.29'luk sapmalar ile sonuçlar elde edilmiştir. Elde edilen bu sonuçlara dört veri seti için de sırasıyla ortalama 5460.01 sn, 6.11 sn, 6540.78 sn ve 6251.46 sn de ulaşılmıştır. Yapılan karşılaştırmalar sonucunda 16 tane set 1 probleminin 2 tanesinde, 52 tane set 2 probleminin 24 tanesinde ve 38 tane set 3 probleminin 9 tanesinde daha iyi sonuçlar elde edilmiştir.

Tez çalışmasında, ilk olarak problemin tanımı yapılmıştır. Sonrasında bu konudaki literatür çalışmalarından bahsedilmiş ve ilerleyen kısımlarda sezgisel algoritma ve karşılaştırmalı sonuçlara yer verilmiştir.

2. GSP, OSGSP VE LİTERATÜR ARAŞTIRMASI

2.1. GSP

GSP'nde bir başlangıç noktasından yola çıkılır, hedeflenen tüm noktalara uğranır ve yine başlangıç noktasına dönülür. Bu işlem "Tur" olarak adlandırılmaktadır. Tur boyunca tüm noktalara uğranması zorunludur. Fakat bir noktaya yalnızca bir defa uğranmalıdır. Standart bir GSP seyahat süresi/mesafesini en küçüklemeyi amaçlamaktadır. Problem için noktalar arasındaki seyahat süresi/mesafesi parametre olarak bilinmektedir. Örnek bir GSP Şekil 2.1.'de gösterilmiştir.



Şekil 2.1. GSP Çözümü Örneği

2.2. OSGSP

OSGSP ise çalışma süresi/mesafesi kısıtını içermektedir ve bu haliyle GSP'nin genişletilmiş bir versiyonunu oluşturmaktadır. OSGSP s adet otelden oluşan otel kümesi ($i=1,...,s$) ile, n adet müşteriden oluşan müşteri kümesini ($i=s+1,...,s+n$) içermektedir. Her i müşterisinden/otelinden j müşterisi/oteline gidiş süresi/mesafesi $c_{i,j}$ ile, her i müşterisi için hizmet süresi ise T_i ile ifade edilmektedir. Gezginin her gün bir otelde/başlangıç noktasında başlayıp, bir otelde/bitiş noktasında sonlanan ziyaretleri "gezi" olarak

adlandırılmakta, tüm gezilerin toplamı ise "tur" olarak adlandırılmaktadır. Her tur *m* adet geziden oluşmaktadır (*d=1,...,m*) [1].

Bu problem için iki tür amaç tanımlanmıştır. Bunlardan ilki toplam turdaki gezi sayısını en küçüklemek, ikincisi ise gezginin gerçekleştirdiği toplam tur uzunluğunu en küçüklemektir. Bu amaçlardan birinci amaç ikinci amaca göre daha önceliklidir. Çünkü gezginin fazladan yapacağı her bir gezi; fazladan otel masrafı, gezginin günlük ücreti vb. masrafları içermektedir. Bu maliyetlerin toplamı ise yol ücreti masrafından daha fazladır. Bu nedenle turda fazladan bir gezi daha olmasındansa, turun uzunluğunun daha fazla olması tercih edilmektedir.

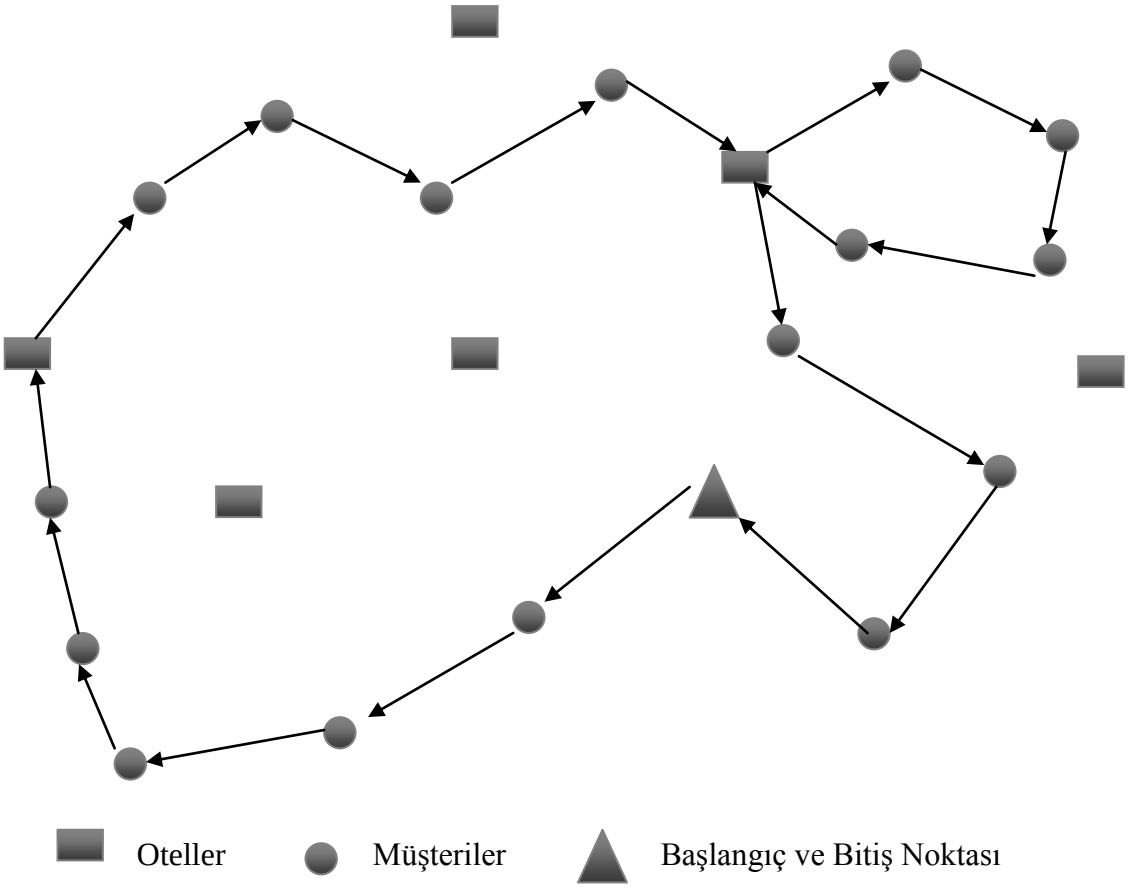
Klasik GSP'nde olduğu gibi OSGSP'nde de her müşteriye uğrama zorunluluğu vardır. Gezgin her noktaya uğradığında yolculuk süresinden ayrı olarak o noktada hizmet vermek için belli bir süre harcamaktadır. Bu sürede günlük çalışma süresine dahil edilmektedir.

OSGSP'nde çalışma süresi/mesafesi kısıtından dolayı tüm noktalar tek bir günde ziyaret edilememektedir. Bu süre sınırlaması *C* ile ifade edilmektedir. Bu nedenle gezgin çalışma süresi/mesafesi kısıtını aşmadan gün sonunda bir bekleme noktasında (otelde) duraklamak (konaklamak) durumundadır. Gezginin seyahati süresince konaklayabileceği otellerin listesi önceden belirlenmiştir. Gezgin gün sonunda en son noktayı ziyaret ettikten sonra konaklayacağı oteli seçerken oteller listesinde bulunduğu noktaya en yakın oteli tercih etmektedir. Ertesi gün konakladığı otelden ziyaret etmesi gereken noktaya doğru yola çıkarak seyahatini sürdürmektedir.

Gezginin yolculuğunda tüm noktalara uğrama kısıtı bulunmasına karşın belirlenen oteller listesindeki her otele uğrama zorunluluğu yoktur. Hatta gerekli görülmesi durumunda gezgin bir otelde birden fazla kere konaklayabilmektedir. Ayrıca turun aynı otelde/başlangıç noktasında başlayıp aynı otelde/bitiş noktasında bitme zorunluluğu vardır. Örnek bir OSGSP Şekil 2.2.'de gösterilmiştir.

Gerçek yaşamda OSGSP olarak modellenebilen bazı problemlere örnek vermek gerekirse: Birden fazla günden oluşan turist gezi programları; ağır vasıta sürücülerinin birden fazla günden oluşan, ve her gün sonunda uygun bir dinlenme noktasında durakladığı tur problemi; bir postacının taşıyacağı yükü hafifletmek amacıyla turunu birbiriyle bağlantılı daha küçük turlara böldüğü dağıtım problemi; maksimum kullanım zamanı pil kapasitesiyle sınırlı elektrikli bir aracın tur problemi.

Bu problemle ilgili ilk matematiksel model Vansteenwegen ve ark. tarafından oluşturulmuştur ve VSS olarak adlandırılmıştır [1]. Bir diğer matematiksel model ise CSVG modelidir [4].



Şekil 2.2. OSGSP Çözümü Örneği

2.2.1. VSS modeli

OSGSP için VSS modeli, sabit sayıda gezi içeren bir karma tamsayılı doğrusal programlama modeli olarak formüle edilmiştir. Bu formülasyon ile mümkün bir çözüm bulunana kadar m değerlerini artırarak çözüme ulaşılmıştır.

Karar değişkeni;

$$x_{k,l,d} = \begin{cases} 1, d. \text{ gezide } k. \text{ düğümden } l. \text{ düğüme gidiliyor ise} \\ 0, \text{diğer durumlarda} \end{cases}$$

Amaç fonksiyonu;

$$Enk \sum_{d=1}^{m} \sum_{k=0}^{s+n} \sum_{l=0}^{s+n} (x_{k,l,d} c_{k,l}) \tag{2.1}$$

Kısıtlar;

$$\sum_{l=0}^{s+n} x_{0,l,1} = \sum_{k=0}^{s+n} x_{k,0,m} = 1 \tag{2.2}$$

$$\sum_{h=0}^{s} \sum_{k=0}^{s+n} x_{k,h,d} = \sum_{h=0}^{s} \sum_{l=0}^{s+n} x_{h,l,d} = 1 \qquad d = 1, \dots m \tag{2.3}$$

$$\sum_{k=0}^{s+n} x_{k,h,d} = \sum_{l=0}^{s+n} x_{h,l,d+1} \qquad d = 1, \dots m-1;\ h = 0, \dots s \tag{2.4}$$

$$\sum_{d=1}^{m} \sum_{k=0}^{s+n} x_{k,i,d} = 1 \qquad i = s+1, \dots s+n \tag{2.5}$$

$$\sum_{k=0}^{s+n} x_{k,i,d} = \sum_{l=0}^{s+n} x_{i,l,d} \qquad d = 1, \dots m;\ i = s+1, \dots s+n \tag{2.6}$$

$$\sum_{k=0}^{s+n} \sum_{l=0}^{s+n} x_{k,l,d} \left(c_{k,l} + T_l\right) \leq C \qquad d = 1, \dots m \tag{2.7}$$

$$u_i - u_j + 1 \leq (n-1)\left(1 - \sum_{d=1}^{m} x_{i,j,d}\right) \quad i = s+1, \dots s+n; j = s+1, \dots s+n \tag{2.8}$$

$$x_{i,j,d} \in 0,1 \qquad d = 1, \dots m; i = 0, \dots s+n; j = 0, \dots s+n \tag{2.9}$$

$$u_i \geq 0, u_j \geq 0 \qquad i = s+1, \dots s+n; j = s+1, \dots s+n \tag{2.10}$$

Amaç fonksiyonu (2.1) toplam seyahat süresini en küçüklemektedir. Problem çözümü için yolculuk sayısı m, uygun bir çözüm bulana kadar her iterasyonda birer

artırılarak probleme parametre olarak girilmiş ve sonuçta problem için en az sayıdaki gezi miktarı belirlenmiştir. Böylelikle m, matematiksel modelin karar değişkeni yerine parametresi haline gelmiştir. (2.2) kısıtı turun başlangıç otelinde başlamasını ve bitmesini sağlamaktadır. (2.3) kısıtı gezilerin bir otelde başlayıp bitmesini garanti etmektedir. (2.4) kısıtı gezi hangi otelde bitmiş ise takip eden gezinin aynı otelde başlamasını sağlamak için oluşturulmuştur. (2.5) kısıtı her müşteriye bir kez uğranmasını sağlar. (2.6) kısıtı müşteriler arasında devamlılığı sağlar. (2.7) kısıtı gezginin çalışma süresi/mesafesini aşmadan gezisini tamamlamasını garanti etmektedir. (2.8) ve (2.10) numaralı kısıtlar alt tur eleme kısıtlarıdır. (2.9) kısıtı ise değişkenin 0 veya 1 değerini almasını sağlar.

2.2.2. CSVG modeli

CSVG modelinde hem gezi sayısını hem de toplam mesafeyi en aza indiren ağırlıklı bir amaç fonksiyonu kullanılmıştır.

Karar değişkenleri;

$$x_{i,j}^{d} = \begin{cases} 1, d. \text{ gezide } i. \text{ düğümden } j. \text{ düğüme gidiliyor ise} \\ 0, \text{diğer durumlarda} \end{cases}$$

$$y^{d} = \begin{cases} 1, d. \text{ gezi gerçekleşti ise} \\ 0, \text{diğer durumlarda} \end{cases}$$

Amaç fonksiyonu;

$$Enk = \sum_{d=1}^{m} c_{i,j} x_{i,j}^{d} + M \sum_{d=1}^{m} y^{d} \qquad (2.11)$$

Kısıtlar;

$$\sum_{d=1}^{m} \sum_{i=1}^{s+n} x_{i,j}^{d} = 1 \qquad j = s+1, \dots s+n \qquad (2.12)$$

$$\sum_{i=1}^{s+n} x_{i,j}^{d} = \sum_{j=1}^{s+n} x_{j,i}^{d} \qquad j = s+1, \dots s+n\,; d = 1, \dots m \qquad (2.13)$$

$$\sum_{h=1}^{s}\sum_{j=1}^{s+n} x_{h,j}^{d} = y^{d} \qquad d = 1, \dots m \tag{2.14}$$

$$\sum_{h=1}^{s}\sum_{i=1}^{s+n} x_{i,h}^{d} = y^{d} \qquad d = 1, \dots m \tag{2.15}$$

$$\sum_{i=1}^{s+n}\sum_{j=1}^{s+n} x_{i,j}^{d}\,(c_{i,j} + T_{j}) \le C \qquad d = 1, \dots m \tag{2.16}$$

$$\sum_{j=2}^{s+n} x_{1,j}^{1} = 1 \tag{2.17}$$

$$\sum_{i=2}^{s+n} x_{i,1}^{d} \ge y^{d} - y^{d+1} \qquad d = 1, \dots m-1 \tag{2.18}$$

$$\sum_{i=1}^{s+n} x_{i,h}^{d} + y^{d} \ge \sum_{i=1}^{s+n} x_{h,i}^{d+1} + y^{d+1} \qquad h = 1, \dots s\,; d = 1, \dots m-1 \tag{2.19}$$

$$\sum_{i=1}^{s+n} x_{i,h}^{d} - \sum_{i=1}^{s+n} x_{h,i}^{d+1} \le 1 - y^{d+1} \qquad h = 1, \dots s\,; d = 1, \dots m-1 \tag{2.20}$$

$$x_{i,j}^{d} \le y^{d} \qquad d = 1, \dots m; i = 1, \dots s+n; j = 1, \dots s+n \tag{2.21}$$

$$y^{d} \ge y^{d+1} \qquad d = 1, \dots m-1 \tag{2.22}$$

$$y^{d} \in 0,1 \qquad d = 1, \dots m \tag{2.23}$$

$$u_i - u_j + 1 \le (n-1)\left(1 - \sum_{d=1}^{m} x_{i,j}^{d}\right) \quad i = s+1, \dots s+n; j = s+1, \dots s+n \tag{2.24}$$

$$u_i \ge 0, u_j \ge 0 \qquad i = s+1, \dots s+n; j = s+1, \dots s+n \tag{2.25}$$

Amaç fonksiyonu (2.11) toplam seyahat süresi ve gezi sayısını en küçüklemektedir. (2.12) kısıtı her müşterinin bir kez ziyaret edilmesini sağlar. (2.13) kısıtı gezi içerisinde devamlılığı garanti eder. (2.14) ve (2.15) kısıtları gezilerin bir otelde başlamasını ve bir otelde sonlanmasını sağlar. (2.16) kısıtı her geziyi süre bakımından kısıtlar. (2.17) ve (2.18) kısıtları turun başlangıç otelinde başlamasını ve bitmesini garanti etmektedir. (2.19) ve (2.20) numaralı kısıtlar bir gezinin sonlandığı otelin bir sonraki gezinin başlangıç oteli olmasını sağlar. (2.21) kısıtı bir müşteri veya otel ziyaret edilir ise ona karşılık gelen gezi değişkeninin 1 değerini almasını amaçlar. (2.22) kısıtı gezilerin ardışık günlerde yapılmasını sağlar. (2.23) nolu kısıt değişkenin 0 veya 1 değerini almasını garanti eder. (2.24) ve (2.25) numaralı kısıtlar alt tur eleme kısıtlarıdır.

Literatürde bu modelleri geliştirmek ve yeni modeller oluşturmak için 2019 yılında bir çalışma yapılmıştır [10]. Bu çalışmada literatürde var olan VSS ve CSVG modellerinin alt tur eleme kısıtları değiştirilerek VSSGG ve CSVGGG modelleri oluşturulmuştur. Ayrıca FGG isminde yeni bir model tasarlanmıştır. Yapılan analizler sonucunda ortalama çözüm süresi dikkate alındığında VSSGG modelinin küçük ölçekli problemlerde, CSVGGG modelinin ise büyük ölçekli problem çözümlerinde daha etkin olduğu gözlemlenmiştir.

2.3. Literatür Araştırması

OSGSP ilk kez Vansteenwegen ve ark. tarafından 2011 yılında ele alınmıştır. Çalışmada CPLEX (10.0) ile 40 düğüme kadar çözüm alınabilen matematiksel bir model oluşturulmuştur. Bu problem seyahat eden satıcı problemine (GSP) çok benzese de otel seçimi gerekliliğinden dolayı çok daha zor bir problemdir. Bu nedenle modelin çözüm zamanı açısından elverişli olmadığı görülmüştür ve çözüm süresini azaltmak için sezgisel bir algoritma tasarlanmıştır [1]. Castro ve ark. [2], daha önce Vansteenwegen ve ark. tarafından önerilen formülasyonu uygulayan yeni bir tamsayı programlama modelini sunmuştur. Ayrıca çalışmada iyileştirilmiş bir başlangıç çözümden yola çıkılarak DKİ algoritması ile sonuçlar elde edilmiş ve karşılaştırmalar yapılmıştır. Gerçek bir hayat problemi Tele Atlas firması için uygulanmıştır [3]. Bu çalışmada coğrafi veri tabanında kullanmak için her bir cadde bir kez ziyaret edilerek yol ve levhalarının fotoğrafları çekilmelidir. Alıntılanan makalede detaylar mevcuttur. Matematiksel model ile elde edilen optimum sonuçlardan daha az sapma ile sonuç elde edebilmek için memetik algoritma içeren bir sezgisel algoritma önerilmiştir [4]. Fakat elde edilen çözümler iyi olsa da bu

algoritmanın çözüm zamanı açısından elverişli olmadığı görülmüştür. OSGSP basit bir uygulama kapsamında, elektrikli araç ile ulaşım ağındaki tüm müşterileri ziyaret ederek optimal yolun bulunması problemi olarak çalışılmıştır [5]. Her seyahatin sonunda, elektrikli aracın, aküsü bitmeden tekrar şarj edilmesi için bir şarj istasyonu bulunmalıdır. Yol, maksimum süresi pil şarjı tarafından sınırlanan yolculuklara bölünebilir. Castro ve ark. tarafından hem çözüm zamanı hem de çözüm kalitesi açısından etkili olan P-LS sezgiseli önerilmiştir [6]. Oluşturulan başlangıç çözümler üzerinden DKİ algoritması ve perturb operatörleri yardımıyla iyileştirmeler yapılmış ve elde edilen çözümler karşılaştırılmıştır. Sousa ve ark. [7] OSGSP için değişken bir komşu arama yöntemi sunmuştur. Hesaplamalı sonuçlar, önerilen algoritmanın, düşük hesaplama süresi içinde daha az seyahate sahip olan ve en iyi bilinen çözümlerle karşılaştırılabilecek toplam seyahat süresine sahip çözümler bulduğunu göstermiştir. Yapılan bir diğer çalışmada OSGSP tropical matematiğin çerçevesinde ele alınmış ve en iyi sonuçlara ulaşılması hedeflenmiştir [8]. Yapılan karşılaştırmalar sonucunda tropical matematik ile en iyi sonuçlara daha kısa sürede ulaşıldığı gözlemlenmiştir. Lu ve Ark. tarafından 2017 yılında [9] dinamik programlama ve memetik algoritmanın birlikte kullanıldığı HDM algoritması geliştirilmiştir. 2019 yılında literatürde var olan matematiksel modeller dışında üç yeni model önerilmiş ve literatürdeki problemler üzerinden bu modellerin performans değerlendirmeleri yapılmıştır [10]. Son olarak ise Sousa ve ark. tarafından yinelemeli yerel arama sezgisellerini temel alan bir yaklaşım geliştirilmiştir [11].

Literatürde OSGSP ile bağlantılı olabilecek çeşitli problemler bulunmaktadır. Çok gezginli satıcı problemi (ç-GSP) bunlardan biridir [12]. Bu problemde birden fazla satış görevlisi vardır. Bu satış görevlileri seyahatlerine aynı depodan başlarlar ve seyahatlerini aynı depoda bitirirler. Araç rotalama probleminde (ARP), [13] belirli kapasiteli araçlar ile toplam kat edilen mesafeyi en küçüklemek amaçlanmaktadır. Bu problemde araçların seyahatlerine tek bir depoda başlayıp, seyahatlerini tek bir depoda bitirme zorunluluğu vardır. Her biri bir araç filosuna sahip çeşitli depoların bulunduğu çok depolu araç rotalama problemi (ÇDARP) [14, 15] ARP'nin farklı bir varyantıdır. Bu problemde de yine araçların aynı depoda seyahate başlayıp aynı depoda seyahati sonlandırma zorunluluğu vardır. OSGSP'nde ise bu durumun aksine satıcı turuna bir otelde başlayıp farklı bir otelde sonlandırabilmektedir. Yer Seçimi-Rotalama problemlerinde (YRP) (Location-Routing Problem) [16] ise depolar önceden sabit değildir. YRP'nde belirli bir depo setinden uygun birtakım depolar seçilir. Araçların, başladığı depoya geri dönmesi şartı ile depolardan çıkarak seçilen depolara gitmesi sağlanır ve depoların toplam kullanım maliyetini en

küçüklemek amaçlanmaktadır. Bu problem için heterojen depolarla alakalı olan [17] ve her aracın birden fazla sefer yapabileceği [18] varyantları çalışılmıştır. GSP, ARP, ÇDARP ve YRP'nin her birinin OSGSP ile ortak yönleri vardır. OSGSP'nin bu problemlerden önemli farkı ise seyahat eden bir satıcının olması ve tüm turların birbirine bağlanması gerekliliğidir.

OSGSP'nde her bir gezi farklı otellerde başlayıp sonlanabilmektedir. Bu da rotanın farklı bir ara tesis (AT)'den başlayıp sonlanmasına imkan verir. Literatürde AT'ler ile bazı yönlendirme sorunları ele alınmıştır. Çoklu Zaman Pencereli GSP problemi (ÇZP-GSP) ilk olarak Pesant ve ark. [19] tarafından ele alınmıştır. Sonrasında Zaman Pencereli Atık Toplama Aracı Rotalama Problemi (ZPATARP) Kim ve ark. [20, 21] tarafından çalışılmıştır. Bu problemde atıklar müşterilerden toplanarak atık berteraf tesislerine götürülürler. Problem aynı zamanda zaman pencereleri, sürücünün öğle yemeği molaları, iş yükü dengeleme, araç kapasitesi ve rota sıkıştırması gibi çeşitli pratik kısıtlamaları içermektedir. Crevier ve ark. [22] AT'ler ile yönlendirme sorunlarını ele almışlardır. Depolar Arası Rotalar ile Çoklu Depolu Araç Rotalama Problemini (DAR-ÇDARP) ortaya koymaktadır. AT ile diğer ilginç yönlendirme problemleri olarak Ara Yenileme Tesisleri ile Araç Rotalama Problemi (AYTARP) [23] ve Ara Tesislerle Periyodik Araç Rotalama Problemi (ATPARP) [24] ele alınmıştır. AT'ler OSGSP ile bazı benzerlikler göstermesinin yanı sıra ikisi arasında farklılıklarda vardır. Bu farklılıklar, (i) her bir seyahatin süresi bir zaman bütçesi ile sınırlandırılmıştır. AT'ler de ise rota uzunluğu araç kapasitesi ile sınırlandırılmıştır ve rota uzunluğunda açık bir zaman kısıtı bulunmaz. (ii) OSGSP'nde birincil amaç toplam kat edilen mesafeyi en aza indirmekten ziyade tur sayısını en küçüklemektir.

OSGSP'nin zaman pencereli OSGSP ve otel seçimli oryantiring problemi gibi çeşitleri vardır. Zaman pencereli OSGSP'nde, OSGSP'ne ek olarak her noktanın ziyaret edilmesi için belli bir zaman aralığı vardır ve gezginin o zaman aralığında o noktayı ziyaret etmiş olması gerekmektedir. Baltz ve ark. tarafından 2014 yılında bu problem ele alınmıştır [25]. Problem çözümü için matematiksel bir model kurulmuş ve sezgisel bir algoritma geliştirilmiştir. OSGSP ile ilgili bir başka çalışmada Souffriau ve arkadaşları [26] tarafından sunulmuştur. Bu çalışmada bir turist bir bölgeye çok günlük bir ziyaret yapmaktadır ve bu bölgedeki cazibe yerlerini ziyaret etmek istemektedir. Bu problemde farklı olarak toplam tur mesafesini en küçüklemek yerine belirli turistik yerleri ziyaret ederek alınan toplam puanı en büyük yapmak amaçlanmıştır. Otel seçimli oryantiring probleminde ise tüm noktaların ziyaret edilme zorunluluğu yoktur. Ziyaret edilecek

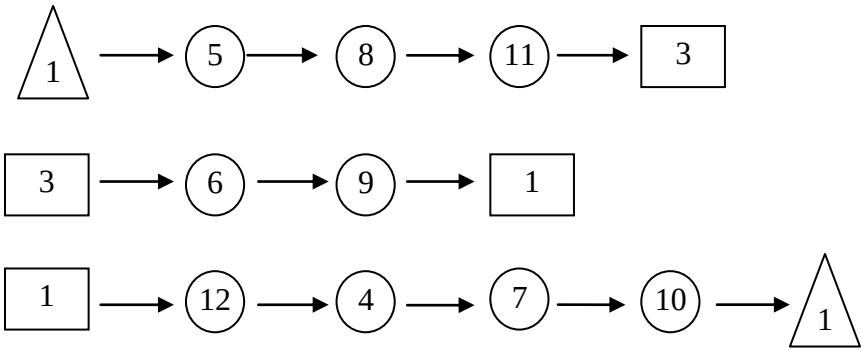
noktalar toplam fayda en büyüklenecek şekilde belirlenir. Bu problem ilk olarak Divsalar ve Ark. [27] tarafından 2014 yılında çalışılmıştır ve memetik bir algoritma geliştirilmiştir. 2015 yılında hipersezgisel bir algoritma önerilmiştir [28]. Problem çözümünde iki sezgisel yöntem çözüm kalitesi açısından karşılaştırılmıştır [29]. Sonuç olarak Tabu Arama algoritmasının çözüm zamanı ve kalitesi açısından daha iyi sonuç verdiği gözlemlenmiştir. Yapılan diğer bir çalışmada, literatürde var olan modeller dışında iki tane daha model önerilmiş ve bu modeller performans kriterlerine göre değerlendirilmiştir [30].

3. OSGSP İÇİN SEZGİSEL ALGORİTMALAR

Literatürde OSGSP'ni çözmek için birçok yöntem önerilmiştir. Bunlar içinde lineer programlama, dinamik programlama, dal-sınır algoritması gibi kesin çözüm veren yöntemlerin yanı sıra; komşuluk mekanizmaları, tabu arama algoritması, genetik algoritmalar, melez yöntemler gibi sezgisel yaklaşımlar da önerilmiştir.

OSGSP, NP-zor problem sınıfına girdiği için çözümde kullanılan matematiksel modeller kesin çözümler elde etmesine rağmen çözüm zamanı açısından avantajlı bulunmamıştır. Bu nedenle özellikle büyük boyutlu problemlerde çözüm yöntemi olarak daha kısa sürede sonuç veren sezgisel yöntemler önerilmiştir. Bu tez kapsamında da OSGSP çözüm yöntemi olarak sezgisel bir yaklaşım geliştirilmiştir.

Önerilen sezgisel algoritma da permütasyon çözüm gösterimi benimsenmiştir. Bu gösterime göre her bir satır bir geziyi ifade etmektedir. Tüm satırların toplamı ise turu oluşturmaktadır. Yani bir turda satır sayısı kadar gezi sayısı vardır. Her bir satırın sonundaki otel ile ardışık satırın başındaki otel aynı oteli temsil etmektedir. Turun başlangıç ve bitiş noktası da aynı noktayı ifade etmektedir. OSGSP için küçük boyutlu çözüm gösterim örneğine şekil 3.1.'de yer verilmiştir. Bu çözüm gösterim örneğine göre, gezgin başlangıç noktasından (1) turuna başlamakta ve otel sayısının 3 (s = 1, 2, 3), müşteri sayısının 9 (n = 4, 5, 6, 7, 8, 9, 10, 11, 12) olduğu bir problem için; toplamda 3 (m = 3) gezi yaparak tekrar başlangıç noktasına dönmüş (1) ve turunu tamamlamıştır.



Şekil 3.1. OSGSP için Çözüm Gösterimi Örneği

Bu algoritma da çözüme ulaşılırken ilk olarak En Yakın Komşu Prensibi (EYKP) kullanılarak başlangıç çözüm elde edilmiştir. Sonrasında bu çözümü iyileştirmek amacıyla Karşıt Çözüm Temelli Değişken Komşu İniş Sezgiseli (KÇTDKİS) kullanılmıştır. Bu algoritma uygulanırken eldeki çözümü iyileştirmek amacıyla "ekleme (insertion/shift)" ve

"takas (swap)" komşuluk mekanizmaları ile "otel iyileştirme (changeHotels)" hareketi kullanılmıştır. "Gezi azalt (joinTrips)" hareketi ile de turdaki gezilerin azaltılması hedeflenmiştir. Sonrasında daha fazla çözüm olasılığını incelemek adına "değiş tokuş (exchange)" hareketi kullanılmıştır. Böylelikle optimal sonuca daha yakın sonuçlar bulmak hedeflenmiştir. Daha iyi bir çözüme ulaşılamadığı noktada algoritma sonlandırılmıştır.

3.1. EYKP

EYKP 11. yüzyıla dayanan bir yöntemdir. Günümüzde birçok modern yöntem, tahmin problemlerinde EYKP'ni içinde barındıran yöntemler kullanmaktadır. Bu yöntemin yıllar boyunca popüler kalmasının ampirik başarılarıyla ilgisi vardır. Bu denli başarılı olması yöntemin dört özelliğine dayandırılabilir. İlki, "yakın" terimini ne anlama geldiğinin seçilmesindeki esnekliktir. Bu esneklik yöntemde derin sinir ağları veya karar ağacı temelli öğrenme yaklaşımları gibi uzaktan öğrenim mekanizmalarından yararlanılmasına izin verir. Mesela eğer bu terim mesafe işlevi olarak kullanılacaksa ortak bir seçim olarak öklid mesafesi kullanılabildiği gibi, daha ayrıntılı seçimler yapılarak zaman serileri için çalışılırken mesafe işlevi doğrusal olmayan bir zaman atlama içerebilir. Eğer bu terim "iyi" olarak tanımlanmış ise derin sinir ağları kullanan veri yöntemine bağlı olarak el ile tasarlanabilir [31]. İkincisi, EYKP'nin hesaplama etkinliği, modern uygulamalarda yüksek boyutlu veri kümelerinde kullanılmasını yaygınlaştırmıştır. Yöntem arama alanındaki hangi veri noktalarının birbirine yakın olduğunu hızla belirleyebilmektedir. Üçüncüsü, bu metot parametrik değildir. Yani en yakın komşu yöntemi veriler için temel model üzerinde çok az varsayım yapar. Bu da asosyal ağlar, sağlık hizmetleri gibi yapısını önceden belirleyemediğimiz büyük verileri analiz ederken verilerin altında yatan yapının açıkça ifade edilmesi sorununu ortadan kaldırır. Parametrik olmayan yöntemler verilerin daha doğrudan tahminlere dayanmasını sağlar. Fakat parametrik olmayan yöntem olması en yakın komşu yöntemlerinin hiçbir parametresi olmadığı anlamına gelmez. Son olarak, EYK metotları yorumlanabilir (interpretability). Uygulayıcı, seçilen örnek uzayının ve seçilen mesafe fonksiyonunun uygulama için yeterli olup olmadığını teşhis etmek için bulunan en yakın komşuları kullanabilir [32].

En yakın komşu prosedürü, bilgisayar bilimlerinde hızlı bir şekilde her bölgeye yayılmıştır. Cover ve Hart'ın 1967 [33] yılında yayınladığı çalışmadan kısa süre sonra Knuth [34] Bilgisayar Programlama Sanatı adlı çalışmasının üçüncü cildinde EYKP'ni konu almış ve bu problemi postane problemi olarak ortaya koymuştur. Cover yaptığı

çalışmada en yakın komşu prosedürü olarak nitelendirilen bir şema kullanmıştır [35]. Günümüze kadar da bu yöntem etkin bir şekilde birçok çalışmada kullanılmıştır ve halen de kullanılmaktadır.

Tezde, başlangıç aşamasında çözüm elde etmek için öncelikli olarak bir başlangıç çözüme ihtiyaç duyulmuştur. Bu başlangıç çözümü elde edebilmek için En Yakın Komşu Prensibi (EYKP) kullanılmıştır. Bu prensibin temelinde gidilecek noktaya karar verilirken mevcut konuma en yakın olan noktanın tercih edilmesi düşüncesi vardır. Buna göre gezgin başlangıç noktasından gezisine başlamakta, sonrasında gideceği noktayı belirlerken kendine en yakın olanı seçmektedir. Bu seçim yapılırken günlük süre/mesafe kısıtı aşılmadan gezi sonunda gidilecek otel de hesaba katılmıştır. Yani bir sonraki adımda gidilecek müşteriye karar verilmeden önce her bir müşteri için en yakın otel belirlenir. En yakın olan müşteri ataması yapılırken müşteri daha önceden belirlenen oteliyle beraber seçilir ve bu şekilde günlük süre/mesafe kısıtını aşıp aşmadığı kontrol edilmektedir. Eğer kısıt aşılmıyor ise sadece müşteri geziye eklenmekte bir sonraki müşteri seçiminde yine aynı yol izlenmektedir. Bu şekilde günlük süre/mesafe kısıtı da aşılmadan gezgin, gezi içerisindeki son noktayı da ziyaret ettikten sonra konaklayacağı otele karar verirken de aynı prensipten yararlanmaktadır. Böylelikle başlangıç çözümündeki her bir gezi tamamlanmakta ve bu gezilerin toplamı turu oluşturmaktadır.

3.2. Karşıt Çözüm Yöntemi

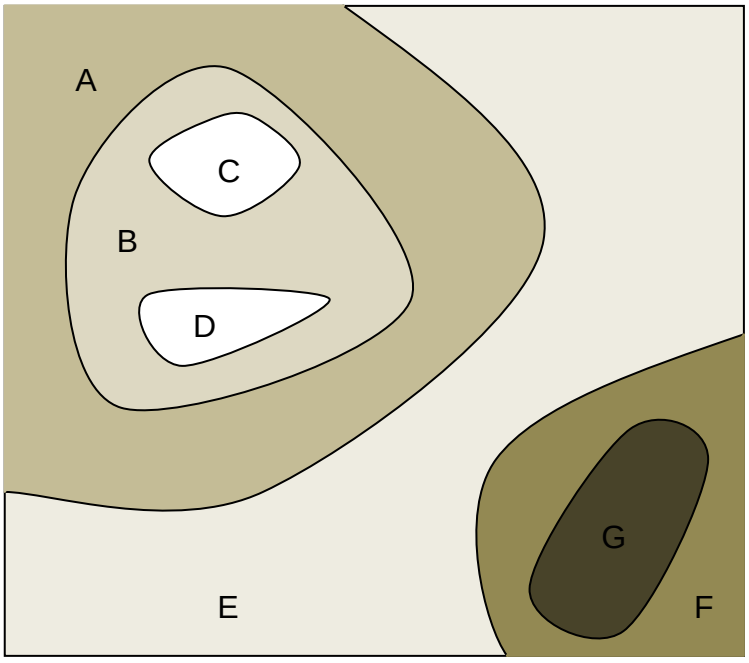
Bu tez çalışmasında elde edilen başlangıç çözümü iyileştirmek için bir algoritma tasarlanmıştır. Karşıt çözüm yöntemi (KÇY) de bu algoritma içerisinde farklı çözüm alanlarındaki çözümleri araştırmaktadır. Bu yöntem, çözümlerden karşıt adaylar elde etmeyi sağlar. Karşıt adayların çözüme dahil edilmesi arama sürecinde daha fazla bölgenin ziyaret edilmesi ile arama sürecinde odaklanmaya veya yayılmaya izin verilmesini sağlamaktadır. Amaç büyük boyutlu problemlerde daha kısa sürede sonuca ulaşmayı sağlamak ve karmaşık problemler için daha güçlü bir çözüm yöntemi önermektir.

Yöntem için karşıt kelimesinin tam olarak neyi ifade ettiğine dair çeşitli tanımlamalar vardır. Optimizasyon problemleri üzerine kurulmuş olan KÇY'nde karşıt kelimesi anlamı bir çift aday çözüm arasındaki ilişki olarak tanımlanmıştır. Bu problemler için her çözümün kendine özel tanımlanmış bir karşıt adayı vardır. Genel olarak karşıt aday çözümler kullanılırken iki farklı yol izlenir. Bunlardan ilki arama alanındaki her çözümün

karşıt aday çözümünü belirlemek için bir işlev geliştirmek, ikincisi ise karşıt kalitede çözümler aramayı sağlamaktır.

KÇY ilk kez 2005 yılında Tizboosh [36] tarafından tanımlanmıştır. Bu çalışmada bir dizi çözümden tamamlayıcı adaylar elde edilmiştir. Çözümlerin tamamlayıcı adayları karşıt olarak adlandırılmış ve arama alanının doğruluğunu artırmak için çözümler haritalandırılmıştır [37]. Buradaki amaç, karşıt çözümü olan daha fazla aday çözüm ile algoritmanın etkinliğini artırmaktır.

Literatürde M-boyutlu bir çözüm uzayında değişkenler dikkate alınarak, karşıt çözümler elde etmek için farklı KÇY tipleri sunulmuştur. Bunlardan ilki Tip-I KÇY'dir. Bu yöntemde arama uzayındaki her çözüm ile eşleşen bir karşıt çözüm vardır. Şekil 3.2.'te bununla ilgili örnek bir gösterime yer verilmiştir [38]. Şekilde A'dan G'ye kadar adlandırılan yedi bölge vardır. Bu bölgelerden koyu renkli olanlar açık renkli olanlara göre daha iyi kalitede çözümler vermektedir. Bir aday çözümün şekildeki en kötü kalitede çözüm veren D alanında olduğunu varsayalım. Eğer bu çözümün karşıt aday çözümü B alanında ise çözüme bu alandan devam etmemiz bize çözüm zamanı ve kalitesi açısından avantaj sağlayacaktır. Çünkü şekilde B alanı D alanına göre daha iyi kalitede çözümler veren bir alandır. Tip-I Yarı-Karşıt ve Tip-I Süper-Karşıt yöntemler mesafelere göre tanımlanan yöntemlerdir [39].



Şekil 3.2. KÇY Örnek Gösterimi

Tip-II KÇY ise düşük kaliteli çözüm alanlarını ayrıştırmak için kullanılmıştır. Mesela Şekil 3.2'de bulunan aday çözümün E uç bölgesinde olduğunu varsayalım. Bu çözümün karşıt çözümü D bölgesinde ise KÇY ile daha kötü bölgedeki bir çözüm bulunmuş olur. Böylelikle çözüm uzayındaki kalitesiz çözüm veren alanların ayrıştırma işlemi yapılabilir. Genelleştirilmiş KÇY (GKÇY) [40] daha önce Uzay Dönüşüm Araması olarak da bilinen KÇY'nin bir diğer versiyonudur. GKÇY'de bir aday çözümün karşıt çözümü bulunurken bir ağırlık parametresi tanımlanmakta ve işlemler buna göre yapılmaktadır. Merkez Tabanlı Öğrenme (MTÖ) [41] ise GKÇY benzer bir KÇY çeşitidir. Amaç her bir çözüm adayı alanının merkezine daha yakın karşıt aday çözümlerin elde edilmesidir. KÇY'nin diğer versiyonları da karşıt çözümün merkezden yansıtma ile bulunduğu Yarı-Yansıtmalı KÇY [42] ve GKÇY'ye benzer olan Optimum Akımlı KÇY'dir [43].

Tip-I KÇY ile var olan bir noktanın tersi (3.1) formülü ile hesaplanır. Bu formüle göre x elde var olan noktayı, $\bar{x}$ ise eldeki noktanın karşıtını göstermektedir. Formülasyonda yer alan a ve b ise sırasıyla çözüm uzayındaki en büyük ve en küçük noktaları ifede etmektedir.

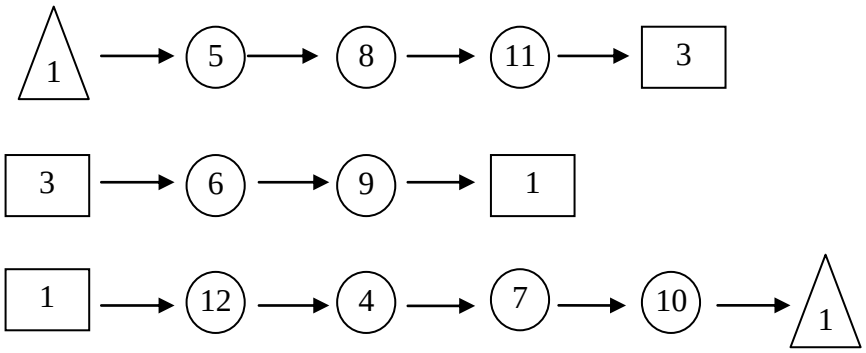
$$\bar{x} = a + b - x \tag{3.1}$$

Literatürde çizelgeleme problemi ve KÇY'nin bir arada kullanıldığı bir çalışmada, Tip-I KÇY kullanılmıştır [44]. Bu çalışmada Tip-I KÇY kullanılırken karşıt çözümlerin hesaplanması (3.2) formülü ile yapılmıştır. Bu formüle göre x_i, M boyutlu bir çözüm uzayındaki noktayı; $\bar{x_i}$, ise bu noktanın karşıtını ifade etmektedir. a_i ve b_i noktaları ise M boyutlu çözüm uzayındaki en küçük ve en büyük noktaları göstermektedir.

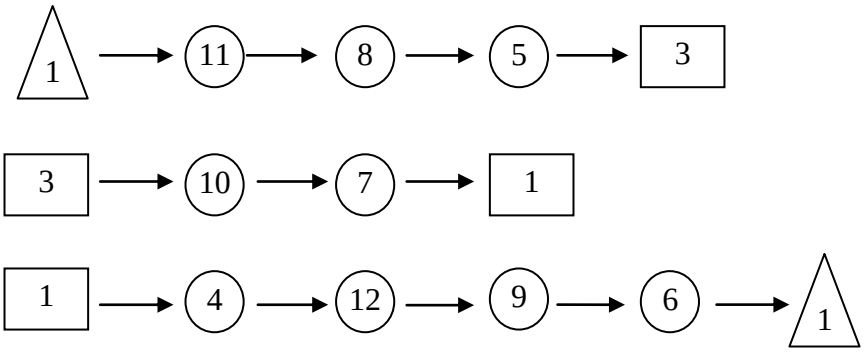
$$\bar{x_i} = a_i + b_i - x_i \tag{3.2}$$

Bu tez çalışmasında Tip-I KÇY kullanılmıştır. Bunun bir kaç nedeni vardır. Bunlardan ilki tezde sürekli bir çözüm uzayı ile çalışılmamıştır yani kesikli bir çözüm uzayı ile çalışılmıştır. Bir diğeri ise yukarıda bahsedilen çalışma ile tezdeki problemin benzerliğinden dolayı o çalışmada tercih edilen KÇY bu tez içinde daha uygun bulunmuştur. Bir diğer sebebi ise bu tez kapsamında çalışılan problemin kısıtları arasında her müşterinin ziyaret edilmesi zorunluluğu vardır. Bu kısıt sadece Tip-I KÇY kullanılır ise sağlanmaktadır. Tip-I KÇY'nin bu tez kapsamında nasıl kullanıldığı daha ayrıntılı biçimde şekil 3.3. ve şekil 3.4. de açıklanmıştır. Şekil 3.3., 9 müşteri ve 3 otele sahip OSGSP için uygun bir çözümü ifade etmektedir. Bu çözüme göre turda üç tane gezi

mevcuttur. Şekil 3.4.'te ise, şekil 3.3'te bulunmuş olan uygun çözümün Tip-I KÇY ile karşıt çözümü gösterilmiştir. Karşıt çözüm oluşturulurken her bir müşterinin karşıtı (3.2) formülü kullanılarak bulunmuştur. Algoritma içerisinde bulunan her bir karşıt çözümün mutlaka uygunluk kontrolü yapılmaktadır.



Şekil 3.3. OSGSP İçin Uygun Çözüm Gösterimi



Şekil 3.4. OSGSP İçin Karşıt Çözüm Gösterimi

Algoritma içerisinde kullanılan bu KÇY çeşitlendirme amaçlı kullanılmaktadır. Yani bu yöntem ile algoritma içerisinde bir çözüm alanında arama yaparken farklı bir çözüm alanında arama yapma imkanı elde edilmiştir. Böylelikle farklı alanlarda çözümler aranarak sonuçların daha kısa sürede daha da iyileştirilmesi amaçlanmaktadır.

Bu tez çalışmasında problemin yapısına sadece bir tane KÇY uygun bulunmuştur. Bu nedenle sadece Tip-I KÇY kullanılmıştır. Fakat diğer KÇY varyantlarının da kullanılanıma uygun olduğu problemler için, algoritma içine birden fazla KÇY tipi de dahil edilebilir. Böylelikle daha geniş bir arama alanında çalışılır ve daha çeşitli çözüm alanlarında arama gerçekleştirildiği için daha iyi çözümler elde edilir.

3.3. Değişken Komşu İniş Sezgiseli

Günümüzde birçok alanda NP-zor problemlerle çalışılmaktadır. Bu problemler için makul bir sürede kesin çözümler bulmak pek mümkün olmaz. Bu nedenle çoğu zaman hızlı ve yaklaşık çözümler elde edilebilecek sezgisel algoritmalar kullanılır. Değişken Komşu İniş (DKİ) sezgiseli de bu sezgisel algoritmalardan bir tanesidir. Bu yaklaşım kombinatöryel optimizasyon problemlerinin yaklaşık çözümleri için tasarlanmış, karma tamsayılı, doğrusal olmayan ve karma tamsayılı doğrusal olmayan problemlerin çözümlerinde kullanılan nispeten yeni bir yöntemdir. Sezgiselin ağ tasarımı, konumlandırma teorisi, küme analizi, yapay zeka, araç güzergahı vb. birçok alanda uygulamaları vardır [45].

DKİ Sezgiseli, Hansen ve Mladenovic tarafından ilk defa 1997 yılında önerilmiştir [46]. Tesisler için toplam nakliye maliyetini en aza indirecek şekilde tasarlanmıştır ve etkinliği literatürdeki diğer bazı sezgisellerle karşılaştırılmıştır. DKİ sezgiselinin yerel arama rutininden bahsedilmiş ve yöntem açıklanmıştır [47, 48, 49]. Sezgisel sistematik olarak üç olgu kullanır.

- Komşuluk yapısına ilişkin olarak kullanılan bir asgari şart, bir diğer komşuluk yapısı için şart değildir.
- Global bir minimum çözüm, olası tüm komşuluk yapılarına göre yerel minimumdur.
- Birçok problem için bir veya daha fazla komşuluk bakımından yerel minimum birbirine nispeten daha yakındır.

DKİ sezgiselinin, temel akışı basittir ve çok az sayıda parametre gerektirir. Bu nedenle diğer sezgisellere nazaran daha basit yollarla sonuca ulaşılır. Sezgiselde birbiri ardına kullanılacak hareketler belirlenir. Daha sonra her adımda olabilecek daha iyi çözümler araştırılır. Şekil 3.5. de DKİ algoritması gösterilmiştir. Algoritmada N_k komşuluk mekanizmaları kümesini, K_{max} ise algoritma içerisinde kullanılan komşuluk mekanizması sayısını ifade etmektedir. İlk olarak eldeki çözüme ilk komşuluk hareketi uygulanır, bulunan çözüm x^n ile ifade edilir ve bu işleme daha iyi bir çözüm bulunamayana kadar devam edilir. Her bir iyi çözüm bulunduğunda eldeki çözüm iyi olan çözüm ile değiştirilerek eldeki çözüm güncellenir. Sonrasında artık ilk komşuluk hareketi ile daha iyi bir çözüm bulunamayınca eldeki en iyi çözüme ikinci komşuluk hareketi uygulanır. Eğer bu adımda daha iyi bir çözüm bulunur ise eldeki çözüm güncellenir ve ilk adıma dönülür. Sonrasında eldeki çözüme ilk komşuluk hareketi uygulanır. Bu işleme daha

iyi bir çözüm bulunamayana kadar devam edilir. Bu adımda daha iyi bir çözüm bulunamayınca yine eldeki çözüme ikinci komşuluk hareketi uygulanır eğer iyi bir sonuç bulunur ise bulunan çözüm üzerinden yukarıdaki işlemler tekrarlanır. Eğer daha iyi bir çözüm bulunamaz ise diğer adıma geçilir ve sıradaki komşuluk hareketi eldeki çözüme uygulanır. Eğer bu adımda daha iyi bir çözüm bulunur ise ilk adıma dönülür. Bütün bu adımlar daha iyi bir çözüm bulunamayana kadar devam ettirilir. Her bir adımda bulunan iyi çözümler güncellenir.

Temel DKİ sezgiselinde karşılaşılabilecek bazı sorunlar vardır. Bunlardan ilki sezgiselde kullanılan hareketlerin karmaşıklık derecesinin belirlenmesidir. Eğer hareketler çok fazla temel değişiklik içeriyorsa algoritma çok yavaş işler ve kesin sonuç veren algoritmadan daha uzun sürebilir. Bir diğeri ise DKİ sezgiseli ile ne kadar kesin bir sonuç istendiğine karar verilmesidir. Bu kararlaştırılan hassasiyet derecesi de yine sezgisel algoritmanın çözüm hızına etki eder.

DKİ Algoritması

0 N_k'yı belirle, K_{max}'ı belirle
1 for $k=1$, $k=K_{max}$
2 $x^n = N_k(x)$;
3 if $f(x^n) < f(x)$
4 $x = x^n$;
5 $k = 1$;
6 else
7 $k = k+1$;
8 end;

Şekil 3.5. DKİ Algoritması

Literatürde bu sezgiselin farklı çeşitleri bulunmaktadır. İndirgenmiş DKİ (İDKİ) sezgiseli çok büyük boyutlu problemler için kullanışlıdır. Bu yöntemde maksimum çözüm süresi veya maksimum yineleme sayısı gibi durdurma kriterleri belirlenmektedir. İDKİ sezgiseli bir Monte Carlo yöntemine benzemektedir fakat daha sistematik ve daha iyi bir yöntemdir [50]. Bu yöntem DKİ sezgiseline göre daha hızlı çözümler vermektedir. Temel DKİ yöntemi [51] ise komşulukların deterministik ve stokastik değişimlerini birleştirmektedir. Deterministik kısmı DKİ sezgiselindeki gibidir. Stokastik kısmı ise birinci komşuluktan bir noktanın rastgele seçilmesi ile ifade edilir. Eğik DKİ [52] yöntemi bulunan çözümün uzağındaki alanlarda da daha iyi çözümleri arama imkanı tanıdığından

dolayı büyük ölçekli problemlerde avantaj sağlamaktadır. Değişken Komşuluk Ayrıştırma Algoritmasın da [53] ise problem iki seviyeli bir DKİ şeması ile çözülür. Bu algoritmada ayrıştırılmış daha küçük boyutlu problemler üzerinde çözüm aranır.

Bu tezde başlangıç çözüm elde edildikten sonra çözümü iyileştirmek amacıyla DKİ algoritması içerisinde komşuluk mekanizmaları, otel iyileştirilmesi ve gezi azalt yöntemleri ile değiş tokuş hareketi kullanılmaktadır. Turun komşuluk mekanizmaları ve hareketlerle en iyi faydayı sağlayan olası hareketi dikkate alınmıştır.

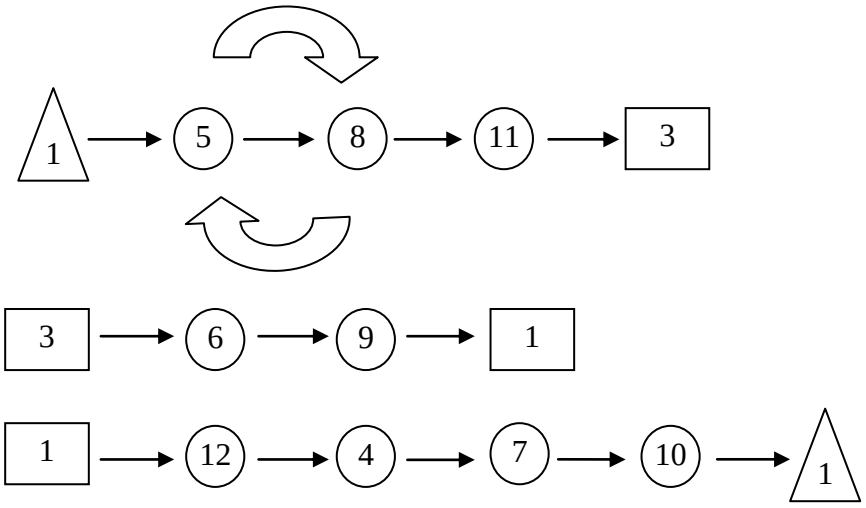
3.3.1. Komşuluk mekanizmaları

Komşuluk mekanizmaları oluşturulan turun toplam süresini/mesafesini azaltmak için kullanılan yöntemlerdir. Bu yöntemler sayesinde gezgin daha kısa sürede/mesafede tüm noktaları ziyaret ederek başlangıç noktasına dönmektedir.

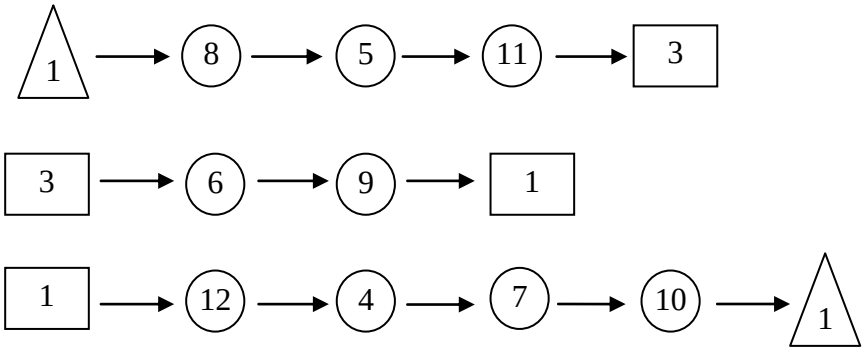
Çözüm yaklaşımında "takas" ve "ekleme" olmak üzere iki tür komşuluk mekanizması kullanılmıştır. Kullanılan her iki komşuluk mekanizması da hem gezi içerisinde hem de geziler arasında noktaların sıralarını ve yerlerini değiştirerek iyileştirmeler yapmaktadır. Bu iyileştirmeler yapılırken günlük süre/mesafe kısıtı da göz ardı edilmemelidir.

Takas komşuluk mekanizmasının temelinde çözüme fayda sağlayacak biçimde iki noktanın birbiri ile değişimi esas alınmaktadır. Yani bu komşuluk mekanizmasında çözümde var olan iki noktanın çözüme fayda sağlamak için yerlerinin birbiri ile değiştirilmesi söz konusudur.

Bu komşuluk mekanizması algoritma içerisinde uygulanırken bir noktanın gezi içerisinde ve geziler arasında tüm yapılabilecek takas harelketleri denenir. Takas işlemleri sonucunda elde edilen çözümlerin uygunluk kontrolleri yapılır ve uygun olan çözümler içerisinden en iyi amaç fonksiyonu değerine sahip olan çözüm seçilir. Seçilen bu çözüm mevcut çözüm ile karşılaştırılır ve eğer daha iyi bir çözüm elde edilmiş ise mevcut çözüm güncellenir. Yapılan bu karşılaştırmalar sonucunda uygulanan takas komşuluk mekanizması ile en iyi çözümü veren takas işleminin gezi içerisinde yapılan, şekil 3.6.'da gösterildiği gibi birinci gezideki 5 ve 8 numaralı müşterilerin değiştirilmesi olduğunu varsayarsak, bu takas hareketi yapılır ve mevcut çözüm güncellenerek şekil 3.7.'deki halini alır. Artık gezgin başlangıç noktasından hareket ettiğinde ilk olarak 8 numaralı müşteriye sonrasında ise 5 numaralı müşeriye uğramak durumundadır.

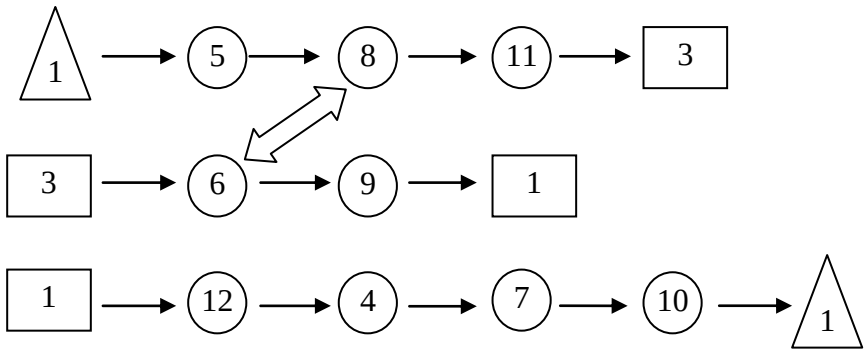


Şekil 3.6. Gezi İçi Takas Komşuluk Hareketi Öncesi

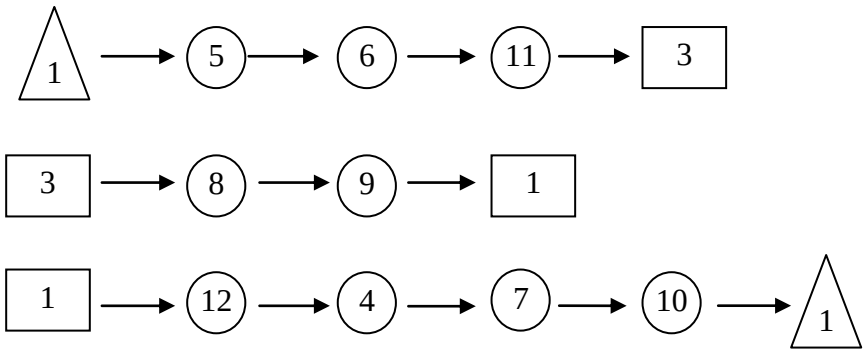


Şekil 3.7. Gezi İçi Takas Komşuluk Hareketi Sonrası

Eğer yapılan karşılaştırmalar sonucunda uygulanan takas komşuluk mekanizması ile en iyi çözümü veren takas işleminin geziler arası yapılan, şekil 3.8.'de gösterildiği gibi birinci gezideki 8 numaralı müşteri ile ikinci gezideki 6 numaralı müşterinin değiştirilmesi olduğunu varsayarsak, bu takas hareketi yapılır ve mevcut çözüm güncellenerek şekil 3.9.'daki halini alır. Artık gezgin birinci gezisinde ikinci sırada 6 numaralı müşteriyi, ikinci gezisinde ise ilk sırada 8 numaralı müşteriyi ziyaret etmek durumundadır.



Şekil 3.8. Geziler Arası Takas Komşuluk Hareketi Öncesi

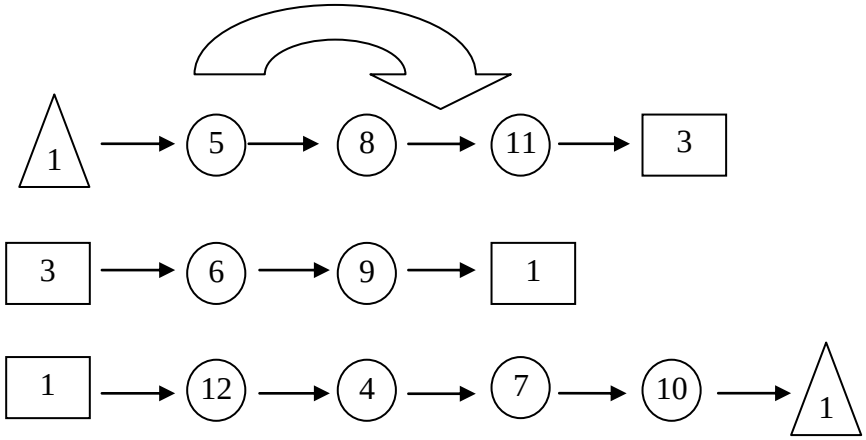


Şekil 3.9. Geziler Arası Takas Komşuluk Hareketi Sonrası

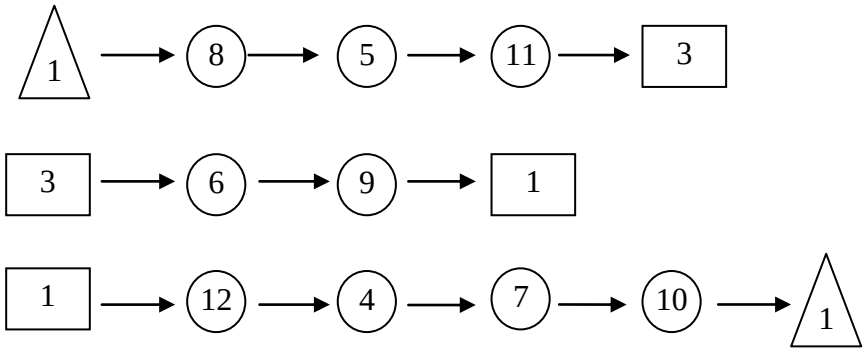
Ekleme komşuluk mekanizmasında ise fayda sağlamak için çözümde var olan bir noktanın yeri değiştirilir. Aslında bu komşuluk mekanizmasında seçilen noktanın taşınması söz konusudur. Yani belirlenen nokta mevcut konumundan alınarak başka bir konumdaki iki noktanın arasına yerleştirilir. Böylelikle başlangıçta bulunan çözümden daha iyi kalitede bir çözüm elde edilmiş olur.

Bu komşuluk mekanizması algoritma içerisinde uygulanırken bir noktanın gezi içerisinde ve geziler arasında tüm yapılabilecek ekleme harelketleri denenir. Ekleme işlemleri sonucunda elde edilen çözümlerin uygunluk kontrolleri yapılır ve uygun olan çözümler içerisinden en iyi amaç fonksiyonu değerine sahip olan çözüm seçilir. Seçilen bu çözüm mevcut çözüm ile karşılaştırılır ve eğer daha iyi bir çözüm elde edilmiş ise mevcut çözüm güncellenir. Yapılan bu karşılaştırmalar sonucunda uygulanan ekleme komşuluk mekanizması ile en iyi çözümü veren ekleme işleminin gezi içerisinde yapılan, şekil 3.10.'da gösterildiği gibi birinci gezideki 5 numaralı müşterilerinin, 8 ile 11 numaralı

müşterilerin arasına taşınması olduğunu varsayarsak, bu ekleme hareketi yapılır ve mevcut çözüm güncellenerek şekil 3.11.'deki halini alır. Artık gezgin başlangıç noktasından hareket ettiğinde ilk olarak 8, sonrasında 5 ve 11 numaralı müşterilere uğramak durumundadır.

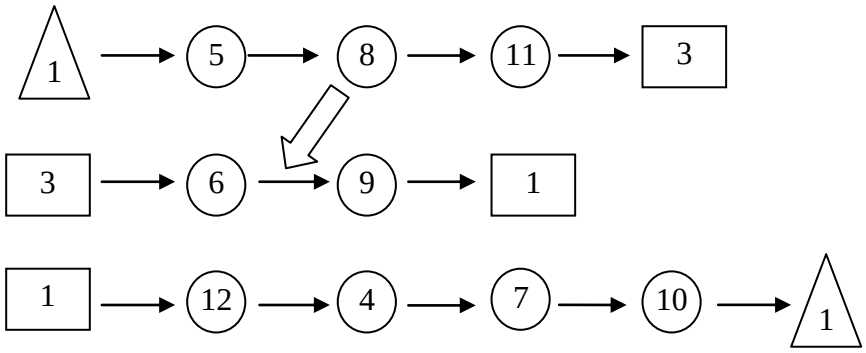


Şekil 3.10. Gezi İçi Ekleme Komşuluk Hareketi Öncesi

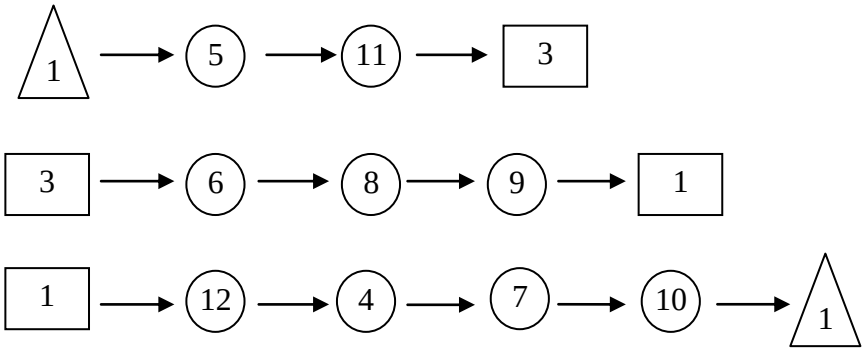


Şekil 3.11. Gezi İçi Ekleme Komşuluk Hareketi Sonrası

Eğer yapılan karşılaştırmalar sonucunda uygulanan ekleme komşuluk mekanizması ile en iyi çözümü veren ekleme işleminin geziler arası yapılan, şekil 3.12.'de gösterildiği gibi birinci gezideki 8 numaralı müşterinin ikinci gezideki 6 ve 9 numaralı müşterilerinin arasına eklenmesi olduğunu varsayarsak, bu ekleme hareketi yapılır ve mevcut çözüm güncellenerek şekil 3.13.'deki halini alır. Artık gezgin birinci gezisinde 8 numaralı müşteriye uğramamakta, bu müşteriye ikinci gezisinde 6. müşteriden sonra uğramaktadır.



Şekil 3.12. Geziler Arası Ekleme Komşuluk Hareketi Öncesi



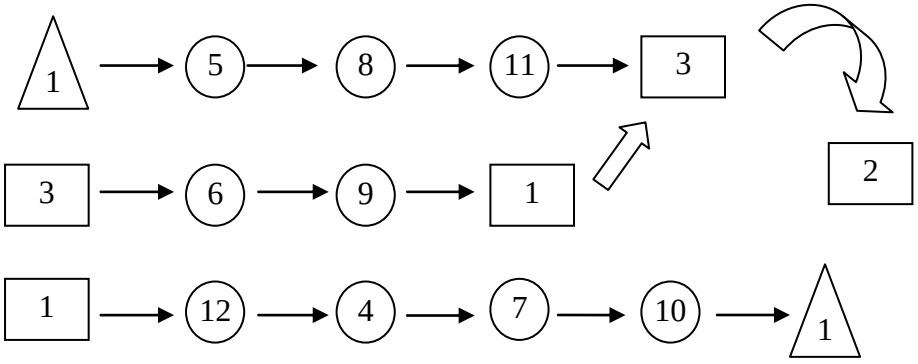
Şekil 3.13. Geziler Arası Ekleme Komşuluk Hareketi Sonrası

3.3.2. Otel iyileştirilmesi

Yapılan tez çalışmasında gezginin gideceği müşteriler en iyi seviyede belirlendikten sonra otellerinde iyileştirilmesi gerekliliği ortaya çıkmıştır. Bu uygulama da "otel değiştir" mekanizması kullanılmıştır. Bu amaçla yapılan çalışmada önceden belirlenen, gezginin konaklayabileceği oteller listesinden faydalanılmıştır. Mevcut çözümde var olan oteller, listede var olan her bir otelle değiştirilerek çözümde bir iyileşme olup olmadığına bakılır. Eğer bir iyileşme olur ise otel değişimi yapılır ve mevcut çözüm güncellenir. Her otel birden fazla kere tur içerisinde kullanılabileceği için bu mekanizma uygulanırken oteller listesinde herhangi bir güncelleme yapılmamıştır. Bu sayede oteller listesindeki otellerde en verimli şekilde değerlendirilmiş olur. Bu hareket sırasında bağlantılı gezilerin ilk ve son otellerinin aynı olmasına dikkat edilmelidir.

Bu iyileştirme hareketi algoritma içerisinde uygulanırken tur içerisinde uğranan tüm oteller yerine, mevcutta bulunan oteller konulur. Bu işlem yapılırken şekil 3.14.'te de gösterildiği gibi 3 numaralı otel yerine daha öncesinde tur içerisinde kullanılmış 1 numaralı

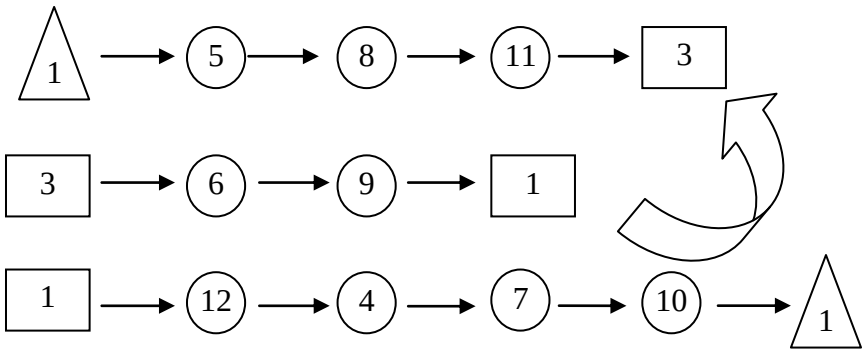
otelin konulabileceği gibi, daha önce tur içerisinde kullanlmamış olan 2 numaralı otel de konulabilir. Yapılan bu değişim işleminden sonra elde edilen çözümlerin uygunluk kontrolleri yapılır ve uygun çözümler içerisinden en iyi çözüme sahip olanı seçilir. Eğer seçilen bu çözüm mevcut çözümden daha iyi amaç fonksiyonu değerine sahip ise mevcut çözüm güncellenir.



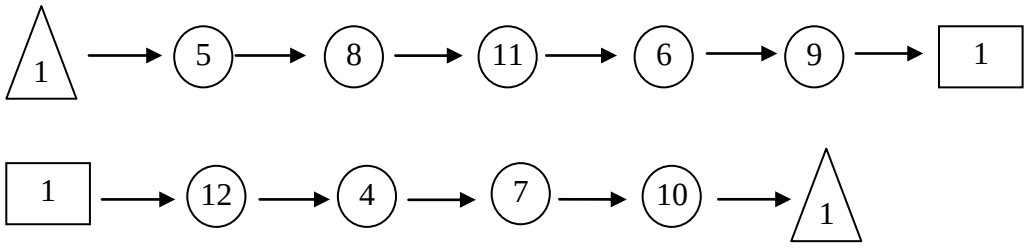
Şekil 3.14. Otel Değiştir Komşuluk Hareketi Örneği

3.3.3. Gezi azalt

Toplam tur mesafesinin azaltılabilmesi için turdaki gezilerin sayısı önem taşımaktadır. Bu nedenle toplam turdaki gezi sayısı ne kadar az olur ise toplam tur uzunluğu da o kadar kısa olur. Gezi azalt mekanizması tur içerisindeki gezi sayısının en iyi seviyede olmasını sağlar. Bunu yaparken oluşturulan çözümdeki her bir gezinin arasındaki oteli kaldırarak bu gezileri birbirinin ardına ekler. Bu ekleme işleminden sonra günlük süre/mesafe kısıtının sağlanıp sağlanmadığı kontrol edilir. Eğer bu kısıt sağlanıyor ise ve toplam tur uzunluğu bir önceki çözüme göre daha avantajlı bir hale geliyor ise gezi azaltma işlemi yapılır. Böylelikle OSGSP'nin diğer bir amacıda sağlanmış olur.



Şekil 3.15. Gezi Azalt Hareketi Öncesi Örneği



Şekil 3.16. Gezi Azalt Hareketi Sonrası Örneği

Şekil 3.15.' de küçük boyutlu bir tur örneği gösterilmiştir. Bu tur içerisinde, yukarıda bahsedildiği gibi her bir gezi birbiri ardına eklenerek günlük süre/mesafe kısıtının aşılıp aşılmadığı kontrol edilmiştir. Yapılan kontroller sonucunda, birinci ve ikinci gezi birleştirilerek uygunluk koşulu bozulmadan turdaki gezi sayısının üçten ikiye düşürülebileceğini varsayarsak, iki gezi birleştirilerek mevcut çözüm güncellenir. Bunun için birinci gezi sonundaki otel kaldırılmış, ikinci gezideki müşteriler birinci gezideki müşterilerin ardına eklenmiş ve şekil 3.16. elde edilmiştir. Böylelikle turdaki gezi sayısı azaltılmıştır.

3.3.4.Değiş tokuş hareketi

Değiş tokuş hareketi ile eldeki mevcut çözümden farklı kombinasyonlarda çözümler elde edilerek daha iyi sonuçlara ulaşmak hedeflenmektedir. Bu harekette öncelikli olarak mevcut çözümdeki turda en az iki noktası bulunan herhangi iki gezi seçilir. Bu geziler şekil 3.17.'deki gibi ikişer parçaya bölünür. Bölme işlemi yapılırken ilk olarak kısım 1'e bir müşteri ve kısım 2'ye gezi içerisindeki diğer müşteriler ve oteller yerleştirilir. Gezi 1 bu şekilde bölünmüş iken gezi 2 içerisindeki bölme işlemi için önce kısım 3'e bir müşteri yerleştirilir, diğer müşteriler ve oteller ise kısım 4'e yerleştirilir. Hareketin diğer adımları tamamlandıktan sonra kısım 3 içerisindeki müşteri sayısı her defasında birer artırılır. Bu durumda kısım 4 içerisindeki müşteri sayıları her defasında bir azaltılmış olur. Bu işlem kısım 4 içerisinde bir müşteri kalana kadar devam ettirilir. Diğer adımlar tamamlandıktan sonra tekrar hareketin bu adıma dönülür ve gezi 1'deki kısım 1 içerisindeki müşteri sayısı bir artırılarak ve kısım 2 içerisindeki müşteri sayısı bir azaltılarak, gezi 2 içerisindeki aynı bölme işlemleri tekrarlanır. Gezi 2 içerisindeki tüm yapılabilecek bölme işlemleri tamamlanınca gezi 1 içerisindeki 1. kısımdaki, müşteri sayısı bir artırılır, 2. kısımdaki müşteri sayısı ise bir azaltılır. Bütün bu bölme işlemleri gezi 1'deki kısım 2 içerisinde bir müşteri kalana kadar devam ettirilir.

Gezi 1	Kısım 1	Kısım 2
Gezi 2	Kısım 3	Kısım 4

Şekil 3.17. Değiş Tokuş Hareketi İle Gezilerin Parçalara Bölünmüş Gösterimi

Her bir bölme işlemi yapıldıktan sonra kısımlar kendi aralarında değiş tokuş yapılır. Bu hareket uygulanırken tek bir gezinin içerisinde yapılan değiş tokuş hareketlerindense, her iki gezidede eş zamanlı olarak yapılan değiş tokuş hareketleri esas alınmıştır. Bu aşamada kısımlar değiş tokuş yapılırken şekil 3.18'de görüldüğü gibi dokuz farklı tipte değiş tokuş kombinasyonu dikkate alınmıştır. Her bir kombinasyon sonucunda uygun veya uygun olmayan çözümler bulunmuştur.

1. Tip

Kısım 2	Kısım 1
Kısım 4	Kısım 3

4. Tip

Kısım 3	Kısım 1
Kısım 2	Kısım 4

7. Tip

Kısım 1	Kısım 4
Kısım 2	Kısım 3

2. Tip

Kısım 1	Kısım 3
Kısım 2	Kısım 4

5. Tip

Kısım 3	Kısım 1
Kısım 4	Kısım 2

8. Tip

Kısım4	Kısım 1
Kısım 3	Kısım 2

3. Tip

Kısım 1	Kısım 3
Kısım 4	Kısım 2

6. Tip

Kısım 1	Kısım 4
Kısım 3	Kısım 2

9. Tip

Kısım 4	Kısım 1
Kısım 2	Kısım 3

Şekil 3.18. Değiş Tokuş Hareketi 9 Farklı Değiş Tokuş Tipi

Değiş tokuş işleminden sonra her bir kısımın kendi içerisinde tersi alınır. Tersinin alınması işlemi, her bir kısımın içerisindeki otel ve müşterilerin tersten sıralanması şeklinde yapılır. Bu işlem yapılırken öncelikle tek bir kısımın tersi alınarak işlem yapılır.

Yani ilk olarak sadece kısım 1'in tersi alınır diğer kısımlar tersi alınmadan gezinin içerisine yerleştirilir. Sonrasında sadece ikinci kısımın tersi alınır, sonrasında üçüncü, sonrasında ise dördüncü kısımın tersi alınır. Tüm kısımların tekli ters alma işlemi tamamlandıktan sonra her bir kısımın ikili kombinasyonlarla tersi alınarak geziler oluşturulur. Sonrasında ise tersinin alınması işlemine üçlü ve dörtlü kombinasyonlarla devam edilir.

Tur içerisinde seçilen iki gezi içinde tüm kısımların ayrı ayrı ve birlikte terslerinin alınmasıyla bir çok farklı çözüm elde edilmiş olur. Bu çözümlerin uygun birer çözüm olup olmadığını tespit etmek için her bir çözüm için uygunluk kontrolü yapılır. Bu kontrol yapılırken bulunan çözümler içerisindeki gezilerin günlük süre/mesafe kısıtını aşıp aşmadığı kontrol edilir. Eğer bu kısıt aşılmıyor yani uygun çözümler ise amaç fonksiyonu değerleri hesaplanır ve içlerinde en iyi değere sahip olan çözüm DKİ sezgiselinde mevcut çözüm ile karşılaştırılmak için hafızada tutulur.

Eldeki çözümdeki tur içerisinde tüm gezilerin farklı ikili kombinasyonları için yukarıdaki adımlar tekrarlanır. Her biri sonucunda elde edilen sonuçlar karşılaştırılır. Değiş tokuş hareketi sayesinde birçok farklı kombinasyon ve dizilim incelenir ve bulunan çözümler ile en iyi sonuca daha da yaklaşılır.

3.4. OSGSP İçin KÇTDKİ Sezgiseli

OSGSP'nde daha etkin sonuçlar alabilmek için önceki bölümlerde açıklanan yöntemler, algoritmalar ve komşuluk mekanizmaları DKİ sezgiseli işleyişine uygun olarak birleştirilmiştir. Algoritmada KÇY her bir yöntemin içerisine yerleştirilmiştir. Yani otel değiştir mekanizması hariç diğer tüm yöntemlerde yeni bir çözüm bulunduğunda o çözümün karşıt çözümü de oluşturularak iki çözüm karşılaştırılmış ve hangi çözüm daha iyi ise o çözüm esas alınmıştır. Böylelikle çeşitlilik artırılmış, bulunan iyi çözümlerden daha farklı alanlarda çözüm aramaları yapma imkanı sağlanmıştır. Sonrasında ise ele alınan esas çözüm mevcut en iyi çözüm ile karşılaştırılmış ve hangi çözümün daha iyi olduğuna karar verilmiştir. Çalışmanın çözüm uzayı sonsuz büyüklükte olmadığından dolayı sadece Tip-I KÇY yöntemi kullanılmıştır. Eğer problem yapısı izin verse idi KÇY yönteminin diğer tipleri de algoritmaya eklenebilirdi.

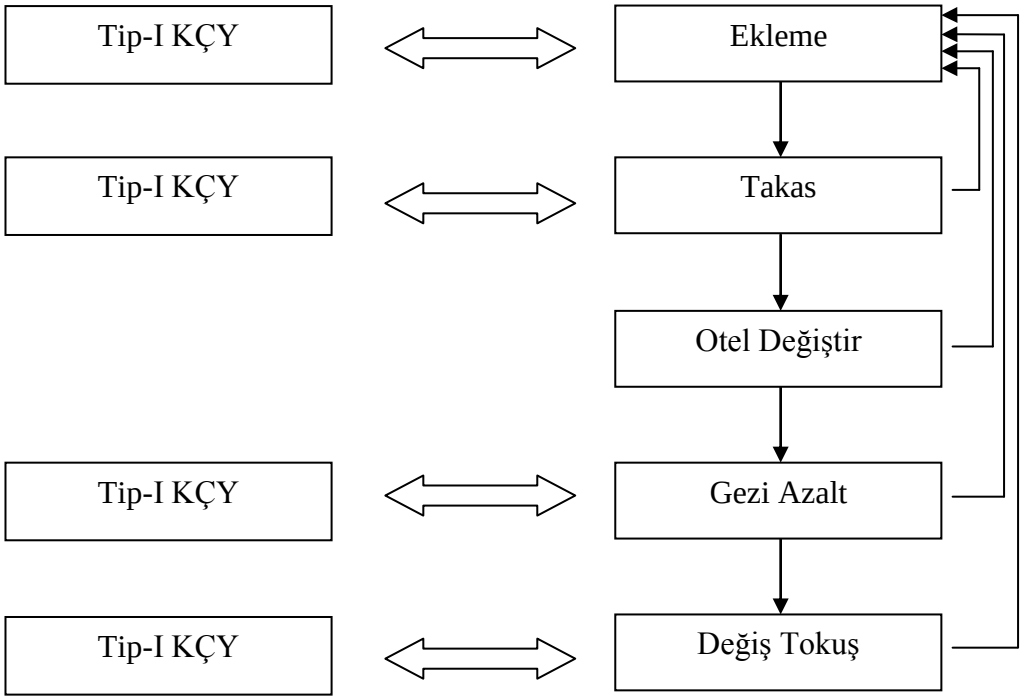
Bu tez için KÇTDKİ uygulama aşamaları Şekil 3.19.'da gösterilmiştir. EYKP ile elde edilen başlangıç çözüme ilk olarak Ekleme komşuluk mekanizması uygulanır. Bu komşuluk mekanizması uygulanırken her bir müşteri turda bütün aralıklara eklenir ve oluşan yeni çözümlerin uygun çözüm olup olmadığına bakılır. Eğer çözümler uygun ise

Tip-I KÇY ile karşıt çözümleri bulunur. Yine bu çözümlerin uygun olup olmadığı incelenir. Tip-I KÇY uygulanmadan önceki ve uygulandıktan sonraki tüm uygun çözümlerin amaç fonksiyonu değerleri karşılaştırılır ve aralarından en iyi sonuca sahip çözüm seçilir. Bu seçilen çözüm EYKP ile bulunan başlangıç çözüm ile karşılaştırılır. Eğer başlangıç çözümden daha iyi bir çözüm bulunmuş ise mevcut çözüm güncellenir. Algoritma içerisinde ekleme komşuluk mekanizması ile daha iyi bir sonuç bulunduğu için bu adım devam ettirilir ve en iyi çözüme yukarıda anlatılan adımlar uygulanır eğer daha iyi bir çözüm bulunur ise mevcut çözüm güncellenir. Daha iyi bir çözüm bulunamaz ise algoritmanın bir sonraki adımı olan takas komşuluk mekanizmasına geçilir.

Takas komşuluk mekanizması uygulanırken ekleme komşuluk mekanizmasında olduğu gibi her bir müşteri için ayrı ayrı tur içerisinde tüm takas işlemleri yapılır, Bulunan çözümlerin uygunluğu kontrol edilir. Eğer çözümler uygun bulunur ise, bu çözümlerin karşıt çözümleri bulunur. Bulunan karşıt çözümler içerisinden de uygun çözümler seçilerek eldeki tüm uygun çözümler birbiri ile karşılaştırılır. İçlerinden en iyi olan çözüm mevcut çözüm ile karşılaştırılır. Karşılaştırma sonucu takas komşuluk mekanizması ile bulunan sonuç daha iyi çıkar ise mevcut çözüm güncellenir ve algoritmanın ilk adımına yani ekleme komşuluk mekanizması uygulama adımına dönülür. Mevcut çözüme algoritmanın adımları uygulanır. Eğer takas komşuluk mekanizması ile daha iyi bir çözüm bunamaz ise, algoritmanın üçüncü adımına geçilir.

Üçüncü adımda mevcut çözüme otel değiştir yöntemi uygulanır. Eğer bu yöntem ile daha iyi bir çözüm bulunur ise mevcut çözüm güncellenir ve algoritmanın ilk adımı olan ekleme adımına dönülür. Yukarıda bahsedildiği gibi algoritmanın adımları uygulanır. Otel değiştir yöntemi ile daha iyi bir sonucun bulunamadığı noktada gezi azalt adımına geçilir.

Gezi azalt adımında tur içerisindeki gezi sayılarını azaltmak için geziler bölüm 3.3.3.'te bahsedildiği gibi birbiri ardına eklenir. Bulunan her bir yeni çözümün uygunluğu kontrol edilir, uygun olan çözümlerin karşıt çözümleri bulunarak bu çözümlerinde uygun olup olmadığına bakılır. Bu aşamada bulunan tüm uygun çözümler kendi içlerinde karşılaştırılır ve içlerinden en iyi olan çözüm mevcut çözüm ile karşılaştırılır. Eğer daha iyi bir çözüm bulunmuş ise KÇTDKİ algoritmasının ilk adımına dönülür, bulunamaz ise bir sonraki adım olan değiş tokuş adımına geçilir.



Şekil 3.19. Karşıt Çözüm Temelli Değişken Komşu İniş Sezgiseli Örneği

Değiş tokuş adımı aslında en kapsamlı aramanın yapıldığı adımdır. Öncelikle mevcut çözümdeki tur içerisinde her bir gezi ikişerli gruplar halinde seçilir, geziler tüm olası durumlara göre ayrı ayrı kendi içlerinde ikişer parçaya ayrılır, gezilerin ayrılan parçaları dokuz farklı kombinasyon ile kendi içlerinde değiş tokuş yapılır. Sonrasında ise oluşturulan her bir kombinasyonun kendi arasında birerli, ikişerli, üçerli ve dörderli olarak tersi alınır ve bulunan yeni çözümlerin uygunluğu kontrol edilir. Uygun bulunan çözümlerin Tip-I KÇY ile karşıt çözümleri bulunur. Bulunan karşıt çözümlerinde uygunluğu kontrol edilir. Algoritmanın beşinci adımı olan değiş tokuş adımında da bulunan bütün uygun çözümler karşılaştırılır ve içlerinden en iyi olan çözüm seçilir. Bu çözüm mevcut çözüm ile karşılaştırılır eğer daha iyi bir çözüm bulunmuş ise mevcut çözüm güncellenir ve algoritmanın ilk adımına dönülür. Daha iyi bir çözüm bulunamamış ise algoritma sonlandırılır, bulunan çözüm en iyi çözüm olarak kaydedilir.

Değiş tokuş hareketinin, sezgisel algoritma içerisinde son sırada kullanılmasının sebebi; diğer komşuluk hareketleri ile daha iyi bir çözüm bulunamaması durumunda, çözüm uzayında çok kapsamlı bir arama yaparak çözümü daha da iyileştirmesidir. Çözüm uzayında yapılan bu kapsamlı arama çok zaman alabilmektedir. Fakat diğer komşuluk mekanizmaları ile daha iyi bir çözüm bulunamadığı durumda bu hareket ile daha iyi bir

çözüm bulunması olasılığı daha yüksektir. Bu nedenle bu hareketin kullanılması çözüm kalitesi açısından avantaj sağlamıştır.

4. SAYISAL ANALİZLER

Sayısal analizler yapılırken OSGSP için eniyi çözümü bulabilmek için veri setlerine KÇTDKİ algoritması uygulanmıştır. Elde edilen sonuçlar listelenmiştir ve tasarlanan algoritmanın performansı değerlendirilmiştir. Ayrıca bu bölümde çözüm alınırken kullanılan bilgisayar özelliklerinden ve programlama dilinden de bahsedilmiştir.

4.1. Test Problemleri

Analizler yapılırken daha önce Vansteenwegen ve ark. tarafından tasarlanan test verileri kullanılmıştır [1]. Önerilen sezgisel yöntemin performansını incelemek için daha küçük boyutlu olan Set-1 ve Set-2 ile büyük boyutlu Set-3 ve Set-4 olmak üzere dört test örneği de kullanılmıştır [54]. Kullanılan test verilerinin boyutları tablo 4.1.de özetlenmiştir.

Tablo 4.1. Test Verileri

Set	Müşteri	Otel
SET 1	48-288	6
SET 2	10 15 30 40	2
SET 3	52-1002	3 5 10
SET 4	52-1002	10

Set-1 örnek kümesi, Solomon'un altı adet zaman pencereli ARP'si (c101, c201, r101, r201, rc101 ve rc201) [55] ve Cordeau ve ark. 10 adet çoklu depolu ve zaman pencereli ARP'si (pr1-pr10) [56] olmak üzere toplamda 16 adet problemden oluşmaktadır. Solomon'un çalışmasından alınan veriler 100 müşteri, Cordeau ve ark.'larının

çalışmasından alınan veriler ise 48 ile 288 arasında müşteri içermektedir. Alınan veri setleri OSGSP'ne uyarlanırken zaman pencereleri hesaba katılmamış ve otellerin başlangıç ve bitiş zamanları arasındaki süre zaman kısıtı olarak işleme alınmıştır. OSGSP oluşturabilmek içinde beş müsait otel, müşteri lokasyonlarına eklenmiştir.

Set-2 örnek kümesi, 52 farklı problemden oluşmaktadır. Bu veri seti oluşturulurken Set-1 müşteri setinden küçük bir küme seçilmiştir. Set-2 verileri referans alınan problemin yanına müşteri sayıları eklenerek isimlendirilmiştir. Her problem için 10, 15, 30 ve 40 müşteri olacak şekilde düzenlenmiştir.

Set-3 örnek kümesi, 51-1002 müşteri ile 4, 6 veya 11 oteli içeren daha karmaşık örnekler içermektedir. Bu örnekler bilinen GSP örneklerinden türetilmiştir ve her biri OSGSP'nin optimal çözümü bilinecek şekilde tasarlanmıştır. Bu örnekler türetilirken GSP içerisindeki bazı müşteri noktaları aynı zamanda otel noktaları olarak seçilmiş ve işlemler buna göre yapılmıştır.

Set-4 örnek kümesinde ise, Set-3'te kullanılan aynı GSP örneği kullanılmıştır. Farklı olarak ise mevcut müşterilere 10 adet otel eklenmiştir. Bu oluşturulan örneklerin en iyi çözümleri bilinmemektedir.

4.2. Bilgisayar Özellikleri ve Programlama Dili

Algoritma çözümlenirken i7 işlemci 3.5 GHZ, 8GB ram ve Linux işletim sistemine sahip bir bilgisayar kullanılmıştır. Algoritma kodlanırken dizi işlemlerini yapma kolaylığı sağladığı için kodlama dili olarak Python dili kullanılmıştır. Python kodu Ek 1'de yer almaktadır. Python programlama dili veri bilimi, makine öğrenimi, sistem otomasyonu, web ve API geliştirme ve daha fazlası için bir temel yapıdır. 1991'den beri Python programlama dili sadece gereksiz programlar için tamamlayıcı bir dil olarak değerlendirilmekteydi. Fakat son birkaç yılda Python modern yazılım geliştirme, altyapı yönetimi ve veri analizinde birinci sınıf bir programlama dili olarak ön plana çıkmıştır. Python dili önemli avantajlara sahiptir. Bunlardan ilki, dili kullanmanın ve öğrenmenin oldukça kolay olmasıdır. Bu nedenle algoritmayı kodlarken Python kullanmak kodlama zamanı açısından avantaj sağlamıştır. İkinci avantajı ise, kullanım alanın oldukça geniş olmasıdır. Buda Python dilinin birçok alanda kullanılabilir olmasını sağlamaktadır. Python dilinin bu avantajlarının yanısıra en büyük dezavantajı ise script dili olduğundan dolayı çalışma zamanı diğer üst seviye dillere göre (C, Java vb.) daha fazladır. Ancak Python dili ile paralel proses işlemleri kolaylıkla yapılabilmektedir. Bu sayede eğer ihtiyaç duyulur ise

çalışma zamanını kısaltmak için işlemler işlemcilere dağıtılabilir. Böylelikle daha kısa zamanda sonuçlara ulaşılır.

4.3. Sayısal Sonuçlar

Değişken komşu iniş sezgiseli dört veri seti için uygulanmıştır. Tüm veri setleri elektronik ortamdaki kaynaktan [54] indirilerek Ek 1'de verilmiş olan Python koduna uygun hale getirilmiştir. İndirilen her veri seti içerisindeki problemlerde kullanılabilecek otel sayısı (s) ve koordinatları (XPos, YPos) ile gidilecek müşterilerin sayısı (n) ve koordinatları (XPos, YPos) mevcuttur. Ayrıca dosya içerisinde gezi sayısı (m), günlük süre/mesafe sınırlaması (C) ve her bir müşteride gezginin harcadığı hizmet süresi matrisi (T) yer almaktadır. Eğer gezginin müşteride harcadığı zaman, problem çözümü için önemsiz ise T matrisi sıfır değerlerini almaktadır. Bu veri setlerinin Python koduna uygun hale getirilmiş bir örneğine Ek 2 de yer verilmiştir. Bu örnek 3. veri setindeki a_280.s3 problemine aittir. Ek 2 de verilen örnek gösterimde otel ve müşteri koordinatları aynı satırda yer almaktadır. Problem tanımında olduğu gibi koordinat matrisinde baştan otel sayısı kadarki koordinatlar otel koordinatları, geriye kalanlar ise müşterilerin konumlarını belirleyen koordinatlarıdır. Tüm veri setleri içerisindeki problemlere ait veriler Ek 2 de gösterilen biçime getirilmiş ve Python kodu ile sonuçlar elde edilmiştir. Elde edilen karşılaştırmalı sonuçlar tablo 4.2., tablo 4.3., tablo 4.4., tablo 4.5., tablo 4.6., tablo 4.7. ve tablo 4.8. de gösterilmiştir.

Tablo 4.2. de ilk veri setinden elde edilen sonuçlar listelenmiştir. Bu veri seti büyük ölçekli problemler için tasarlanmıştır. Set 1 içerisinde coğrafi koordinat verilerinden oluşan 16 farklı problem vardır. Her bir problemin ismi, müşteri sayısı, zaman bütçesi, seyahat sayısı ve çözüm zamanları ilk beş sütunda listelenmiştir. Sonraki sütunda Wansteenwegen ve ark. tarafından 2011 yılında oluşturulan I2LS sezgiseli ile elde edilen sonuçlar yer almaktadır [1]. KÇTDKİ çözümü ise bu çalışmada tasarlanan algoritma ile elde edilen çözümü ifade etmektedir. Son sütunda I2LS çözümden elde edilen sapma miktarı listelenmiştir. Bu sapma miktarı formülü (4.1) ile yüzde cinsinden gösterilmiştir. Bu formül kullanılarak her bir problem için sapma miktarları hesaplanmıştır.

$$[(\text{KÇTDKİ çözüm} - \text{I2LS Çözüm}) / \text{I2LS çözüm}] * 100 \quad (4.1)$$

Tablo 4.2. Set 1 İçin Karşılaştırmalı Sonuçlar

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
c101	100	1236	9	1126.81	9699.1	9750.7	0.53
c201	100	3390	3	476.48	9600.0	9606.2	0.06
pr01	48	1000	2	112.42	1446.0	1478.1	2.22
pr02	96	1000	3	754.46	2569.3	2824.6	9.94
pr03	144	1000	4	2207.64	3632.5	3823.2	5.25
pr04	192	1000	5	4196.06	4366.3	4575.9	4.80
pr05	240	1000	6	17278.58	5122.1	5451.8	6.44
pr06	288	1000	7	22695.36	6137.3	6421.3	4.63
pr07	72	1000	3	250.18	2090.9	2309.6	10.46
pr08	144	1000	4	2000.72	3504.7	3693.6	5.39
pr09	216	1000	5	9075.16	4617.6	4798.7	3.92
pr10	288	1000	7	24935.58	6097.5	6605.4	8.33
r101	100	230	9	749.39	1801.3	**1763.5**	-2.10
r201	100	1000	2	448.12	1678.0	**1669.8**	-0.49
rc101	100	240	8	628.72	1724.1	1741.7	1.02
rc201	100	960	2	424.51	1670.0	1675.8	0.35
Ort.				5460,01			3.80

Tablo 4.2. de I2LS sonuçları ile KÇTDKİ sonuçları karşılaştırıldığında r101 ve r201 problemlerinde sırasıyla 2.10 ve 0.49'luk sapmalar ile daha iyi sonuçlar alındığı gözlemlenmiştir. Daha iyi sonuç alınan bu problem çözümleri çizelgede koyu renk ile gösterilmiştir. Set 1 verileri için KÇTDKİ sezgiseli ile bulunan sonuçların, I2LS ile bulunan sonuçlardan sapmalarının ortalaması ise %3.80 olarak hesaplanmıştır. Set 1 içerisinde yer alan farklı büyüklüklerdeki problemlerin çözüm zamanı ortalamaları ise 5460.01 saniye olarak hesaplanmıştır.

Tablo 4.3. de iki sezgisel karşılaştırıldığında KÇTDKİ sezgiselinin, I2LS sezgisel sonuçlarıyla aynı veya daha iyi olan sonuçlar bulduğu görülmüştür. Bu veri setindeki problemlerden rc101, pr02, pr03, pr10 olmak üzere 4 tanesinde I2LS sezgiseli ile aynı sonuçlar bulunurken, r101, pr04, pr05, pr07, pr09 olmak üzere 5 problem için I2LS sezgiselinden daha iyi sonuçlar bulunmuştur. Diğer problemlerde ise KÇTDKİ sezgiseli ile bulunan sonuçların diğer sezgisel sonuçlarından sapmalarının ortalaması %-0.98 bulunmuştur. Yani bu çalışmada tasarlanmış olan KÇTDKİ sezgiseli ile 10 müşterili set 2 verilerinde, I2LS sezgiseli ile bulunan sonuçlardan ortalama olarak daha iyi sonuçlar elde

edilmiştir. 10 müşteriye sahip set 2 problemlerinin çözüm zamanı ortalamaları 0.04 saniye olarak hesaplanmıştır.

Tablo 4.3. Set 2 İçin Karşılaştırmalı Sonuçlar (10 müşteri)

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
c101	10	1236	1	0.03	955.4	959.5	0.43
r101	10	230	2	0.14	286.2	**272.8**	-4.68
rc101	10	240	1	0.16	237.5	**237.5**	0.00
pr01	10	1000	1	0.03	426.6	434.3	1.80
pr02	10	1000	1	0.03	661.9	**661.9**	0.00
pr03	10	1000	1	0.01	559.1	**559.1**	0.00
pr04	10	1000	1	0.02	511.1	**476.4**	-6.79
pr05	10	1000	1	0.02	560.9	**528.9**	-5.71
pr06	10	1000	1	0.01	604.1	615.9	1.95
pr07	10	1000	1	0.01	708.1	**702.6**	-0.78
pr08	10	1000	1	0.02	573.4	580.9	1.31
pr09	10	1000	1	0.01	647.5	**645.5**	-0.31
pr10	10	1000	1	0.01	461.5	**461.5**	0.00
Ort.				0,04			-0.98

Tablo 4.4. de KÇTDKİ sezgiseli ile bulunan sonuçlar pr06, pr10 olmak üzere 2 problem için I2LS sezgiseli ile bulunan sonuçlarla aynı iken, c101, r101, pr07, pr09 olmak üzere 4 problem için KÇTDKİ sezgiseli ile daha iyi sonuçlar bulunmuştur ve bu problemler için sapma değerleri negatif olarak yazılmıştır. 15 müşterili set 2 verileri için sonuçlar karşılaştırıldığında sapma değerlerinin ortalaması %2.02 olarak hesaplanmıştır. 15 müşteriye sahip set 2 problemlerinin çözüm zamanı ortalamaları 0.24 saniye olarak hesaplanmıştır.

Tablo 4.5. de sapma değerlerine bakıldığında KÇTDKİ sezgiseli ile bulunan sonuçların c101, rc101, pr02, pr03, pr04, pr08 olmak üzere 6 problem için I2LS sezgiseli ile bulunan sonuçlardan daha iyi olduğu görülmüştür. 30 müşterili set 2 verileri için sonuçlar karşılaştırıldığında sapma değerlerinin ortalaması %1.10 olarak hesaplanmıştır. 30 müşteriye sahip set 2 problemlerinin çözüm zamanı ortalamaları 5.67 saniye olarak hesaplanmıştır.

Tablo 4.4. Set 2 İçin Karşılaştırmalı Sonuçlar (15 müşteri)

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
c101	15	1236	2	0.55	1456.7	**1452.5**	-0.29
r101	15	230	2	0.78	391.5	**379.8**	-2.99
rc101	15	240	2	0.86	306.1	307.2	0.36
pr01	15	1000	1	0.14	590.4	635.6	7.66
pr02	15	1000	1	0.09	751.1	790.0	5.18
pr03	15	1000	1	0.10	649.1	698.9	7.67
pr04	15	1000	1	0.12	683.4	733.0	7.26
pr05	15	1000	1	0.10	660.4	674.1	2.07
pr06	15	1000	1	0.06	685.2	**685.2**	0.00
pr07	15	1000	1	0.13	812.5	**802.9**	-1.18
pr08	15	1000	1	0.09	707.2	712.7	0.78
pr09	15	1000	1	0.05	773.6	**771.7**	-0.25
pr10	15	1000	1	0.10	611.9	**611.9**	0.00
Ort.				0,24			2.02

Tablo 4.5. Set 2 İçin Karşılaştırmalı Sonuçlar (30 müşteri)

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
c101	30	1236	3	5.77	2907.8	**2863.3**	-1.53
r101	30	230	3	11.77	676.2	692.8	2.45
rc101	30	240	4	12.32	747.0	**684.4**	-8.38
pr01	30	1000	1	2.50	964.8	1054.5	9.30
pr02	30	1000	2	5.07	1140.6	**1126.0**	-1.28
pr03	30	1000	1	6.93	957.1	**952.5**	-0.48
pr04	30	1000	2	8.54	1149.3	**1130.9**	-1.60
pr05	30	1000	1	3.60	936.3	951.3	1.60
pr06	30	1000	2	3.76	1114.4	1186.1	6.43
pr07	30	1000	2	4.23	1158.0	1169.7	1.01
pr08	30	1000	2	0.70	1056.1	**1025.7**	-2.88
pr09	30	1000	2	6.38	1133.1	1171.1	3.35
pr10	30	1000	1	2.16	927.1	985.6	6.31
Ort.				5,67			1.10

Tablo 4.6. incelendiğinde KÇTDKİ sezgiseli ile bulunan sonuçların 3 problem için I2LS sezgiseli ile bulunan sonuçlardan daha iyi olduğu görülmüştür. 40 müşterili set 2 verileri için sonuçlar karşılaştırıldığında sapma değerlerinin ortalaması %2.28 olarak hesaplanmıştır. 40 müşteriye sahip set 2 problemlerinin çözüm zamanı ortalamaları 18.50 saniye olarak hesaplanmıştır.

Tablo 4.6. Set 2 İçin Karşılaştırmalı Sonuçlar (40 müşteri)

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
c101	40	1236	4	22.76	3950.0	**3933.1**	-0.43
r101	40	230	4	18.89	895.5	907.2	1.31
rc101	40	240	4	18.73	851.2	870.8	2.30
pr01	40	1000	2	22.21	1223.6	1244.0	1.67
pr02	40	1000	2	27.39	1418.2	**1376.2**	-2.96
pr03	40	1000	2	16.49	1386.9	1471.1	6.07
pr04	40	1000	2	20.72	1292.9	1298.6	0.44
pr05	40	1000	2	18.77	1200.8	1245.5	3.72
pr06	40	1000	2	16.25	1279.2	1318.1	3.04
pr07	40	1000	2	20.90	1426.5	1480.2	3.76
pr08	40	1000	2	15.02	1305.9	**1297.5**	-0.64
pr09	40	1000	2	10.27	1287.0	1323.4	2.83
pr10	40	1000	2	12.10	1233.6	1338.7	8.52
Ort.				18.50			2.28

Genel olarak set 2 verileri için karşılaştırmalı sonuçların yer aldığı tablolar incelendiğinde 10 müşterili problemlerde KÇTDKİ sezgiseli ile daha iyi sonuçlar bulunurken, müşteri sayısı arttıkça I2LS sezgiselinden sapmalarda artmıştır. 10, 15, 30 ve 40 müşterili problemlerde sırasıyla sapma değerleri yüzde cinsinden; -0.98, 2.02, 1.10 ve 2.28 olarak hesaplanmıştır. Bunun nedeni müşteri sayısı arttıkça problemlerin daha karmaşık hale gelmesi ve daha iyi sonuçların bulunmasının zorlaşmasıdır. Veri seti 2 de tüm problemler için ortalama sapma ise %1.11 olarak hesaplanmıştır. Bu veri seti için çözüm zamanları 10, 15, 30 ve 40 müşterili problemler için sırasıyla ortalama 0.04, 0.24, 5.67, 18.50 saniye olarak hesaplanmıştır. Bu sonuçlarda da görüldüğü gibi müşteri sayıları arttıkça problemlerin çözüm zamanlarıda artmaktadır. Set 2 verileri için ortalama çözüm zamanı ise 6.11 saniye olarak bulunmuştur.

Veri seti 3 içerisindeki problemler tasarlanırken GSP'leri OSGSP'ne dönüştürülmüştür. Bu dönüşüm yapılırken problemlerin içerisinde müşteri olarak bulunan bazı noktalar aynı zamanda otel olarak kabul edilmiş ve veri seti bu şekilde oluşturulmuştur. Her problem için fazladan 3, 5 ve 10 müşteri seçilmiş ve bu GSP'leri 4, 6 ve 11 otelli OSGSP haline getirilmştir.

Tablo 4.7.'de veri seti 3 için KÇTDKİ sezgiseli ve I2LS sezgiseli sonuçları karşılaştırılmıştır. Bu karsılaştırma sonucunda çizelgede hesaplanan sapma değerleri incelendiğinde KÇTDKİ sezgiselinde 9 problem için daha iyi sonuçların bulunduğu gözlemlenmiştir. Tabloda da görüldüğü gibi veri set 3 için KÇTDKİ sezgisel sonuçlarının I2LS sezgisel sonuçlarından ortalama sapması %5.50 olarak hesaplanmıştır. Set 3 içerisindeki problemlerin çözüm zamanı ortalamaları 6540.78 saniye olarak hesaplanmıştır. Ortalama zaman hesaplanırken tabloda son sırada yer alan 1002 müşteriye sahip olan problemin çözüm zamanı hesaba katılmamıştır.

Tablo 4.7. Set 3 İçin Karşılaştırmalı Sonuçlar (3, 5 ve 10 ekstra otelli)

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
eil_51.s3	51	114	5	40.0	477	**454.4**	-4.74
eil_51.s5	51	79	>	36.01	469	490.9	4.67
eil_51.s10	51	46	13	34.70	533	**504.3**	-5.38
berlin_52.s3	52	2041	5	43.66	8150	9192.3	12.79
berlin_52.s5	52	1478	6	36.69	8200	9213.4	12.36
berlin_52.s10	52	1148	9	58.70	8328	8945	7.41
st_70.s3	70	181	5	93.36	754	778.7	3.28
st_70.s5	70	123	7	102.68	777	917.2	18.04
eil_76.s3	76	138	5	125.41	629	653.4	3.88
eil_76.s5	76	101	7	160.34	621	**620.5**	-0.08
eil_76.s10	76	55	13	186.13	650	650.4	0.06
pr_76.s3	76	30734	4	117.31	111260	118084.8	6.13
pr_76.s5	76	20331	7	114.12	123266	156117.8	26.65
kroA_100.s3	100	5639	5	337.81	22806	24882.8	9.11
kroA_100.s5	100	3819	7	360.16	22644	25743.9	13.69
kroC_100.s3	100	5269	5	629.25	23532	24456.2	3.93
kroC_100.s5	100	3751	8	278.91	23394	27767.2	18.69
kroD_100.s3	100	5629	5	267.36	24870	27898.3	12.18
kroD_100.s5	100	3814	7	337.70	24287	29358.2	20.88
rd_100.s3	100	2048	5	452.69	8869	9832.6	10.86
rd_100.s5	100	1424	7	401.77	8638	9217.7	6.71
rd_100.s10	100	921	12	367.42	9471	10728.8	13.78

Tablo 4.7. devam ediyor.

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
eil_101.s3	101	164	5	740.73	730	**704**	-3.56
eil_101.s5	101	109	8	579.34	727	784.6	7.92
eil_101.s10	101	65	13	399.22	723	726	0.41
lin_105.s3	105	3661	5	463.59	16878	17304.6	2.53
ch_150.s3	150	1131	5	1980.22	7426	7674.5	3.35
ch_150.s5	150	678	8	1150.56	7444	7841.6	5.34
tsp_225.s3	225	993	5	6062.49	4555	4562.7	0.17
tsp_225.s5	225	671	8	4697.50	4792	**4330**	-9.64
tsp_225.s10	225	396	15	6419.89	4760	**4514.6**	-5.16
a_280.s3	280	665	5	21968.14	3003	3105.2	3.40
a_280.s5	280	436	8	8898.58	3122	3594.6	15.14
a_280.s10	280	248	16	16900.53	3520	**3094.8**	-12.08
pcb_442.s3	442	12819	5	34057.52	56058	**55606.1**	-0.81
pcb_442.s5	442	8678	8	74025.04	58494	**56231.4**	-3.87
pcb_442.s10	442	4830	14	59083.24	58110	61167.9	5.26
pr_1002.s3	1002	65186	5	88139+	291158	308981.9	6.12
Ort.				6540.78			5.50

Set 4 için karşılaştırmalı sonuçlar tablo 4.8. de listelenmiştir. Sonuçlar incelendiğinde KÇTDKİ sezgisel sonuçlarının, I2LS sezgisel sonuçlarından sapmaları %10.29 olarak hesaplanmıştır. Bu sonuçlar otel sayısı ve müşteri sayısı arttıkça daha iyi çözümler bulmanın zorlaştığını göstermektedir. Set 4 içerisindeki problemlerin çözüm zamanı ortalamaları 6251.46 saniye olarak hesaplanmıştır. Ortalama çözüm zamanı hesaplanırken tabloda son sırada yer alan 1002 müşteriye sahip olan problemin çözüm zamanı hesaba katılmamıştır. Tüm veri setleri için çözüm zamanları karşılaştırıldığında otel sayısı ve müşteri sayısı arttıkça çözüm zamanlarınında arttığı gözlemlenmiştir.

Tablo 4.8. Set 4 İçin Karşılaştırmalı Sonuçlar

İsim	Müşteri Sayısı	Zaman Bütçesi	Seyahat Sayısı	Çözüm Zamanı (sn)	I2LS	KÇTDKİ	Sapma(%)
eil_51	51	46	13	35.71	479	486.6	1.59
berlin_52	52	1148	9	41.74	8823	9931.9	12.57
st_70	70	72	14	108.94	745	945.7	26.94
eil_76	76	55	13	123.77	595	636.2	6.92
kroA_100	100	2202	12	317.15	22828	28529.5	24.98
kroC_100	100	2215	13	264.61	23744	26508.8	11.64
kroD_100	100	2184	13	292.37	24904	26575.9	6.71
rd_100	100	921	12	603.57	8769	9462.3	7.91
eil_101	101	65	13	356.28	693	726.8	4.88
ch_150	150	678	14	1118.32	7679	7814.4	1.76
tsp_225	225	396	15	5383.07	4819	5744.1	19.20
a_280	280	248	16	13351.11	3123	3420.4	9.52
pcb_442	442	4830	14	59272.32	63822	65022.7	1.88
pr_1002	1002	23799	14	5561.6+	330282	355336	7.59
Ort.				6251.46			10.29

5. SONUÇ VE ÖNERİLER

Bu tez kapsamında GSP'nin bir varyantı olan OSGSP üzerinde çalışılmıştır. Bu problem birden fazla geziden oluşması, her gezinin bir otelde başlayıp bir otelde bitmesi, her gezi için günlük süre/mesafe kısıtının olması gibi sınırlamalara sahip olduğu için NP-zor problem sınıfına girmektedir. Bu nedenle Cplex çözücüsü ile yapılan çözümler uzun zaman almaktadır. Bu dezavantajı ortadan kaldırmak için KÇTDKİ sezgiseli tasarlanmıştır ve literatürde var olan 4 veri seti içinde algoritma çalıştırılarak çözümler alınmıştır. Alınan sonuçlar ile Cplex çözücüsü sonuçları tablo 5.1.'de karşılaştırılmıştır. Tabloda her veri seti için Cplex çözücüsü ile optimum sonuç bulunan problem sayısı ve eğer makul sürede Cplex çözümü tamamlayamadıysa elde edilen en iyi çözümlerin sayısı listelenmiştir. Set 1 verileri için Vansteenwegen ve ark. tarafından yapılan çalışmada 6 saat içerisinde hiç bir problem için optimum sonuç bulunamamıştır. Bu nedenle 16 problem içinde bilinen en iyi sonuçlar yazılmıştır. KÇTDKİ sezgiselinde ise 2 problem için bilinen en iyi sonuçlardan daha iyi sonuçlar bulunmuştur. Set 2 verilerinde 28 problem için Cplex çözücüsü ile optimum sonuçlar bulunurken, 24 problem de 2 saat içerisinde optimum sonuçlar bulunamadığı için bilinen en iyi değerler alınmıştır. KÇTDKİ sezgisel sonuçları karşılaştırıldığında ise optimal sonuçların 11 problem için yakalandığı, 6 problem için de bilinen en iyi sonuçlardan daha iyi sonuçlar bulunduğu görülmüştür. Set 3 verilerinde ise 38 problem içinde optimum sonuçlar bulunamamış, 9 problem için KÇTDKİ sezgiselinde bilinen en iyi sonuçlardan daha iyi sonuçlar bulunmuştur.

Tablo 5.1. Cplex Çözüm ve KÇTDKİ Sezgiseli Sonuçları Karşılaştırılması

	SET 1		SET 2		SET 3	
	Cplex	KÇTDKİ	Cplex	KÇTDKİ	Cplex	KÇTDKİ
Optimum sonuçlar	-	-	28	11	-	-
Bilinen en iyi sonuçlar	16	2	24	6	38	9

Tablo 5.2. de I2LS ve KÇTDKİ sezgiseli sonuçları karşılaştırılmıştır. Bu tabloya göre KÇTDKİ sezgiselinde 1. veri seti içerisindeki 16 problemden 2'sinde, 2. veri seti içerisindeki 52 problemden 24'ünde ve 3. veri setindeki 38 problemden 9'unda daha iyi sonuçlar bulunmuştur. 4. veri setindeki problemler daha karmaşık problemler olduğu için daha iyi sonuçlar bulunamamıştır.

Tablo 5.2. I2LS Sezgiseli ve KÇTDKİ Sezgiseli Sonuçları Karşılaştırılması

	SET 1	SET 2	SET 3	SET 4
	KÇTDKİ	KÇTDKİ	KÇTDKİ	KÇTDKİ
Sonuçlar	2/16	24/52	9/38	0/14

Tablo 5.3.'te KÇTDKİ sezgiselinin I2LS sezgiselinden minimum, maksimum ve ortalama sapma değerleri listelenmiştir. Set 2 verileri daha küçük problemlerden oluştuğu için ortalama sapma değeri diğer veri setleri için hesaplanan ortalama sapma değerlerinden daha küçük çıkmıştır. Bunun yanı sıra set 2 verilerinde olduğu gibi set 1 ve set 3 verilerinde de minimum sapma değerleri negatif bulunurken daha karmaşık problemleri içeren set 4 verilerinde pozitif değer almıştır. Tablodaki maksimum sapma değerlerine bakıldığında set 3 ve set 4 verileri en büyük değerleri almıştır. Tüm tablo sonuçları beraber değerlendirildiğinde sırasıyla veri seti 2, veri seti 1, veri seti 3 ve veri seti 4 için daha iyi sonuçlar bulunmuştur.

Tablo 5.3. KÇTDKİ Sezgiseli Min., Mak. ve Ort. Sapma Değerleri

	SET 1	SET 2	SET 3	SET 4
Min. Sapma (%)	-2.10	-8.38	-12.08	1.59
Mak. Sapma (%)	10.46	9.30	26.65	26.94
Ort. Sapma (%)	3.80	1.11	5.50	10.29

İleriki çalışmalarda, dört veri seti içinde elde edilen sezgisel algoritma sapma değerlerinin iyileştirilebileceği gibi, OSGSP'nin genelleştirilmiş bir türü olan birden fazla gezgin içeren çoklu OSGSP için çözüm yöntemleri de geliştirilebilir. Ayrıca bu problem için KÇY kullanılırken problem kısıtlarından dolayı sadece Tip-I KÇY uygun bulunmuştur. Eğer problem tipi diğer KÇY tiplerinin de kullanılmasına imkan sağlar ise, algoritma içerisine diğer KÇY tipleri de eklenerek, algoritmanın her bir adımında diğer KÇY tipleri ile de karşıt çözümler bulunur ve tüm bulunan çözümlerin karşılaştırması yapılır. İçlerinden en iyi olanı seçilir. Böylelikle çözüm uzayında daha kapsamlı arama yapan algoritmalar tasarlanmış olur. Hatta diğer KÇY tiplerinin uygun olduğu problemlerde, KÇY tipleri arasında da DKİ sezgiseli mantığı uygulanabilir. Bu şekilde iç içe DKİ sezgiseli içeren bir algoritma tasarlanabilir.

KAYNAKLAR

[1] P. Vansteenwegen, W. Souffriau, and K. Sörensen, "The travelling salesperson problem with hotel selection," *J. Oper. Res. Soc.*, vol. 63, no. 2, pp. 207–217, 2011, doi: 10.1057/jors.2011.18.

[2] M. Castro, K. Sörensen, P. Vansteenwegen, and P. Goos, "A simple GRASP + VND for the travelling salesperson problem with hotel selection, " *Faculty of Applied Economics*, vol. 24, 2012.

[3] P. Vansteenwegen, W. Souffriau, and K. Sörensen, "Solving the mobile mapping van problem: A hybrid metaheuristic for capacitated arc routing with soft time windows," *Comput. & Opns. Res*, vol. 37, no. 11, pp. 1870-1876, 2010.

[4] M. Castro, K. Sörensen, P. Vansteenwegen, and P. Goos, "A memetic algorithm for the travelling salesperson problem with hotel selection," *Computers & Operations Research*, vol. 40, no. 7, pp. 1716–1728, 2013, doi: 10.1016/j.cor.2013.01.006.

[5] M. Schneider, A. Stenger, and D. Goeke, "The electric vehicle-routing problem with time windows and recharging stations," *Transp. Sci.*, vol. 48, no. 4, pp. 500–520, Nov. 2013, doi: 10.1287/trsc.2013.0490.

[6] M. Castro, K. Sörensen, P. Vansteenwegen, and P. Goos, "A fast metaheuristic for the travelling salesperson problem with hotel selection," *4OR*, vol. 13, no. 1, pp. 15–34, 2014, doi: 10.1007/s10288-014-0264-5.

[7] M. M. Sousa, S. Ochi, I. M. Coelho, and L. B. Gonçalves, "A Variable Neighborhood Search Heuristic for the Traveling Salesman Problem with Hotel Selection," *Conferencia LatinoAmericana de Informatica*, pp. 1-12, 2015.

[8] M. Radmanesh, M. Kumar, A. Nemati, and M. Sarim, "Solution of Traveling Salesman Problem with Hotel Selection in the Framework of MILP-Tropical Optimization," *American Control Conference (ACC)*, pp. 6-8, 2016.

[9] Y. Lu, U. Benlic, and Q. Wu, "A hybrid dynamic programming and memetic algorithm to the Traveling Salesman Problem with Hotel Selection," *Comput. Oper.*

Res., vol. 90, pp. 193–207, 2017, doi: 10.1016/j.cor.2017.09.008.

[10] C. A. Gencel, "Otel Seçimli Gezgin Satıcı Problemi İçin Yeni Matematiksel Modeller," Yüksek Lisans Tezi, Başkent Üniversitesi Fen Bilimleri Enstitüsü, Ankara, 2019.

[11] M. M. Sousa and L. S. Ochi, "An Efficient Heuristic to the Traveling Salesperson Problem With Hotel Selection," *Hybrid Metaheuristics Conference*, vol. 2. pp. 31-45, 2019.

[12] T. Bektaş, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209-219, 2006.

[13] P. Toth, and D. Vigo, "The Vehicle Routing Problem," *Society for Industrial and Applied Mathematics*, vol. 9, 2002.

[14] J. F. Cordeau, M. Gendreau, and G. Laporte, "A tabu search heuristic for periodic and multi-depot vehicle routing problems," *Networks*, vol. 30, no. 2, pp. 105–119, 1997, doi: 10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G.

[15] M. Polacek, R. F. Hartl, K. Doerner, and M. Reimann, "A Variable Neighborhood Search for the Multi Depot Vehicle Routing Problem with Time Windows," *Journal of Heuristics*, vol. 10, no. 6. pp. 613–627, Dec-2004, doi: 10.1007/s10732-005-5432-5.

[16] G. Nagy, S. Salhi, "Location-routing: Issues, models and methods," *European Journal of Operational Research*, vol. 117, no. 2, pp. 649-672, 2007.

[17] T. Wu, C. Low, and J. Bai, "Heuristic solutions to multi-depot location-routing problems," *Comput Opns. Res.*, vol. 29, no. 10, pp. 1393-1415, 2002.

[18] C. Lin, C. Chow, and A. Chen, "A location-routing-loading problem for bill delivery services," *Comput Indust. Eng.*, vol. 43, no. 1-2, pp. 5-25, 2002.

[19] G. Pesant, M. Gendreau, J. Potvin, and J. Rousseau, "On the flexibility of constraint programming models: From single to multiple time windows for the traveling salesman problem," *European Journal of Operational Research*, vol. 117, no. 2, pp.

253-263, 1999.

[20] B. Kim, S. Kim, and S. Sahoo, “Waste collection vehicle routing problem with time windows,” *Comput. Oper. Res.*, vol. 33, no. 12, pp. 3624-3642, 2006.

[21] A. Benjamin, and J. Beasley, “Metaheuristics for the waste collection vehicle routing problem with time windows, driver rest period and multiple disposal facilities,” *Computers&Operations Research*, vol. 37, no. 12, pp. 2270-2280, 2010.

[22] B. Crevier, J. Cordeau, and G. Laporte, “The multi-depot vehicle routing problem with inter-depot routes,” *Eur. J. Oper. Res.*, vol. 176, no. 2, pp. 756-773, 2007.

[23] C. D. Tarantilis, E. E. Zachariadis, and C. T. Kiranoudis, “A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities,” *INFORMS J. Comput.*, vol. 20, no. 1, pp. 154–168, 2008, doi: 10.1287/ijoc.1070.0230.

[24] E. Angelelli, M. Speranza, “The periodic vehicle routing problem with intermediate facilities,” Eur. J. Oper. Res., vol. 137, no. 2, pp. 233-247, 2002.

[25] A. Baltz, M. E Ouali, G. Jäger, V. Sauerland, and A. Srivastav, “Exact and heuristic algorithms for the Travelling Salesman Problem with Multiple Time Windows and Hotel Selection,” *Journal Of The Operational Research Society*, vol. 66, no. 4, pp. 615–626, 2014, doi: 10.1057/jors.2014.17.

[26] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. V. Berghe, and D. V. Oudheusden, “A personalized tourist trip design algorithm for mobile tourist guides,” *Appl. Artif. Intell.*, vol. 22, no. 10, pp. 964–985, Nov. 2008, doi: 10.1080/08839510802379626.

[27] A. Divsalar, P. Vansteenwegen, K. Sörensen, and D. Cattrysse, “A Memetic Algorithm for the Orienteering Problem with Hotel Selection,” *Journal Of Operational Research*, vol. 237, no. 1, pp. 29-49, 2014,

[28] A. Toledo, and M. Riff, “HOPHS: A hyperheuristic that solves orienteering problem with hotel selection,” *International Conference on Digital Information Processing and Communications (ICDIPC)*, 2015.

[29] S. V. Hoek, "Tabu Search for the Orienteering Problem with Hotel Selection," *Econometrics and Operations Research*, 2016.

[30] E. Gencel., "Otel Seçimli Oryantiring Problemi İçin Yeni Matematiksel Modeller," Yüksek Lisans Tezi, Başkent Üniversitesi Fen Bilimleri Enstitüsü, Ankara, 2019.

[31] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," *MIT Press*, England ,2016.

[32] G. H. Chen, and D. Shah, "Explaining the Success of Nearest Neighbor Methods in Prediction," *Foundation and Trends in Machine Learning*, Now Publishers, pp. 1-250, 2018, doi: 10.1561/2200000064.

[33] T. M. Cover and P. E. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, pp. 21-27, 1967.

[34] D. E. Knuth, "The Art of Computer Programming, Volume 3, Sorting and Searching," *Adison-Wesley*, 1973

[35] T. M. Cover,b"This week's Citation Classic," *Current Contents*, 1982.

[36] H. Tizboosh, "Opposition-based learning: a new scheme for machine intelligence," *In Computational intelligence for modelling, control and automation 2005 and international conference on intelligent agents*, web technologies and internet commerce, vol. 1, pp. 695–701, 2005.

[37] A. R. Malisia, "Improving the exploration ability of ant-based algorithms," *Stud. Comput. Intell.*, vol. 155, pp. 121–142, 2008, doi: 10.1007/978-3-540-70829-2_7.

[38] N. Rojas-Morales, M. Rojas, E. Ureta, "A survey and classification of opposition-based metaheuristics," *Computers&Industrial Engineering*, pp. 424-435, 2017.

[39] H. R. Tizhoosh, M. Ventresca, and S. Rahnamayan, "Opposition-based computing," *Stud. Comput. Intell.*, vol. 155, pp. 11–28, 2008, doi: 10.1007/978-3-540-70829-2_2.

[40] H. Wang, Z. Wu, D. Rahnamayan, and L. Kang, "A scalability test for accelerated DE using generalized opposition-based learning," *IEEE*, pp. 1090–1095, 2009.

[41] S. Rahnamayan, and G. G. Wang, "Center-Based Sampling for Population-Based Algorithms," *In 2009 IEEE congress on evolutionary computation*, pp. 933-938, 2009, doi: 10.1109/CEC.2009.4983045.

[42] M. Ergezer, D. Simon, and D. Du, "Oppositional biogeography-based optimization," *SMC*, vol. 9, pp. 1009–1014, 2009, doi: 10.1109/CEC.2011.5949792.

[43] L. Wang, Q. Xu, B. He, and N. Wang, "Modified Opposition-Based Differential Evolution for Function Optimization," *J. Comput. Inf. Syst.*, vol. 7, no. 5, pp. 1582–1591, 2011, doi: 10.1073/pnas.1317360111.

[44] F. Zhao, J. Zhang, J. Wang, and C. Zhang, "A shuffled complex evolution algorithm with opposition-based learning for a permutation flow shop scheduling problem," *Int. J. Comput. Integr. Manuf.*, vol. 28, no. 11, pp. 1220–1235, 2014, doi: 10.1080/0951192X.2014.961965.

[45] P. Hansen, N. Mladenovic, J. Brimberg, and J. A. Perez, "Variable Neighborhood Search," *Int. Ser. Oper. Res. Manag. Sci.*, vol. 146, 2010, doi: 10.1007/978-1-4419-1665-5.

[46] P. Hansen, and N. Mladenovic, "Variable neighborhood search for the p-median," *Location Sci.*, vol. 5, pp. 207–226, 1997.

[47] P. Hansen and N. Mladenović, "An Introduction to Variable Neighborhood Search," in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer, Dordrecht, Springer US, 1999, pp. 433–458.

[48] P. Hansen and N. Mladenović, "Developments of Variable Neighborhood Search," *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, Boston/Dordrecht/London, 2001, pp. 415–439.

[49] P. Hansen and N. Mladenović, "Variable Neighborhood Search," *Handbook of Metaheuristics*, Kluwer Academic Publisher, 2003, pp. 145–184.

[50] N. Mladenovic, J. Petrovic, V. Kovacevic-Vujcic, and Cangalovic, M, "Solving spread spectrum radar polyphase code design problem by tabu search and variable neighbourhood search," *Eur. J. Oper. Res.*, vol. 151, pp. 389–399, 2003.

[51] N. Mladenović and P. Hansen, "Variable neighborhood search," *Comput. Oper. Res.*, vol. 24, no. 11, pp. 1097–1100, 1997, doi: 10.1016/S0305-0548(97)00031-2.

[52] P. Hansen, B. Jaumard, N. Mladenovic, and A. Parreira, "Variable neighborhood search for weighted maximum satisfiability problem." *Les Cahiers du GERAD G*, vol. 62, 2000.

[53] P. Hansen, N. Mladenović, and D. Perez-Britos, "Variable neighborhood decomposition search," *J. Heuristics*, vol. 7, no. 4, pp. 335–350, Jul. 2001, doi: 10.1023/A:1011336210885.

[54] «ANT/OR,» [Çevrimiçi]. Available: http://antor.uantwerpen.be/instances-in-the-paper-a-memetic-algorithm-for-the-travelling-salesperson-problem-with-hotel-selection/.

[55] M. M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *Oper. Res.*, vol. 35, no. 2, pp. 254–265, 1987, doi: 10.1287/opre.35.2.254.

[56] J.-F. Cordeau, G. Laporte, and A. Mercier, "A unified tabu search heuristic for vehicle routing problems with time windows," *J. Oper. Res. Soc.*, vol. 52, pp. 928–936, 2001, doi: 10.1057/palgrave.jors.2601163.

EKLER

EK 1: KÇTDKİ Sezgiseli Python Kodu

```
#Import Libraries
from random import *
import math
from copy import deepcopy
import time
from itertools import combinations
#Calculate Distance Between i-j
def CalculateDist(i,j):
    return math.trunc (math.sqrt ((XPos[i-1]-XPos[j-1])**2 + (YPos[i-1]-YPos[j-
1])**2)*10)/10 + T[j-1]
#Calculate Objective Function
def CalculateObj(SolutionIn):
    obj = 0.0
    for i in range(0,m):
        obj += CalculateTotDist(SolutionIn[i])
    return obj
#Calculate Objective Function with New m
def CalculateObjNewm(SolutionIn, newm):
    obj = 0.0
    for i in range(0,newm):
        obj += CalculateTotDist(SolutionIn[i])
    return obj
#Calculate Total Distance of Array
def CalculateTotDist(SubArrIn):
    TotDist = 0.0
    SizeofSubArrIn = len(SubArrIn)
    for i in range(0,SizeofSubArrIn-1):
        TotDist += CalculateDist(SubArrIn[i], SubArrIn[i+1])
    return TotDist
#Exchange Move
def Exchange(SolutionIn, r1, c1, r2, c2, EType, rcomb):
    SolutionOut = deepcopy(SolutionIn)
    lastr1 = len(SolutionOut[r1])-1
    lastr2 = len(SolutionOut[r2])-1
    TourI = SolutionOut[r1][1:c1]
    TourII = SolutionOut[r1][c1:lastr1]
    TourIII = SolutionOut[r2][1:c2]
    TourIV = SolutionOut[r2][c2:lastr2]
    SolutionOut[r1][1:lastr1] = []
    SolutionOut[r2][1:lastr2] = []
   for i in rcomb:
        if i == 'T1':
            TourI.reverse()
        if i == 'T2':
            TourII.reverse()
        if i == 'T3':
            TourIII.reverse()
        if i == 'T4':
            TourIV.reverse()
    if EType == 1:
        SolutionOut[r1][1:1] = TourII + TourI
        SolutionOut[r2][1:1] = TourIV + TourIII
    if EType == 2:
        SolutionOut[r1][1:1] = TourI + TourIII
        SolutionOut[r2][1:1] = TourII + TourIV
    if EType == 3:
        SolutionOut[r1][1:1] = TourI + TourIII
        SolutionOut[r2][1:1] = TourIV + TourII
    if EType == 4:
        SolutionOut[r1][1:1] = TourIII + TourI
        SolutionOut[r2][1:1] = TourII+ TourIV
    if EType == 5:
        SolutionOut[r1][1:1] = TourIII + TourI
        SolutionOut[r2][1:1] = TourIV + TourII
    if EType == 6:
        SolutionOut[r1][1:1] = TourI + TourIV
        SolutionOut[r2][1:1] = TourIII + TourII
    if EType == 7:
        SolutionOut[r1][1:1] = TourI + TourIV
        SolutionOut[r2][1:1] = TourII + TourIII
    if EType == 8:
        SolutionOut[r1][1:1] = TourIV + TourI
        SolutionOut[r2][1:1] = TourIII + TourII
    if EType == 9:
```

```
        SolutionOut[r1][1:1] = TourIV + TourI
        SolutionOut[r2][1:1] = TourII + TourIII
    ExSolisFeasible = 1
    for r in range(0,m):
        if CalculateTotDist(SolutionOut[r]) > C:
            ExSolisFeasible = 0
    if ExSolisFeasible == 1:
        return SolutionOut
    else:
        return SolutionIn
#Exchange Search
def ExchangeSearch(SolutionIn, EType):
    TourSections = ["T1" , "T2" , "T3" , "T4"]
    Combinations = []
    ReverseCombinations = []
    for i in range(1,5):
        Combinations.append(set(combinations(TourSections, i)))
    for i in Combinations:
        for j in i:
            ReverseCombinations.append(j)
    ObjInitial = CalculateObj(SolutionIn)
    ObjBest = ObjInitial
    SolutionOut = deepcopy(SolutionIn)
    for r1 in range(0,m):
        for r2 in range(0,m):
            for c1 in range(2,len(SolutionIn[r1])-1):
                for c2 in range(2,len(SolutionIn[r2])-1):
                    if r1 != r2:
                        for rcomb in ReverseCombinations:
                            ExchangedSolution = Exchange(SolutionIn, r1, c1, r2, c2, EType,
rcomb)
                            ObjExchangedSolution = CalculateObj(ExchangedSolution)
                            OppositeSolution = FindOppositeSolutionNodes(ExchangedSolution,
m)
                            ObjOppositeSolution = CalculateObj(OppositeSolution)
                            if min(ObjExchangedSolution,ObjOppositeSolution) < ObjBest:
                                ObjBest = min(ObjExchangedSolution,ObjOppositeSolution)
                                if ObjExchangedSolution < ObjOppositeSolution:
                                    SolutionOut = deepcopy(ExchangedSolution)
                                else:
                                    SolutionOut = deepcopy(OppositeSolution)
    return SolutionOut
#Find the Nearest Customer Index
def FindNearestCustIndx(customer1, DummyArr):
    NearestDist = math.inf
    for idx, customer2 in enumerate(DummyArr):
        Distance = CalculateDist(customer1,customer2)
        if Distance < NearestDist:
            NearestDist = Distance
            j = idx
    return j
#Find the Nearest Hotel Index
def FindNearestHotelIndx(i):
    NearestDist = math.inf
    for h in range(1,s+1):
        Distance = CalculateDist(i,h)
        if Distance < NearestDist:
            NearestDist = Distance
            nearest_h = h
    return nearest_h
#Find Opposite Solution for Hotels
def FindOppositeSolutionHotels(SolutionIn, newm):
    SolutionOut = deepcopy(SolutionIn)
    for r in range(0,newm-1):
        SolutionOut[r][len(SolutionOut[r])-1] = (s) + (1) -
SolutionOut[r][len(SolutionOut[r])-1]
        SolutionOut[r+1][0] = (s) + (1) - SolutionOut[r+1][0]
    return SolutionOut
#Find Opposite Solution for Nodes
def FindOppositeSolutionNodes(SolutionIn, newm):
    SolutionOut = deepcopy(SolutionIn)
    for r in range(0,newm):
        for c in range(1,len(SolutionOut[r])-1):
            SolutionOut[r][c] = (s+n) + (s+1) - SolutionOut[r][c]
    OpSolisFeasible = 1
    for r in range(0,newm):
        if CalculateTotDist(SolutionOut[r]) > C:
            OpSolisFeasible = 0
    if OpSolisFeasible == 1:
```

```
        return SolutionOut
    else:
        return SolutionIn
#Generate Initial Solution Nearest Neighbor
def GenerateInitialSolutionNN():
    DummyArr = []
    for i in range(s+1,s+n+1):
        DummyArr.append(i)
    global Solution, m
    Solution = []
    i = 0
    while True:
        TotDist = 0.0
        customer1 = 1
        Solution.append([])
        Solution[i].append(customer1)
        while len(DummyArr) != 0:
            j = FindNearestCustIndx(customer1,DummyArr)
            customer2 = DummyArr[j]
            if CalculateDist(customer1,customer2) + CalculateDist(customer2, 0) + TotDist
<= C:
                customer2 = DummyArr.pop(j)
                Solution[i].append(customer2)
                TotDist += CalculateDist(customer1,customer2)
                customer1 = customer2
            else:
                customer1 = 1
                Solution[i].append(customer1)
                i += 1
                break
        if len(DummyArr) == 0:
            Solution[i].append(1)
            break
    m = i+1
#Hotel Search
def HotelSearch(SolutionIn):
    ObjInitial = CalculateObj(SolutionIn)
    ObjBest = ObjInitial
    SolutionOut = deepcopy(SolutionIn)
    IsBestSolutionFound = False
    for i in range(1,m):
        for h in range(0,s):
            SolutionOut = deepcopy(SolutionIn)
            SolutionOut[i][0], SolutionOut[i-1][len(SolutionOut[i-1])-1] = h+1, h+1
            if CalculateTotDist(SolutionOut[i]) <= C and CalculateTotDist(SolutionOut[i-1])
<= C:
                #OppositeSolution = FindOppositeSolutionHotels(SolutionOut, m)
                #ObjOppositeSolution = CalculateObj(OppositeSolution)
                ObjChangedSolution = CalculateObj(SolutionOut)
                #if min(ObjChangedSolution,ObjOppositeSolution) < ObjBest:
                if ObjChangedSolution < ObjBest:
                    #ObjBest = min(ObjChangedSolution,ObjOppositeSolution)
                    ObjBest = ObjChangedSolution
                    BestSolutionOut = deepcopy(SolutionOut)
                    #if ObjChangedSolution < ObjOppositeSolution:
                    #    BestSolutionOut = deepcopy(SolutionOut)
                    #else:
                    #    BestSolutionOut = deepcopy(OppositeSolution)
                    IsBestSolutionFound = True
    if IsBestSolutionFound == True:
        SolutionOut = BestSolutionOut
    return SolutionOut
#Print Solution and Objective Function
def PrintSol(SolutionIn):
    print("*****************************************************************")
    for i in range(0,m):
        print(SolutionIn[i], "- %7.1f" % (CalculateTotDist(SolutionIn[i]))),
    print("*****************************************************************")
    print("Objective Function = %7.1f" % (CalculateObj(SolutionIn)))
    print("*****************************************************************")
#Read Instance File
def ReadInstanceFile(filename):
    instancefile = open(filename,'r')
    ElementsRemoved = ['s','n','m','C','XPos','YPos','T','=','[',']','];',';']
    global s,n,m,C,XPos,YPos,T
    #Index of XPos,YPos,T ==> 0,1,2,...,(s+n-1)
    s,n,m,C,XPos,YPos,T = 0,0,0,0,[],[],[]
    i=1
    for Line in instancefile:
```

```
        DataRaw = Line.split()
        DataClear = [item for item in DataRaw if item not in ElementsRemoved]
        if DataClear:
            if i == 1:
                s = int(DataClear[0].rstrip(';'))
            if i == 2:
                n = int(DataClear[0].rstrip(';'))
            if i == 3:
                m = int(DataClear[0].rstrip(';'))
            if i == 4:
                C = float(DataClear[0].rstrip(';'))
            if i == 5:
                for j in DataClear:
                    XPos.append(float(j))
            if i == 6:
                for j in DataClear:
                    YPos.append(float(j))
            if i == 7:
                for j in DataClear:
                    T.append(float(j))
        i=i+1
    instancefile.close()
#Shift Move
def Shift(SolutionIn, r1, c1, r2, c2):
    SolutionOut = deepcopy(SolutionIn)
    if r1 == r2:
        if c1 < c2:
            r1c1 = SolutionOut[r1][c1]
            SolutionOut[r2].insert(c2,r1c1)
            SolutionOut[r1].pop(c1)
        elif c2 < c1:
            r1c1 = SolutionOut[r1].pop(c1)
            SolutionOut[r2].insert(c2,r1c1)
    elif r1!=r2:
        r1c1 = SolutionOut[r1][c1]
        SolutionOut[r2].insert(c2,r1c1)
        SolutionOut[r1].pop(c1)
    if CalculateTotDist(SolutionOut[r1]) <= C and CalculateTotDist(SolutionOut[r2]) <= C:
        if len(SolutionOut[r1]) == 2:
            del SolutionOut[r1]
            firstrow = 0
            lastrow = len(SolutionOut)-1
            lastcolinlastrow = len(SolutionOut[lastrow])-1
            SolutionOut[0][0], SolutionOut[lastrow][lastcolinlastrow] = 1, 1
            for row in range(0,lastrow):
                nextrow = row + 1
                lastcolinrow = len(SolutionOut[row])-1
                firstcolinnextrow = 0
                hotelinrowend = SolutionOut[row][lastcolinrow]
                hotelinnextrowstart = SolutionOut[nextrow][firstcolinnextrow]
                if hotelinrowend != hotelinnextrowstart:
                    SolutionOut[row][lastcolinrow], SolutionOut[nextrow][firstcolinnextrow]
= hotelinrowend, hotelinrowend
                    ObjHotelEndtoStart = CalculateObjNewm(SolutionOut, lastrow+1)
                    SolutionOut[nextrow][firstcolinnextrow], SolutionOut[row][lastcolinrow]
= hotelinnextrowstart, hotelinnextrowstart
                    ObjHotelStarttoEnd = CalculateObjNewm(SolutionOut, lastrow+1)
                    if ObjHotelEndtoStart <= ObjHotelStarttoEnd:
                        SolutionOut[row][lastcolinrow],
SolutionOut[nextrow][firstcolinnextrow] = hotelinrowend, hotelinrowend
                    else:
                        SolutionOut[nextrow][firstcolinnextrow],
SolutionOut[row][lastcolinrow] = hotelinnextrowstart, hotelinnextrowstart
        return SolutionOut
    else:
        return SolutionIn
#Shift Search
def ShiftSearch(SolutionIn):
    ObjInitial = CalculateObj(SolutionIn)
    ObjBest = ObjInitial
    SolutionOut = deepcopy(SolutionIn)
    bestnewm = m
    for r1 in range(0,m):
        for r2 in range(0,m):
            for c1 in range(1,len(SolutionIn[r1])-1):
                for c2 in range(1,len(SolutionIn[r2])):
                    if r1 == r2:
                        if c1 != c2:
                            ShiftedSolution = Shift(SolutionIn, r1, c1, r2, c2)
```

```
                            ObjShiftedSolution = CalculateObj(ShiftedSolution)
                            OppositeSolution = FindOppositeSolutionNodes(ShiftedSolution,
m)
                            ObjOppositeSolution = CalculateObj(OppositeSolution)
                            if min(ObjShiftedSolution,ObjOppositeSolution) < ObjBest:
                                ObjBest = min(ObjShiftedSolution,ObjOppositeSolution)
                                if ObjShiftedSolution < ObjOppositeSolution:
                                    SolutionOut = deepcopy(ShiftedSolution)
                                else:
                                    SolutionOut = deepcopy(OppositeSolution)
                    elif r1 != r2:
                        ShiftedSolution = Shift(SolutionIn, r1, c1, r2, c2)
                        newm = len(ShiftedSolution)
                        ObjShiftedSolution = CalculateObjNewm(ShiftedSolution, newm)
                        OppositeSolution = FindOppositeSolutionNodes(ShiftedSolution, newm)
                        ObjOppositeSolution = CalculateObjNewm(OppositeSolution, newm)
                        if min(ObjShiftedSolution,ObjOppositeSolution) < ObjBest:
                            ObjBest = min(ObjShiftedSolution,ObjOppositeSolution)
                            if ObjShiftedSolution < ObjOppositeSolution:
                                SolutionOut = deepcopy(ShiftedSolution)
                            else:
                                SolutionOut = deepcopy(OppositeSolution)
                            bestnewm = newm

    return SolutionOut, bestnewm
#Swap Move
def Swap(SolutionIn, r1, c1, r2, c2):
    SolutionOut = deepcopy(SolutionIn)
    r1c1 = SolutionIn[r1][c1]
    r2c2 = SolutionIn[r2][c2]
    SolutionOut[r1][c1] = r2c2
    SolutionOut[r2][c2] = r1c1
    if CalculateTotDist(SolutionOut[r1]) <= C and CalculateTotDist(SolutionOut[r2]) <= C:
        return SolutionOut
    else:
        return SolutionIn
#Swap Search
def SwapSearch(SolutionIn):
    ObjInitial = CalculateObj(SolutionIn)
    ObjBest = ObjInitial
    SolutionOut = deepcopy(SolutionIn)
    for r1 in range(0,m):
        for r2 in range(0,m):
            if r2 >= r1:
                for c1 in range(1,len(SolutionIn[r1])-1):
                    for c2 in range(1,len(SolutionIn[r2])-1):
                        if r1 == r2:
                            if c2 > c1:
                                SwappedSolution = Swap(SolutionIn, r1, c1, r2, c2)
                                ObjSwappedSolution = CalculateObj(SwappedSolution)
                                OppositeSolution =
FindOppositeSolutionNodes(SwappedSolution, m)
                                ObjOppositeSolution = CalculateObj(OppositeSolution)
                                if min(ObjSwappedSolution,ObjOppositeSolution) < ObjBest:
                                    ObjBest = min(ObjSwappedSolution,ObjOppositeSolution)
                                    if ObjSwappedSolution < ObjOppositeSolution:
                                        SolutionOut = deepcopy(SwappedSolution)
                                    else:
                                        SolutionOut = deepcopy(OppositeSolution)
                        elif r1 != r2:
                            SwappedSolution = Swap(SolutionIn, r1, c1, r2, c2)
                            ObjSwappedSolution = CalculateObj(SwappedSolution)
                            OppositeSolution = FindOppositeSolutionNodes(SwappedSolution,
m)
                            ObjOppositeSolution = CalculateObj(OppositeSolution)
                            if min(ObjSwappedSolution,ObjOppositeSolution) < ObjBest:
                                ObjBest = min(ObjSwappedSolution,ObjOppositeSolution)
                                if ObjSwappedSolution < ObjOppositeSolution:
                                    SolutionOut = deepcopy(SwappedSolution)
                                else:
                                    SolutionOut = deepcopy(OppositeSolution)
    return SolutionOut
#Trip Union Move
def TripUnion(SolutionIn, r1, r2):
    SolutionOut = deepcopy(SolutionIn)
    Croppedr1 = SolutionOut[r1][1:len(SolutionOut[r1])-1]
    Lastofr2 = SolutionOut[r2][len(SolutionOut[r2])-1]
    SolutionOut[r2] = SolutionOut[r2][:len(SolutionOut[r2])-1] + Croppedr1
    SolutionOut[r2].append(Lastofr2)
```

```
    if CalculateTotDist(SolutionOut[r2]) <= C:
        if r1 == 0:
            SolutionOut[r1+1][0] = 1
        if r1 == m-1:
            SolutionOut[m-2][len(SolutionOut[m-2])-1] = 1
        else:
            Lowerr1lastelement = SolutionOut[r1-1][len(SolutionOut[r1-1])-1]
            Upperr1firstelement = SolutionOut[r1+1][0]
            if Lowerr1lastelement != Upperr1firstelement:
                SolutionOut[r1-1][len(SolutionOut[r1-1])-1] = Upperr1firstelement
                ObjLowerr1 = CalculateTotDist(SolutionOut[r1-1])
                ObjUpperr1 = CalculateTotDist(SolutionOut[r1+1])
                Obj1 = ObjLowerr1 + ObjUpperr1
                SolutionOut[r1-1][len(SolutionOut[r1-1])-1] = Lowerr1lastelement
                SolutionOut[r1+1][0] = Lowerr1lastelement
                ObjLowerr1 = CalculateTotDist(SolutionOut[r1-1])
                ObjUpperr1 = CalculateTotDist(SolutionOut[r1+1])
                Obj2 = ObjLowerr1 + ObjUpperr1
                if Obj1 <= Obj2:
                    SolutionOut[r1-1][len(SolutionOut[r1-1])-1] = Upperr1firstelement
                    SolutionOut[r1+1][0] = Upperr1firstelement
                else:
                    SolutionOut[r1-1][len(SolutionOut[r1-1])-1] = Lowerr1lastelement
                    SolutionOut[r1+1][0] = Lowerr1lastelement
        del SolutionOut[r1]
        return SolutionOut, m - 1
    else:
        return SolutionIn, m
#Trip Union Search
def TripUnionSearch(SolutionIn):
    ObjInitial = CalculateObj(SolutionIn)
    ObjBest = ObjInitial
    SolutionOut = deepcopy(SolutionIn)
    bestnewm = m
    for r1 in range(0,m):
        for r2 in range(0,m):
            if r1 != r2:
                UnifiedSolution, newm = TripUnion(SolutionIn, r1, r2,)
                ObjUnifiedSolution = CalculateObjNewm(UnifiedSolution, newm)
                OppositeSolution = FindOppositeSolutionNodes(UnifiedSolution, newm)
                ObjOppositeSolution = CalculateObjNewm(OppositeSolution, newm)
                if min(ObjUnifiedSolution,ObjOppositeSolution) < ObjBest:
                    ObjBest = min(ObjUnifiedSolution,ObjOppositeSolution)
                    if ObjUnifiedSolution < ObjOppositeSolution:
                        SolutionOut = deepcopy(UnifiedSolution)
                    else:
                        SolutionOut = deepcopy(OppositeSolution)
                    bestnewm = newm
    return SolutionOut, bestnewm
# ***************************** M A I N  *  P R O G R A M M E
*****************************
def main():
    import math
    global Solution, m
    instanceslist = open("instances3.list", 'r')
    resultsfile = open("results_set3.txt", 'w')
    for instancename in instanceslist.readlines():
        ReadInstanceFile("./instances/set3/" + instancename.rstrip('\n') + ".dat")
        GenerateInitialSolutionNN()
        #GenerateInitialSolutionRandom()
        #GenerateInitialSolutionCWS()
        #GenerateInitialSolutionRAW()
        #PrintSol(Solution)
        ObjCurrentSolution = CalculateObj(Solution)
        SearchMethod = 0
        AlgorithmStartTime = time.time()
        it = 0
        while SearchMethod <= 13:
            it += 1
            SearchMethod += 1
            SearchStartTime = time.time()
            if SearchMethod == 1:
                NewSolution, newm = ShiftSearch(Solution)
                m = newm
                ObjNewSolution = CalculateObj(NewSolution)
                if ObjNewSolution < ObjCurrentSolution:
                    SearchMethod = 0
                    ObjCurrentSolution = ObjNewSolution
                    Solution = deepcopy(NewSolution)
```

```
                else:
                    SearchMethod += 1
            if SearchMethod == 2:
                NewSolution = SwapSearch(Solution)
                ObjNewSolution = CalculateObj(NewSolution)
                if ObjNewSolution < ObjCurrentSolution:
                    SearchMethod = 0
                    ObjCurrentSolution = ObjNewSolution
                    Solution = deepcopy(NewSolution)
                else:
                    SearchMethod += 1
            if SearchMethod == 3:
                NewSolution = HotelSearch(Solution)
                ObjNewSolution = CalculateObj(NewSolution)
                if ObjNewSolution < ObjCurrentSolution:
                    SearchMethod = 0
                    ObjCurrentSolution = ObjNewSolution
                    Solution = deepcopy(NewSolution)
                else:
                    SearchMethod += 1
            if SearchMethod == 4:
                NewSolution, newm = TripUnionSearch(Solution)
                m = newm
                ObjNewSolution = CalculateObj(NewSolution)
                if ObjNewSolution < ObjCurrentSolution:
                    SearchMethod = 0
                    ObjCurrentSolution = ObjNewSolution
                    Solution = deepcopy(NewSolution)
                else:
                    SearchMethod += 1
            if SearchMethod in range(5,14):
                NewSolution = ExchangeSearch(Solution, SearchMethod-4)
                ObjNewSolution = CalculateObj(NewSolution)
                if ObjNewSolution < ObjCurrentSolution:
                    SearchMethod = 0
                    ObjCurrentSolution = ObjNewSolution
                    Solution = deepcopy(NewSolution)
                else:
                    SearchMethod += 1
            SearchEndTime = time.time()
            print("Iteration : %3d   Obj : %7.1f   Search Time : %7.2f sec." % (it,
ObjCurrentSolution, SearchEndTime - SearchStartTime))
            AlgorithmEndTime = time.time()
        #PrintSol(Solution)
        print("**********************************************************************")
        print("Problem : %9s  -  Objective : %7.1f  -  Time : %7.2f sec." % (instancename,
CalculateObj(Solution), AlgorithmEndTime - AlgorithmStartTime))
        print("**********************************************************************")
        resultsfile.write("%9s %7.1f %7.2f\n" % (instancename, CalculateObj(Solution),
AlgorithmEndTime - AlgorithmStartTime))
    resultsfile.close()
    instanceslist.close()
#
*******************************************************************************************
if __name__ == "__main__":
    main()
```

EK 2: Tezde Kullanılan a_280.s3 Problemi Parametrelerinin Python Koduna Uygun Hale Getirilmiş Gösterimi

s = 4;

n = 280;

m = 5;

C = 665;

XPos = [288 148 48 212 288 270 256 256 246 236 228 228 220 212 204 196 188 196 188 172 164 156 148 140 148 164 172 156 140 132 124 116 104 104 104 90 80 64 64 56 56 56 56 56 56 56 40 40 40 40 40 40 40 32 32 32 32 32 32 32 32 32 40 56 56 48 40 32 32 24 16 16 8 8 8 8 8 8 8 16 8 8 24 32 32 32 32 32 32 40 40 40 40 44 44 44 32 24 16 16 24 32 44 56 56 56 56 56 64 72 72 56 48 56 56 48 48 56 56 48 56 56 104 104 104 104 104 104 104 116 124 132 132 140 148 156 164 172 172 172 172 172 172 180 180 180 180 180 172 172 172 172 164 148 124 124 124 124 124 124 104 104 104 104 104 104 104 104 104 92 80 72 64 72 80 80 80 88 104 124 124 132 140 132 124 124 124 124 124 132 124 120 128 136 148 162 156 172 180 180 172 172 172 180 180 188 196 204 212 220 228 228 236 236 236 228 228 236 236 228 228 236 236 228 228 236 252 260 260 260 260 260 260 260 276 276 276 276 284 284 284 284 284 284 284 284 288 280 276 276 276 268 260 252 260 260 236 228 228 236 236 228 228 228 228 220 212 204 196 188 180 180 180 180 180 196 204 212 220 228 236 246 252 260 280];

YPos = [149 85 83 169 129 133 141 157 157 169 169 161 169 169 169 169 169 161 145 145 145 145 145 145 169 169 169 169 169 169 169 161 153 161 169 165 157 157 165 169 161 153 145 137 129 121 121 129 137 145 153 161 169 169 161 153 145 137 129 121 113 113 113 105 99 99 97 89 89 97 109 109 97 89 81 73 65 57 57 49 41 45 41 49 57 65 73 81 83 73 63 51 43 35 27 25 25 25 17 17 17 11 9 17 25 33 41 41 41 49 49 51 57 65 63 73 73 81 83 89 97 97 105 113 121 129 137 145 145 145 145 137 137 137 137 137 125 117 109 101 93 85 85 77 69 61 53 53 61 69 77 81 85 85 93 109 125 117 101 89 81 73 65 49 41 33 25 17 9 9 9 21 25 25 25 41 49 57 69 77 81 65 61 61 53 45 37 29 21 21 9 9 9 9 9 25 21 21 29 29 37 45 45 37 41 49 57 65 73 69 77 77 69 61 61 53 53 45 45 37 37 29 29 21 21 21 29 37 45 53 61 69 77 77 69 61 53 53 61 69 77 85 93 101 109 109 101 93 85 97 109 101 93 85 85 85 93 93 101 101

109 117 125 125 117 109 101 93 93 101 109 117 125 145 145 145 145 145 145
141 125 129 133];

T = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0];