T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
EKONOMETRİ ANABİLİM DALI
YÖNEYLEM BİLİM DALI

AYLAK ZAMANI ENKÜÇÜKLEYEN TUR OLUŞTURMA PROBLEMLERİNİN GERİ İZLEME YÖNTEMİ İLE ÇÖZÜMÜNE İLİŞKİN BİR YAZILIM GELİŞTİRME UYGULAMASI

(DOKTORA TEZİ)

Şahin İNANÇ

BURSA – 2019

T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
EKONOMETRİ ANABİLİM DALI
YÖNEYLEM BİLİM DALI

AYLAK ZAMANI ENKÜÇÜKLEYEN TUR OLUŞTURMA PROBLEMLERİNİN GERİ İZLEÖME YÖNTEMİ İLE ÇÖZÜMÜNE İLİŞKİN BİR YAZILIM GELİŞTİRME UYGULAMASI

(DOKTORA TEZİ)

Şahin İNANÇ

Danışman

Prof. Dr. H.Kemal SEZEN

BURSA 2019

T. C.
BURSA ULUDAĞ ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ MÜDÜRLÜĞÜNE

.....Ekonometri..... Anabilim/Ana sanat Dalı,
.....Yöneylem..... Bilim Dalı'nda .....
numaralı ...711017021..... ...Şahin İNANIÇ.....'nın
hazırladığı Aylak Zamanı Frekansa Bağlı Tür Dustuma Problemlerinin
Geri İzleme Yöntemi ile Çözümüne İlişkin Bir Yazılım Geliştirme
Uygulaması.." konulu .....Doktora..... (Yüksek Lisans/Doktora/Sanatta Yeterlik
Tezi/Çalışması) ile ilgili tez savunma sınavı, 16/01/ 2019 günü 13:00 -
15:00....saatleri arasında yapılmış, sorulan sorulara alınan cevaplar sonunda adayın
tezinin/çalışmasının ...BAŞARILI..... (başarılı/~~başarısız~~) olduğuna
...oybirliği..... (oybirliği/~~oy çokluğu~~) ile karar verilmiştir.

Üye (Tez Danışmanı ve Sınav Komisyonu Başkanı)
Akademik Unvanı, Adı Soyadı
Üniversitesi
Prof. Dr. H. Kemal SEZEN

Üye
Akademik Unvanı, Adı Soyadı
Üniversitesi
Prof. Dr. Şenkal AYTAÇ

Üye
Akademik Unvanı, Adı Soyadı
Üniversitesi
Dr. Öğretim Üyesi
Mahmut ATLAS
Anadolu Üniversitesi

Üye
Akademik Unvanı, Adı Soyadı
Üniversitesi
Prof. Dr. Kadir Yasin ERYİĞİT

Üye
Akademik Unvanı, Adı Soyadı
Üniversitesi
Doç. Dr. Arzu EREN SENARAS

SOSYAL BİLİMLER ENSTİTÜSÜ

YÜKSEK LİSANS/DOKTORA İNTİHAL YAZILIM RAPORU

BURSA ULUDAĞ ÜNİVERSİTESİ

SOSYAL BİLİMLER ENSTİTÜSÜ

TEMEL İSLAM BİLİMLERİ ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 31/12/2018

Tez Başlığı / Konusu: **Aylak Zamanı Enküçükleyen Tur Oluşturma Problemlerinin Çözümüne İlişkin Bir Yazılım Geliştirme Uygulaması**

Yukarıda başlığı gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler ve d) Sonuç kısımlarından oluşan toplam ...68... sayfalık kısmına ilişkin, 31./12./2018 tarihinde şahsım tarafından .....Turnitin..... adlı intihal tespit programından (Turnitin)* aşağıda belirtilen filtrelemeler uygulanarak alınmış olan özgünlük raporuna göre, tezimin benzerlik oranı % 11'tür.

Uygulanan filtrelemeler:

1- Kaynakça hariç
2- Alıntılar hariç/dahil
3- 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Bursa Uludağ Üniversitesi Sosyal Bilimler Enstitüsü Tez Çalışması Özgünlük Raporu Alınması ve Kullanılması Uygulama Esasları'nı inceledim ve bu Uygulama Esasları'nda belirtilen azami benzerlik oranlarına göre 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.

Gereğini saygılarımla arz ederim.

31 /12/2018

Adı Soyadı: Şahin İNANÇ
Öğrenci No: 711017001
Anabilim Dalı: Ekonometri
Programı: DOKTORA
Statüsü: ☐ Y.Lisans ☒ Doktora

Danışman

Prof. Dr. H.Kemal SEZEN

YEMİN METNİ

Doktora tezi olarak sunduğum "Aylak Zamanı Enküçükleyen Tur Oluşturma Problemlerinin Geri İzleme Yöntemi ile Çözümüne İlişkin Bir Yazılım Geliştirme Uygulaması" başlıklı çalışmanın bilimsel araştırma, yazma ve etik kurallarına uygun olarak tarafımdan yazıldığına ve tezde yapılan bütün alıntıların kaynaklarının usulüne uygun olarak gösterildiğine, tezimde intihal ürünü cümle veya paragraflar bulunmadığına şerefim üzerine yemin ederim.

Tarih ve İmza

04.02.2019

Adı Soyadı:	Şahin İNANÇ
Öğrenci No:	711017001
Anabilim Dalı:	Ekonometri
Programı:	Yöneylem
Statüsü:	Doktora

ÖZET

Yazar Adı ve Soyadı : Şahin İNANÇ
Üniversite : Bursa Uludağ Üniversitesi
Enstitü : Sosyal Bilimler Enstitüsü
Anabilim Dalı : Ekonometri
Bilim Dalı : Yöneylem

Tezin Niteliği : Doktora Tezi
Sayfa Sayısı : xiii + 110
Mezuniyet Tarihi : …. / …. / 20……..

Tez Danışman(lar)ı : Prof. Dr. H.Kemal SEZEN

AYLAK ZAMANI ENKÜÇÜKLEYEN TUR OLUŞTURMA PROBLEMLERİNİN GERİ İZLEME YÖNTEMİ İLE ÇÖZÜMÜNE İLİŞKİN BİR YAZILIM GELİŞTİRME UYGULAMASI

İşletmelerde gün geçtikçe artan rekabet koşullarında küçük avantajlar bile önemli olabiliyorken lojistik de her geçen gün daha da büyük bir öneme sahip olmaktadır. İşletmelerin giderlerinin önemli bir kısmını lojistik oluşturmaktadır.

Bu çalışma karayolunda yolcu taşıması yapan bir lojstik şirketinin araçlarının seferleri arasında beklemelerine ilişkin aylak zaman toplamını en küçük kılacak şekilde, optimal çözümü garanti eden bir (kesin, exact) yöntemle çözümü gerçekleştirilmiştir. Çözüme yönelik C# programlama dili ile bir yazılım geliştirilmiştir. Problemin çözümüne ilişkin yöntem olarak Dal ve Sınır tekniğinin Geri İzleme yaklaşımı kullanılmıştır. Kullanılan bu yöntem; tam sayıma göre sayımlamayı azaltma özelliğine sahiptir.

Anahtar Sözcükler:
Araç Rotalama Problemi, Gezgin Satıcı Problemi, Dal ve Sınır Yöntemi, Geri İzleme Yöntemi, C#

ABSTRACT

Name and Surname	: Şahin İNANÇ
University	: Bursa Uludag University
Institution	: Social Science Institution
Field	: Econometrics
Branch	: Operation Research
Degree Awarded	: PhD
Page Number	: xiii+110
Degree Date	: …. / …. / 20……..
Supervisor (s)	: Prof.Dr.H.Kemal SEZEN

A SOFTWARE DEVELOPMENT APPLICATION FOR SOLVING ROTATING PROBLEM USING BACKTRACKING TECHNIQUE

Even a small advantages can be important in the competition conditions that are increasing day by day in the enterprises, logistics is becoming more important with each passing day. A significant part of the expenses of enterprises constitute logistics.

This study was carried out with a (exact, exact) method, which guarantees the optimal solution so as to minimize the total time spent waiting for the vehicles of a lojstik company carrying road passenger. The software was developed with the C# programming language for the solution. The method of solution of the problem was followed by the Backtracking approach of the Branch and Bound technique. This method used; has the ability to reduce total completed counting.

Keywords:
Vehicle Routing Problem, Traveling Salesman Problem, Branch and Bound Technique, Backtracking Approach, C#

ÖNSÖZ

Bu çalışma, günümüz rekabet koşullarında işletmeler için gittikçe daha önemli birer problem haline gelen taşıma ve dağıtım unsurlarının maliyetlerini azaltma sorununa, Gezgin Satıcı probleminin bir türü olarak ele alınabilen özgün bir Araç Rotlama Probleminnin "Dal ve Sınır Tekniği-Geri İzleme Yöntemi" yoluyla çözüm bulmayı amaçlamıştır.

Araştırmalarım sırasında her zaman ve her konuda sonsuz desteğinden dolayı tez danışmanım Prof. Dr. H. Kemal SEZEN'e şükran ve teşekkürlerimi sunarım. İlgisini ve desteğini hiçbir zaman esirgemeyen, çalışmam sırasında bana sağladığı katkılar ve yardımlarından ötürü sevgili eşim Kübra İNANÇ'a sonsuz sevgimi ve teşekkürlerimi sunmayı borç bilirim.

BURSA, 2019 **Şahin İNANÇ**

İÇİNDEKİLER

ÜÇÜNCÜ BÖLÜM

ARAÇ ROTALAMA PROBLEMLERİ ÇÖZÜM YÖNTEMLERİ

DÖRDÜNCÜ BÖLÜM

UYGULAMA

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR LİSTESİ

Bibliyografik Bilgiler	Uluslararası	Türkçe
Cilt	Vol	C.
Sayı	No	S.
Sayfa	pp.	ss.
Ve diğerleri		vd
Ve benzei		vb
Vesaire		vs

BİRİNCİ BÖLÜM

GİRİŞ

Gün geçtikçe değişen ve gelişen küresel rekabet ortamında işletmelerin varlıklarını sürdürmesi ve rekabet avantajı elde etmesi için maliyet azaltma ile ilgili çözüm yolları aramasıyla, ulaştırma konusu giderek önem kazanmış, önemli bir rekabet unsuru haline gelmiştir. Sınırlı mallar ve ulaşım kaynakları, yüksek planlama karmaşıklığı ve lojistik hizmet sağlayıcıları arasındaki güçlü rekabet sayesinde artan maliyet baskısı, (Caric vd, 2008: 2) müşteri hizmetini optimize etmek için entegre lojistik sistemleri birincil ihtiyaç haline gelmiştir. İşletmeler ulaştırma ve taşıma maliyetlerini minimize etmek, aynı zamanda müşteri ihtiyaçlarının zamanında ve tam olarak karşılanması gibi birbiri ile çelişen problemlerin çözüm yolu arayışına girmiştir. Bu gibi karmaşık problemlerin çözümünde en çok kullanılan yöntemlerden biri, bir dağıtım şebekesinde araç filosuyla bir dizi müşteriye hizmet etmek isteyen bir kombinasyonel optimizasyon problemi olan "Araç Rotalama Problemi" optimizasyonudur. Araç Rotalama Problemi (ARP), bir veya birkaç depodan, coğrafi olarak dağınık şehirlere veya müşterilere, yan kısıtlamalara tabi olmak üzere, en uygun teslimat veya toplama güzergâhlarının tasarlanması problemi olarak tanımlanabilir. (Laporte, 1992: 1). Araç Rotalama Problemlerinde (ARP) amaç genellikle zamanı ya da mesafeyi veya birden çok kısıtı birden minimize etmektir. Bu kısıtlara bağlı kalınarak müşterilerin ihtiyaçları vaktinde karşılanmaya çalışılır. Bu sorun, işletmelerin bir dizi coğrafi olarak dağılmış müşteriye ulaştırılması için bir teslimat filosunun sağlanmasıyla ilgili zaman ve maliyetler nedeniyle işletmeler için ekonomik açıdan önemlidir. Buna ek olarak, bu tür sorunlar, otobüs sistemlerinde, posta taşıyıcılarında ve diğer kamu hizmet araçlarında araç rotalarının belirlenmesini gerektiren kamu sektöründe de önemlidir. Bu örneklerin her birinde, sorun tipik olarak bir tedarik yerinden bir takım müşteri lokasyonlarına teslimatı kolaylaştırmak için bir takım araçlar için birleştirilmiş yolların minimum maliyetini bulmayı içerir. Maliyet, mesafe ile yakından ilişkili olduğundan, bir şirket müşteri talebini karşılamak için birtakım araçların kat ettiği asgari mesafeyi

bulmaya çalışabilir. Bunu yaparken, firma, beklenen müşteri hizmet düzeyini artırırken veya en azından koruyarak maliyetleri en aza indirmeye çalışır (Bell vd, 2004: 41).

ARP ilk olarak George Bernard Dantzig ve John Ramser tarafından 1959 yılında ele alınmıştır. Dantzig ve Ramser çalışmalarında ayrı yerlerde bulunan servis istasyonlarına akaryakıt dağıtım problemini ele alıp çözüm için bir matematiksel programlama modeli geliştirmişler ve algoritmik bir yaklaşım ortaya koymuşlardır (Dantzig vd, 1959).

ARP, fiziksel dağıtım ve lojistik alanlarında merkezi bir rol oynamaktadır (Laporte, 1992: 345). Araç Yönlendirme Problemleri hakkında yazılan 1000'den fazla makale bu NP-Zor optimizasyon probleminin pratik ve teorik önemini göstermektedir (Caric vd, 2008: 1). NP tipi problemler mümkün bütün çözümlerin denenerek bulunması ile çözülebilen problem türüdür. Burada çözüm için gerekli zaman üstel olarak artmaktadır (Çetin, 2007: 16).

ARP için yayımlanan çoğu araştırma, sezgisel gelişime odaklanmıştır. Her ne kadar modern buluşların gelişimi önemli bir ilerlemeye yol açsa da, gelişmiş performans arayışı devam etmektedir.

1.1. ÇALIŞMANIN AMACI

Bu çalışmada aylak zamanı en küçükleyen tur oluşturma probleminin optimal çözümünün bulunması amaçlanmıştır. Örnek uygulama bir otobüs şirketine ilişkindir.

Çalışmada amaç belirli bir noktadan hareket eden otobüsün uğraması gereken tüm noktalara uğrayarak başlangıç noktasına geri dönmesi ve bu esnada noktalardaki beklemeler toplamını en küçük kılacak, bir turun oluşturulmasıdır.

Temel araç rotalama problemi, her biri teslim edilecek malların belirli bir ağırlığını gerektiren bir dizi müşteriden oluşur. Tek bir depodan teslim edilen araçlar, gerekli malları teslim etmeli, daha sonra depoya geri dönmelidir. Her araç sınırlı bir ağırlık taşıyabilir ve aynı zamanda seyahat edebileceği toplam mesafeyle de sınırlandırılabilir. Her müşteriyi sadece bir araç ziyaret edebilir. Sorun, bu gereklilikleri karşılayan ve minimum toplam maliyet sağlayan bir dizi optimal sıralamanın belirlenmesidir. Pratikte, bu genellikle, seyahat edilen toplam mesafeyi en aza

indirmeye veya kullanılan araç sayısını en aza indirmeye ve daha sonra bu araç sayısı için toplam mesafeyi en aza indirmeye eşdeğer olarak kabul edilir.

1.2. PROBLEMİN ÇÖZÜM YÖNTEMİ

Problemin çözümü için Dal ve Sınır yönteminin Geri İzleme yaklaşımı kullanılmıştır. Geri İzleme yaklaşımı ile Dal ve Sınır algoritmasına göre olası bütün yolların denenmesine gerek kalmadan çözüme daha çabuk ulaşılabilmektedir (Sezen, 2017: 140). Örnek bir otobüs şirketi üzerinde uygulanan yöntemde, belirli bir başlangıç noktasından hareket eden aracın daha önce ziyaret ettiği bir şehre ya da bir düğüme tekrar uğramayarak ve minimum miktarda bekleme yaparak turu tamamlayıp başladığı noktaya tekrar dönmesi hedeflenmektedir.

Problemin ayrıntılı çözümünde iki şehir arasındaki seferler birden fazla olabilmektedir. Bu nedenle o şehre tanımlanmış sefer olduğu sürece aynı şehre seferler bitene kadar tekrar tekrar uğranabilir. Aynı şehre tanımlanmış sefer sayısından fazla uğranamaz. Diğer bir deyişle her bir sefer için şehirler ayrı düğümler gibi farz edilmelidir. Bu nedenle şehirlere birden çok kez uğransa dahi problem yine gezgin satıcı problemi türündedir.

Çözüm yöntemi için Dal ve Sınır Geri İzleme yaklaşımı tercih edilmiştir. Bu yöntemin tercih edilmesinin ilk sebebi en iyi çözümü bulmayı garanti etmesidir. İkinci sebebi ise diğer kesin çözüm yöntemlerine göre sayımlamayı büyük oranda azaltması ve böylece optimal çözüme daha çabuk ulaşılabilmesidir.

Uygulama çalışmasında problemin çözümüne ilişkin C# programlama dilinde bir yazılım geliştirilmiştir. Yazılımda Dal ve Sınır yöntemi Geri İzleme yaklaşımı kullanılmıştır.

1.3. LİTERATÜR TARAMASI

Araç rotalama problemeri ilk olarak George Bernard Dantzig ve John Ramser tarafından 1959 yılında ele alınmıştır. Çalışmaları, bir benzin terminali filosunun bir merkezi terminal ile terminal tarafından tedarik edilen çok sayıda servis istasyonu arasında en uygun şekilde rotalanması ile ilgilidir. Sistemdeki iki nokta arasındaki en kısa yollar, dağıtım sistemi içindeki istasyon sayısı verilmiştir. Problemde istasyonlara

istinaden ve filo tarafından katedilen toplam yolun en az olacağı şekilde, kamyonlara istasyonlar tahsis etmenin bir yolu bulunmak istenmektedir. Problemin çözümünde optimal bir çözüm elde etmek için doğrusal programlama formülasyonuna dayanan bir denklem ele alınmıştır. Problemde tanımlanan hesaplamalar elle veya otomatik bir dijital hesaplama makinesi ile kolayca gerçekleştirilebilir. Yöntemin pratik uygulamaları henüz yapılmamıştır. Bununla birlikte, bir dizi deneme problemi hesaplanmıştır (Dantzig vd, 1959: 80).

Laporte vd. (1983), çalışmalarında, ziyaret edilecek her bir şehre negatif olmayan bir ağırlığın verildiği ve tüm araçların aynı olduğu ve aynı kapasiteye sahip olduğu araç yönlendirme probleminin bir versiyonunu ele almaktadır. Problemi tamsayılı programlama ile modellenip Dal ve Sınır yöntemi ile çözülmüştür. 15 ile 50 arasında değişen şehir (düğüm) sayısı için problemin kesin çözümü bulunmuştur.

Song vd. (2015), çalışmalarında, doğrusal olmayan bir matematiksel model ve sezgisel bir algoritma ile bozulabilir gıda ürünleri dağıtımı için hem soğutmalı hem de genel tip araçların bulunduğu bir araç rotası sorunu ele alınmıştır. Bu çalışmanın amacı, bozulabilen gıda ürünlerinin teslimi için soğutmalı tip aracın performansını ve kullanılabilirliğini doğrulamaktır.

Hokama vd. (2016), çalışmalarında, araç rotalama problemlerini boşaltma kısıtı altında Dal Kesme yöntemini kullanarak problemin çözümü için algoritma geliştirmişlerdir.

Montoya vd. (2015), çalışmalarında, yeşil araç rotalama problemi (Yeşil ARP), alternatif yakıt araçları (AYA'ler) kullanılarak rotalama problemini çözmüşlerdir. Problemde AYA'ler sınırlı tank kapasitesine sahip olduğundan, güzergâhlar sadece alternatif yakıt istasyonlarından (AYİ'ler) geçebilirler. Montoya vd. yeşil ARP ile başa çıkmak için basit ama etkili iki aşamalı bir buluşsal yöntem önermişlerdir.

Barkaoui vd. (2015), çalışmalarında, müşteri memnuniyetini artırmaya yönelik bir strateji sunmuşlardır. Önerilen yeni yöntemde, daha önce ziyaret edilmiş müşterilerin memnuniyetini artırmak için zaman penceresiyle dinamik araç rotalamanın birleştirilmesi ile tasarlanmış hibrit bir genetik algoritma kullanılarak gerçekleştirilmiştir. Simülasyonlar, yeni stratejiyi kullanan revize edilmiş algoritmanın

değerini karşılaştırmakta ve bunun müşteri memnuniyeti üzerindeki etkisini açıkça göstermektedir.

Dastghaibifard vd. (2008), çalışmalarında, ARP için yeni bir paralel Dal ve Sınır algoritması önerilmiştir. Bu algoritmada, çok işlemcili paylaşılan bellek yerine çok bilgisayarlı ve dinamik yük dengeleyici bir yaklaşım kullanılmış. Problem türü olarak kapasiteli ARP seçilip yeni yöntem denenmiştir. Buldukları sonuçlar diğer algoritmalara göre daha başarılı olduğu görülmüş.

Liu vd. (2008), çalışmalarında, basit montaj hattı dengeleme tip I probleminin çözümü için kesin algoritmalar örnermişlerdir. Önerilen algoritmalar yapıcı ve iki yıkıcı algoritmadan oluşuyormuş. Bu algoritmalarda, iyi bilinen birkaç alt sınır hesaplama yöntemi de uygulanmış. Bir dizi kıyaslama problemi örneğine dayanarak, önerilen algoritmaların performansını test etmek için hesaplamalı deneyler yapılmış. Hesaplanan sonuçlar, geliştirilen algoritmaların basit montaj hattı dengeleme kıyaslama problemi örneklerini çözmede etkili olduğu görülmüş.

1.4. PROBLEMİN TANIMI

Problem araç rotalama problemlerinin alt türü olan Gezgin Satıcı Problemi tarzında bir problemdir. Diğer bir deyişle başlangıç düğümü ile bitiş düğümü aynı olan kapalı bir tur oluşturacak şekilde bütün düğümlere sadece bir kere uğranan bir problem türüdür. Diğer Gezgin Satıcı Problemlerinden farklı olarak bu problemde her sefer için farklı zamanlarda tekrar tekrar uğranılan aynı şehir farklı düğümler olarak ele alınmaktadır. Bu nedenle aynı şehre tekrar tekrar uğransa dahi farklı düğümler olarak düşünüldüğünden problemi Gezgin Satıcı Problemi olarak düşnmek gerekir.

İKİNCİ BÖLÜM

ARAÇ ROTALAMA PROBLEMLERİ

2.1. ARAÇ ROTALAMA PROBLEMİ

Rota, bir aracın arka arkaya ziyaret edeceği şehirlerin (düğümlerin) peş peşe sıralanmasına denir (Kiremitci, 2014: 392). Araç rotalama problemi, bir merkezi depoda yerleşmiş bulunup her biri aynı veya farklı kapasitelerde olan araçlardan oluşan filoların, her bir aracın farklı bir yerleşim yerlerinde bulunan müşterilere hizmet vermek üzere, en az maliyetli araç rotalarının (mesafeyi veya süreyi enküçükleyecek şekilde) dağıtım yaparak ya da hizmet sunarak merkezi depoya geri dönmesi için rotaların tasarlanmasını içerir (Şahin, 2014: 337). Her müşteriye tam olarak bir kez hizmet verilir ve tüm müşteriler araç kapasiteleri aşmadan araçlara atanmalıdır. Güzergâhlar, düğümler arasındaki depo ve hareket mesafeleri ve aynı zamanda düğüm hizmet süreleri olarak adlandırılan aynı düğümde başlamalı ve bitmelidir. Bir aracın rota maliyeti, seyahat ettiği toplam mesafe ile ilgilidir ve amaç, herhangi bir kısıtlama olmaksızın minimum araç sayısını kullanarak tüm rotaların toplam maliyetini en aza indirmektir. Birincil amaç problemin toplam mesafesini en aza indirgemekle birlikte, toplam seyahat süresini en aza indirmek, hizmet kalitesini en üst düzeye çıkarmak, toplam yakıt tüketimini en aza indirmek veya bunların çoğunu azaltmak gibi literatürde var olan çeşitli hedefler vardır. Diğer taraftan, problemin fiziksel özellikleri sorunu, belirsiz müşteri talebi, bulanık seyahat süreleri veya zaman pencereleri, çoklu depo, heterojen filo tipi, iki veya üç boyutlu kamyon yükleme planları, yük bölme gibi ek kısıtlamalar ile çeşitlendirmektedir. Bu sorunun olası uzantılarından biri de, her müşterinin hizmetinin belirli bir zaman aralığı içinde başlaması gereken zaman pencerelerine sahip ARP'dir.

ARP'de her bir müşterinin hizmet vermesi gereken araç kapasitesi veya zaman aralığı, Kapalı Araç Rotalama Problemini (KARP) ve Zaman Pencereli Araç Rotalama

Problemini (ZPARP) gösterir. Gerçek dünya sorunları çoğunlukla kapasite ve zaman kısıtlamalarını kapsamaktadır. ZPARP problemlerini çözmek için çok çeşitli algoritmalar önerilmiştir. Çoğu zaman, bir müşteri teslimat için en erken ve en son zamana sahip bir zaman penceresi belirleyebilir, bu da ARP'ye zaman pencereleri (ARP-ZP) eklenmesine, neden olur. Diğer bir deyişle, bir araç bir müşteriye bir ARP-ZP'de belirtilen aralıkta bir müşteriye ulaşmalıdır. Bir aracın, bir müşteri tarafından ARP-ZP'lerde belirtilen en erken süreden önce ulaşması, boşta kalma süresine neden olacaktır. Diğer taraftan, bir aracın belirtilen son tarihten sonra bir müşteriye ulaşmasına izin verilmez (Şahin, 2014: 338).

Lenstra ve Rinnooy Kan (1981), araç yönlendirme probleminin karmaşıklığını analiz etmişlerdir ve pratik olarak tüm araç yönlendirme problemlerinin NP-Zor (klasik araç yönlendirme problemi) olduğu sonucuna varmışlardır. Çünkü bu problemler; makul bir sürede çözülemeyebilecek karmaşıklığı üsseel oalrak artan türden problemlerdir (Belfiore, 2013: 589).

Literatürde bu problem, statik ve dinamik strateji olmak üzere iki farklı bakış açısıyla incelenmiştir. Statik stratejide, herhangi bir gerçek talebin bilinmesinden önce, önceden tasarlanmış bir rotanın mevcut olduğu varsayılmaktadır. Ardından, araç depodan tam yük ile ayrılır ve boşalana kadar rotayı takip eder. Böyle bir durumda, bir başarısızlık olduğu söylenir. Bu nedenle, araç tekrar doldurma için depoya geri döner ve daha sonra hatanın meydana geldiği noktada rotayı tekrar başlatır. Toplam maliyet, birincil yolun maliyetinin yanı sıra, ekstra seyahatlerin maliyetini de içerir. Ancak dinamik stratejide, önceden tanımlanmış bir rota yoktur; bunun yerine, gerçekleştirilen bilgiye göre rota, çok aşamalı olarak kademeli bir şekilde inşa edilir. Örneğin, birincil karar ilk aşamada ziyaret edilmesi gereken bir müşteriyi belirler. Araç bu müşteriyi ziyaret ettiğinde, gerçek talep bilinir; daha sonra, gerçekleşen bilgilere bağlı olarak, bir sonraki karar, doğrudan ya da depodaki ikmalin ardından, bir sonraki ziyaret edilmesi gereken bir müşteriyi belirlemektir. Bu süreç tüm müşteriler ziyaret edilene kadar tekrarlanır. Son olarak araç depoya geri döner. Pillac ve arkadaşlarının (2013) belirttiği gibi, dinamik strateji statik stratejiye göre daha çok esneklik, maliyet düşüklüğü ve daha iyi çözüm gibi avantajlar sağlamasının yanında daha karmaşık bir model ve daha çok hesaplama gerektirmesi gibi dezavantajları da vardır. Dinamik stratejili Stokastik

Talepli Tekli ARP (SVRPD (Single VRP with Stokastic Demand)) ile ilgili mevcut literatür sınırlıdır ve şimdiye kadar Markov karar süreci MDP (Markov Decision Process) ve çok aşamalı SP modelleri olmak üzere iki tip formülasyon önerilmiştir. Ancak, çoğu işin odağının MDP modellerinde olduğu görülmektedir. Bunlar arasında Dror, Laporte ve Trudeau (1989), Dror (1993) ve Secomandi (1998) küçük örnekleri (10 müşteriye kadar) ve Secomandi (2000, 2001, 2003), Secomandi ve Margot (2009) ve Novoa ve Storer (2009), daha büyük örnekleri çözmek için rollout algoritması ve kısmi yeniden optimizasyon yaklaşımı gibi dinamik programlamaya dayalı çalışmalar vardır. MDP modelleri ile karşılaştırıldığında, çok aşamalı SP ile ilgili literatür birkaç eserle sınırlıdır. Hvattum, Løkketangen ve Laporte (2006), müşterilerin iki gruba ayrıldığı bir sorunu ele almıştır. Bunlar bilinen ve bilinmeyen müşterilerdir. Bilinen müşteriler gereksinimlerini önceden sipariş ederler ve bu nedenle belirli bir gün için başlangıç planını geliştirirken bilinirler. Ancak, bilinmeyen müşteriler gereksinimlerini gün içinde sipariş ederler ve çağrı süresi, yeri, zaman penceresi ve talebi belirsizdir. Yazarlar, zaman ufkunu, bilinmeyen bazı bilgilerin gerçekleştirildiği ve yeni müşterilere hizmet vermek üzere araç rotalarının değiştirildiği önceden belirlenmiş bir aralık sayısına böldü. İlk olarak, tüm bilinmeyen bilginin belirli bir zamanda açığa çıkarıldığı özel bir durum için iki aşamalı bir SP modeli önermişlerdir. Daha sonra, iki aşamalı modelin çok aşamalı olarak genişletilebileceğini iddia ettiler. Sorunu çözmek için, sezgisel riskten korunan bir örnek senaryo sundular. Bu problem daha sonra Hvattum, Løkketangen ve Laporte (2007) tarafından bilinen her müşterinin talebinin belirsiz olduğu ve ziyaret edildiğinde gerçek değerin açığa çıkarıldığı duruma genişletildi. Yazarlar bu problemi çok aşamalı bir SP modelinin çözümü şeklinde sunamamışlardır. Bildiğimiz kadarıyla, dinamik SVRPD için açıkça sağlanan çok aşamalı tek SP modeli, Dror (1993) tarafından ortaya konan analitik modeldir. Ancak, bu model uygulanamaz ve dolayısıyla, hiçbir hesaplama deneyi yapılmamıştır (Hoohsmand vd, 2016: 579).

Birçok farklı araç yönlendirme problemleri için farklı çözüm yöntemi literatürde bulunmaktadır. Bu çözümlerin dezavantajı, çoğunun yüksek uzmanlık gerektirmeleri ve esnek olmamaları ve yöntemleri değiştirilmiş problemlere uyarlamak için çok fazla çaba sarfetmeyi gerektirmesidir. Ek olarak, çoğu gerçek dünya problemi, literatürdeki idealize edilmiş problemlerden çok daha karmaşıktır ve zamanla değişmektedir (Claric vd, 2008: 15).

2.2. ARAÇ ROTALAMA PROBLEMLERİNİN TARİHÇESİ

Gezgin satıcı problemleri (GSP) tam olarak ilk kez ne zaman ortaya atıldığı bilinmemekle birlikte bu konudaki çalışmaların Euler tarafından ilk kez ortaya konulduğu düşünülmektedir. Leonhard Euler (1707-1783) GSP'ni Königsberg köprüleri ile ilgili bir problemin çözümü için tanımlamıştır. Königsberg köprüleri problemi, eski doğu Prusya topraklarında kalan Königsberg kentinin (bugünkü adı Kaliningrad) halkı tarafından, kentin içinden geçen Pregel nehri üzerindeki 7 köprüden geçiş yapmaya dair pazar eğlencesi olarak tasarlanmış bir oyundur. Euler'den sonra 1800'lü yıllarda İrlandalı matematikçi Sir William Hamilton ve İngiliz matematikçi Thomas Penyngton Kirkman GSP ile ilgili matematiksel problemler üzerinde çalışmışlardır. 1950 ve 1960'lı yıllarda GSP daha çok araştırmacı tarafından ele alınmıştır. Dantzig ve arkadaşları doğrusal programlama yardımı ile kesme düzlemi yöntemini geliştirmişler ve bu yöntem yardımı ile 49 şehirli bir GSP'ni çözebilmişlerdir (Yıldırım, 2014: 13).

2.3. ARAÇ ROTALAMA PROBLEMİ GENEL MATEMATİKSEL MODELİ

ARP ve GSP NP-Zor türü problemlerdir. Düğüm sayısı arttıkça olası rota sayısı exponansiyel olarak artmaktadır. Bu nedenle düğüm sayısı belirli bir değere ulaşınca problemin çözümü bilinen teknolojik imkânlarla hesaplanması imkânsız hale gelebilmektedir. Mesela n düğüm için (her düğümden bütün düğümlere gidilebilecek şekilde) mümkün rotaların sayısı (n-1)! / 2 dir. 20 düğüm için bu 60.822.550.204.416.000 tane rotaya karşılık gelmektedir (Demircioğlu, 2009: 2). Bu nedenle düğüm sayısının çok fazla olduğu araç rotalama problemlerinde genellikle sezgisel algoritmalar tercih edilebilmektedir.

Eldeki problem için matematiksel model kısaca aşağıdaki gibidir;

x, i noktasından j noktasına gidileceğini gösteren 0-1 tamsayı (sadece 0 veya 1 değerini alabilen) değişkendir.

t, i noktasından j noktasına varış zamanı ile kalkış zamanı arasındaki farkı gösteren bekleme ya da boşta kalma zamanı (aylak zaman) olmak üzere;

$Enk\text{üçü}k\ \sum_{i=0}^{n}\sum_{j=0}^{n} t_{ij}x_{ij} \quad , j \neq 0$ (Çam, 2018: 55).

2.4. ARAÇ ROTALAMA PROBLEMİ TEMEL BİLEŞENLERİ

1. Talep Yapısı: ARP'de talep statik ya da dinamik olabilir. Statik talep olma durumunda talep miktarı önceden bilinir. Dinamik olması durumunda ise taleplerin bir kısmı bilinmekte, bazı talepler ise araç talepleri karşılayacak araç dolaşırken belli olmaktadır.

2. Malzeme Tipi: ARP'de çok farklı malzemeler taşınabilmektedir. Bu maddelere örnek olarak tehlikeli maddeler, gazete dağıtımı, gıda, çöp sayılabilir. Bu maddeler basit paketler olarak isimlendirilir ve problem fazladan bir karmaşıklık getirmezler. Diğer taraftan öğrenci servisleri, etkinlik, güvenlik gibi hizmetleri sunan araçlar ARP açısından daha karmaşık bir yapıya sahiptir. Tehlikeli araçlar için rotaların belirlenmesinde ise coğrafi özellikler çok önemli olabilmektedir.

3. Dağıtım/Toplama noktaları: Çoğu ARP'de dağıtım noktaları depolar, malzemelerin dağıtıldığı veya hizmetin verildiği yerler ise müşteriler (düğümler) olarak geçmektedir.

4. Araç Filosu: Birçok araç rotalama probleminde araç kapasitelerinin bilindiği ve araçların özdeş (aynı kapasiteli) olduğu varsayılır. Filo heterojen bir yapıya sahipse bu durum hangi aracın hangi rotaya hizmet vereceğinin belirlenmesi gerekir. Bu durum da probleme ilave bir karmaşıklık katar. Bunlar dışında araçların hızı, yakıt tüketimi gibi özellikleri de sayabiliriz. Bu özellikler rotaların hesaplanmasında genellikle dikkate alınmazlar (Çalışkan, 2011: 3).

2.5. ARAÇ ROTALAMA PROBLEMLERİ ÇEŞİTLERİ

Araç rotalama problemleri genellikle sahip oldukları kısıtlara göre, farklı çeşitlere ayrılmıştır.

2.5.1. Karma Kapasiteli Araç Rotalama Problemi

ARP'de dağıtım yapacak olan araçların belirli bir kapasitesi olması durumudur. Karma kapasiteli araç rotalama probleminde her bir aracın kapasitesi farklı olabilmektedir (Demircioğlu, 2009: 54).

Sadece araç kapasitesi kısıtı dikkate alındığı ARP türlerine Kapasite Kısıtlı ARP (KARP) denir. Bu tür problemlerin çözümü için geliştirilen sezgisel ya da kesin çözüm algoritmalarına genellikle mesafe kısıtı da dikkate alınıp çözülür (Şahin, 2014: 338).

2.5.2. Çoklu Depoya Sahip Araç Rotalama Problemi

Dağıtım yapılacak olan depoların birden fazla olması durumudur. Eğer müşteriler depoların etrafında kümelenecek şekilde dağılmışsa, her bir depo için ayrı bir ARP düşünülebilir. Müşterilerle depolar iç içe ise bu durumda problem türü Çoklu Depoya Sahip Araç Rotalama Problemi türü olarak nitelendirilir (Demircioğlu, 2009: 54).

Birden çok şehir için (n adet) her biri farklı bir satıcıya atanmak üzere m adet ayrı tura bölünmektedir. Bu problem türü diğer problem türlerinden daha zordur. Problemi çözebilmek için hangi satıcılara hangi şehirlerin atanacağının belirlenmesi gerekir. Daha sonra da satıcıların, oluşturulan turlar üzerindeki şehirlerin (düğümlerin) en uygun şekilde sıralanması gerekir (Kuzu, 2014: 4).

Çoklu depoya sahip Araç Rotalama Problemlerinde hedef, toplam seyahat mesafesini minimize etmek ve her bir rota için araçların kapasitelerini aşmayacak şekilde müşterilere servis sunmaktır (Tezer, 2009: 37).

2.5.3. Bölünmüş Talebe Sahip Araç Rotalama Problemi

Bu problem türü aynı müşteriye (düğüme) birden çok aracın servis yapabilmesine olanak veren araç rotalama problemidir (Demircioğlu, 2009: 54).

2.5.4. Belirsiz Talebe Sahip Araç Rotalama Problemi

Belirsiz talebe sahip ARP talep miktarının belirsiz olduğu durumlarda olur. Araç müşteriye (düğüme) vardığı zaman talep miktarı belli olur (Demircioğlu, 2009: 54).

2.5.5. Geri Toplaması Olan Araç Rotalama Problemi

Geri toplamalı ya da eş zamanlı ARP ilk defa Min (1989) tarafından ortaya atılmıştır. Min'in geliştirdiği algoritma önce kümelemeyi sonra da rotalamayı temel alır (Çetin, 2011: 20).

Geri toplaması olan ARP, müşterilerin, ambalaj, depozito gibi, ürünlerin iade edilmesi durumu olan problem türleridir. Bu tür problemlerde müşterinin iade edeceği depozito gibi malzemeler de hesaba katılarak aracın kapasitesi ve rotası ayarlanır (Demircioğlu, 2009: 54).

Bu problem tipinde dağıtım işlemi ve toplama işlemi eş zamanlı olarak gerçekleştirilmektedir. Diğer bir deyişle müşteriye (düğüme) uğrandığında dağıtılacak malların bırakılması ve toplanılacak malların aynı zamanda yapılması kastedilmektedir. Örneğin dolu şişelerin bırakılıp boş şişelerin toplanması gibi (Atmaca, 2012: 24).

Araçların kapasiteleri, kira ücretleri vb. birbirinden farklı olduğu filo türlerinde, depolardan müşterilere (düğümlere) yapılacak olan dağıtım ve toplama işlemlerinin aynı araçlarla eş zamanlı olarak yapılması olarak da tanımlanabilir (Keçeci, 2015: 187).

2.5.6. Zaman Pencereli Araç Rotalama Problemi

Zaman pencereli ARP, her müşteri için teslimatın yapılacağı bir zaman kısıtının olduğu problem türleridir. Bu problem türünde araç her müşteriye belirli bir zaman içerisinde dağıtımı yapmak zorundadır (Demircioğlu, 2009: 54).

Zaman bağımlı Araç Rotalama Problemleri ilk kez Malandraki ve Daskin tarafından Gezgin Satıcı Problemi (GSP) üzerinde ele alınmıştır. Problemlerin çözümünde model olarak, Dal Kesme algoritması ile Açgözlü Sezgiseli önerilmiştir. Düğüm sayısı 10 ile 25 arasında değişen problemler ele alınıp çözülmüştür (Koç, 2014: 550).

2.5.7. Asimetrik Araç Rotalama Problemi

Dağıtım aracının merkezi depodan müşteriye veya müşteriden merkezi depoya olan rotaların aynı olmadığı problem türleridir. Bu tür problemlerde gidiş gelişler simetrik değildir (Demircioğlu, 2009: 54).

ÜÇÜNCÜ BÖLÜM

ARAÇ ROTALAMA PROBLEMLERİ ÇÖZÜM YÖNTEMLERİ

3.1. ARP İÇİN KLASİK SEZGİSEL YÖNTEMLER

ARP için geliştirilen sezgisel yöntemler; Klasik Sezgisel Yöntemler ve Metasezgisel Yöntemler olarak iki ana gruba ayrılmıştır. Klasik Sezgisel Yöntemler, turların oluşturulması ve geliştirilmesini içermektedir (Düzakın, 2009: 76).

3.1.1. Tasarruf Yöntemi

Tasarruf algoritması Clark ve Wright (1964) tarafından geliştirilmiş. Bu algoritmada her adımda tur kümesi değiştirilerek daha iyi bir çözüm seti elde etmeye çalışılmaktadır. Yöntemde ilk önce her araç için ayrı bir rota oluşturulmaktadır. Diğer bir deyişle her bir düğüme ayrı bir araç ile hizmet verilmektedir. Daha sonra elde edilebilecek en büyük tasarrufa göre iki rota birleştirilmektedir. Örneğin a ve b gibi iki düğüme hizmet eden iki araç yerine tek bir araç hizmet vermesi durumunda maliyet azalması sağlanıyorsa algoritmaya göre tasarruf sağlanmış olur (Kosif, 2012: 43).

Bu yöntemde başlangıçta bütün düğümlerin ortak bir depodan ziyaret edildiği varsayılır. Daha sonra en büyük tasarruf yapılabilen yerden başlanarak rotalar birleştirilir. Birleştirilen düğümler bir tek ortak büyük düğüm olarak kabul edilir (Eryavuz, 2001: 143).

3.1.2. Süpürme Yöntemi

Gillet ve Miller, (1974) tarafından geliştirilen Süpürme algoritmasının kökenleri, Wren (1971) ve Wren ve Holliday'ın (1972) çalışmalarına dayanır (Laporte, 1992: 355). Süpürme yönteminde, iki aşamalı bit yöntem kullanılır. Bu aşamalarda önce kümeleme

ardından rotalama yapılır. İlk aşamada müşteriler kümelenir. Müşteriler kümelenirken iki kritere göre kümeleme yapılır. İlk kriter için bütün müşteriler koordinat sistemi üzerinde belirtilir. Koordinat sisteminin orijininde (ortasında) dağıtım yapılacak depo yer alır. Diğer bir deyişle müşterilerin konumları koordinat sisteminde depo merkez temel alınarak belirtilir. İkinci kriterde araçların kapasiteleri müşteri ihtiyacına göre eşleştirilir. Böylece seçilen tüm müşterilerin toplam talebi o kümeye tahsis edilmiş olan araç kapasitesine eşit veya küçük olmak zorundadır. Daha sonra bu iki kriter birleştirilir ve her kümeye atanan araç kapasitesi, o kümede bulunan müşteri ihtiyaçları giderilene kadar kullanılır. Son adımda her küme için çözümler ayrı ayrı optimize edilerek genel çözüm hesaplanır (Caric, 2008: 18).

Süpürme algoritmasının uygulaması kolay olmasına rağmen hız ve performans bakımından tasarruf algoritmasından daha kötü bir performans sergiler. Bunun nedeni kümeleme yapılırken düğümler arasındaki mesafenin dikkate alınmamasındandır (Çalışkan, 2011: 10).

3.1.3. İki Aşamalı Yöntem

İki Aşamalı Yöntem Fisher ve Jaikumar (1981) tarafından geliştirilmiştir. İki Aşamalı Yönteminde ilk önce düğümler (müşteriler) arası mesafeler saptanır. Rota sayısı ile araç sayısı aynı olacağı varsayılır. Her araç için çekirdek bir düğüm belirlenir. Belirlenen bu düğümlerin birbirlerine olan uzaklıkları en çok olanlar seçilmelidir. Sonra gruplanan noktalara gerekli rota atamalar rotaya en yakın noktalara olacak şekilde yapılır. Son olarak da her küme (grup) Gezgin Satıcı Problemi türünde problem gibi çözülür. Kısaca her gruba bir araç atanmış olur ve bu araç Gezgin Satıcı Problemi türünde bir rotası olacak şekilde çözülür. Bu yöntemde kullanıcı tarafından $\lambda \geq 1$ ve $\mu \geq 1$ parametre değerleri belirlenip elde edilen iki çözümden en iyisi seçilir. Daha sonra daha iyi çözüm için bu parametreler değiştirilerek çözüm arayışına devam edilir. İki aşamalı olan yöntemde ilk adımda seri, ikinci adımda paralel rotalar oluşturulur. Bu yöntem, mesafe, zaman ve kapasite kısıtı bulunan Araç Rotalama Problemleri için geliştirilmiştir (Keskintürk, 2015: 93).

3.1.4. Petal Sezgisel Yöntem

Petal sezgisel yöntemi ilk olarak Foster ve Ryan (1976) tarafından önerilmiştir. Sonrasında 1993 yılında Ryan ve arkadaşları tarafından geliştirilmiştir. Geliştirilmiş petal sezgisel yöntemi ise Renaud ve arkadaşları tarafından 1996 yılında önerilmiştir. Bu sezgisel yöntemde, Petal yöntemi ile turlar oluşturulup kolon yenileme işlemi ile de optimal seçim yapılmaktadır. Petal Sezgisel Yöntemi ile hızlı bir şekilde optimale yakın sonuçlar bulunabilmektedir (Demircioğlu, 2009: 67).

3.1.5. En Yakın Komşu

Bellmore and Nemhauser (1966) tarafından geliştirilen En Yakın Komşu sezgisel algoritması (NNH)(The Nearest Neighbour Heuristic), son eklenen müşterinin en yakın komşusunu rotaya yerleştirme fikri ile yapıcı bir yöntemdir. En Yakın Komşu prosedürü (NNP)(Nearest Neighbour Procedure), yalnızca son ziyaret edilen düğümden ağdaki en yakın düğüm noktasına seyahat etme maliyetine veya mesafesine dayalı bir tur oluşturur. Güzergâh üzerindeki ilk eklenen müşteri rastgele veya depodan en uzak mesafe müşterisi gibi bazı keyfi kriterler ile seçilebilir. Bu kök rotadan, en yakın komşu kriteri ile en son eklenen müşteriden diğer bütün müşteriler araç kapasitesi tükenene kadar eklenir. Aracın teslimatı, her müşterinin teslimat veya teslim alma talebinin olduğu problem tanımına göre tüketilir. Bu yöntem, Gezgin Satıcı Probleminin (GSP) sezgisel yaklaşımından türetilmiştir (Caric, 2008: 18).

3.1.6. Fisher ve Jaikumar Algoritması

Fisher ve Jaikumar (1978, 1981) kapasite kısıtlamalarına, zaman pencerelerine ve durma sürelerine sahip olmayan ARP'ler için üç endeksli bir araç akış formülasyonu geliştirdiler. Bu gibi formülasyonlar, bir aracın bir yay veya kenar üzerinde geçmesini temsil etmek için i ve j değişkenleri kullanır. Üç endeksli formülasyonlarda, x_{ij} ve k değişkenleri (i, j) aracın k tarafından taşınıp taşınmadığını gösterir. İki endeksli formülasyonlarda, x_{ij} değişkenleri hangi aracın kullanıldığını belirtmez. Fisher ve Jaikumar da bu formülasyona dayanan bir algoritma geliştirdiler. Her ne kadar bu

algoritma problemi sadece sezgisel bir çözüm sağlamak için kullanılıyor gibi görünse de, tamamlanma aşamasına kadar devam ederse, sonlu sayıda adımda optimal bir çözümü garanti eder (Laporte, 1992: 352).

İki aşamalı bir yöntem olan Fisher ve Jaikumar (1981)'ın çalışması Genelleştirilmiş Atama Metodu olarak da bilinir. Gruplamanın yapıldığı ilk aşamada ilk önce müşteriler arası mesafeler hesaplanır. Mevcut araç sayısı kadar rota oluşturacağımızı varsayarak, her aracın rotası için bir çekirdek (seed) müşteri belirlenir. Belirlenen müşteriler, birbirlerine olan mesafeleri en uzak olanlardan seçilmelidir. Daha sonra ise noktaların oluşturulan bu rotalara başka bir deyişle nokta ve depo arasındaki doğrulara uzaklıkları hesaplanır ve kapasite limiti dikkate alınarak en yakın noktalar rotalara atanır. İkinci aşamada ise belirlenen gruplar GSP sezgiselleri ile çözülür (Keskintürk, 2015: 93).

3.2. METASEZGİSEL YÖNTEMLER

Metasezgisel Yöntemler Sezgisel Yöntemlere göre daha çabuk optimal sonuca yakın sonuçlar vermektedir.

3.2.1. Tavlama Benzetim Yöntemi

Tavlama Benzetim algoritması, ilk olarak 1983 yılında Kirkpatrick, Gelatt ve Vecchi tarafından sunulmuş olup, optimizasyon problemlerinin çözümü için geliştirilmiş bir yerel arama algoritmasıdır. Tavlama Benzetim (TB) algoritması, adını erimiş metalin soğutulması işlemi olan, tavlama işleminden almaktadır. Bu işlemde metalik yapıdaki kusurları azaltmak için materyal ısıtılır, daha büyük bir kristal boyuta ve minimum enerji ile katı kristal duruma yavaşça soğutulur. Tavlama işlemi, sıcaklığın ve soğuma katsayısının dikkatlice kontrolünü gerektirir. Tavlama işlemi sonucunda oluşan kristalleşme, metalin mekanik özelliklerini iyileştiren moleküler yapısındaki değişikliklerle oluşmaktadır Tavlama işlemindeki ısının davranışı, optimizasyondaki kontrol parametresiyle aynı gibi görülür. Isının, daha iyi sonuçlara doğru algoritmaya rehberlik eden bir rolü vardır. Bu durum ancak kontrollü bir tutum içinde, ısının kademeli olarak düşürülmesiyle yapılabilir. Eğer ısı aniden düşürülürse, algoritma lokal

minimum ile durur. TB algoritması; birçok değişkene sahip fonksiyonların maksimum veya minimum değerlerinin bulunması için, özellikle de birçok yerel minimuma sahip doğrusal olmayan fonksiyonların minimum değerlerinin bulunması için tasarlanmıştır (Kuzu, 2014: 6).

3.2.2. Yapay Sinir Ağları

İlk olarak 19.yüzyılda insan beyninin nörofiziksel yapısından esinlenerek temelleri atılan Yapay Sinir Ağları (YSA), beynin nöronlardan oluşan yapısını ve öğrenme yöntemlerini inceler. Bu konulardaki ilk modern çalışmalar McCuloch ve W.Pitts ile başlar (Tektaş, 2006: 4).

YSA, beynin bir işlevi yerine getirme yöntemini modellemek için tasarlanan bir sistem olarak tanımlanabilir. YSA, yapay sinir hücrelerinin birbirleri ile çeşitli şekillerde bağlanmasından oluşur ve genellikle katmanlar halinde düzenlenir. Donanım olarak elektronik devrelerle veya bilgisayarlarda yazılım olarak gerçekleşebilir. Beynin bilgi işleme yöntemine uygun olarak YSA, bir öğrenme sürecinden sonra bilgiyi saklama ve genelleme yeteneğine sahip paralel dağılmış bir işlemcidir (Ataseven, 2013: 102).

3.2.3. Tabu Arama Yöntemi

Tabu Arama yöntemi ARP çözümlerinde başarılı sonuçlar vermiştir. Tabu Arama yöntemi ile literatürde yer alan problem çözümlerinin dışında günlük hayatta da çokça kullanım alanı bulmuştur (Düzakın, 2009: 82).

İlk kez Glover (1989) tarafından geliştirilen Tabu Arama yöntemi, kombinasyonel optimizasyon problemlerini çözmek için geliştirilmiş bir sezgisel tekniktir ve başka metotlarla birlikte kullanılarak, bu metotları yerel optimum tuzağına düşmekten koruyan uyarlanabilir bir yaklaşımdır. Tabu Aramanın bugünkü modern şeklini Glover vd. (1997) ortaya koymuştur (Eren, 2004: 21).

Glover (1990), daha fazla iyileştirme ve yöntemin daha sofistike yönlerini sunmuştur. Bazı temel bileşenlerin çoğu Tabu Aramanın (TA) formülasyonuna dahil edilmesi gerekir ve bunlar TABB'ne dahil edilir:

(a) uygun bir başlangıç çözeltisi,

(b) ilgili bir mahalle yapısı,

(c) uygun bir tabu listesi (TL),

(d) uygun bir değerlendirme fonksiyonu,

(e) durdurma kriteri (Woodcock, 2010: 567).

TA kombinasyonel optimizasyon problemlerinde iyi sonuçlar vermektedir ve Araç Rotalama Problemlerinin çözümünde Genetik Algoritma, Benzetilmiş Tavlama, Karınca Kolonisi, Yapay Sinir Ağları gibi yöntemlerden daha iyi sonuçlar vermektedir (Bozyer, 2014: 32).

3.2.4. Karınca Kolonisi Optimizasyonu

Temel ilkeleri ilk kez Marco Dorigo (1999) tarafından geliştirilmiş bir yöntem olan Karınca Kolonisi Optimizasyonu (Demircioğlu, 2009: 72), kombinasyonel çözüm bulmak için yapay karıncalar kullanan bir metasezgisel tekniktir. Dorigo, Karınca Kolonisi algoritmasını ilk kez Gezgin Satıcı Problemleri (GSP) üzerinde denemiş ve iyi sonuçlar almıştır. Dorigodan sonra diğer araştırmacılar da yöntemi kullanmaya başlamışlardır ve günümüzde ARP çözümlerinde yaygın olarak kullanılan bir sezgisel yöntem haline gelmiştir (Dikmen, 2014: 10).

Gıda arayışında olan gerçek karıncalar üzerinde yapılan gözlemler, karınca kolonilerinin, kombinasyonel optimizasyon problemlerini çözme davranışını taklit etmek için ilham kaynağı olmuştur (Bullnheimer, 1999: 286). Doğada, bireysel bir karınca yiyecek için iletişim kuramaz veya etkili bir şekilde avlanamaz, ancak bir grup olarak karıncalar karmaşık problemleri çözme becerisine sahiptirler ve onların kolonileri için yiyecek bulmak ve toplamak için feromon adı verilen bir kimyasal madde kullanarak iletişim kurarlar. Her karınca rasgele bir şekilde hareket eder, ancak bir karınca feromon iziyle karşılaştığında, onu takip edip etmeyeceğine karar vermelidir.

Eğer iz sürerse, karıncaya ait kendi feromonu varolan patikayı güçlendirir ve feromondaki artış, yolu seçen bir sonraki karıncanın da bu yolu seçme olasılığını artırır. Bu nedenle, bir yol üzerinde seyahat eden daha fazla karınca, sonraki karıncalar için yolu daha çekici hale getirir. Ek olarak, bir yiyecek kaynağına kısa bir yol kullanan bir karınca yuvaya daha erken dönecek ve bu nedenle diğer karıncaların dönüşünden önce yolunu iki kez işaretleyecektir. Bu, yuvadan çıkan bir sonraki karınca için seçim olasılığını doğrudan etkiler. Zamanla, daha fazla karınca daha kısa rotayı tamamlayabildiğinden, feromon daha kısa yollarda daha hızlı birikmekte ve daha uzun yollar daha az güçlendirilmektedir. Feromonun buharlaşması ayrıca daha az tercih edilen rotaları tespit etmeyi zorlaştırır ve kullanımlarını daha da azaltır. Bununla birlikte, bireysel karıncaların devam eden rasgele seçim yolu, koloninin alternatif rotaları keşfetmesine yardımcı olur ve bir rotayı kesen engeller etrafında başarılı bir navigasyon sağlar. Karıncalar tarafından iz seçimi, sözde rasgele orantılı bir süreçtir ve Karınca Kolonisi Optimizasyonu algoritmasının önemi bir unsurudur (Bell, 2004: 42). Dorigo, karınca kolonilerinin davranışlarının matematiksel modelleri üzerine dayandırdığı Karınca Kolonisi Algoritmalarını ilk kez Gezgin Satıcı problemi üzerinde kullanmış ve olumlu sonuçlar elde etmiştir (Düzakın, 2009: 83).

3.2.5. Genetik Algoritma

Genetik Algoritma (GA) ilk olarak John Holland (1975) tarafından doğal ve yapay adaptif sistemlerin çalışmasından esinlenerek geliştirilmiş ve literatüre kazandırılmıştır. Holland bu çalışmalarına doğal sistemlerin işleyişini araştırıp, doğal sistemlerin çalışma prensibine dayanan yapay sistemler geliştirmeyi hedefleyerek başlamıştır (Emel, 2005: 6).

GA'da ilk önce kodlama biçiminin nasıl olacağına karar verilir. Genellikle ikili kodlama, permutasyon kodlama ya da gerçek değerli kodlama kullanılır (Keskintürk, 2016: 60).

GA yöntemi ile kısa sürede global çözüme ulaşabilme gibi avantajların yanında yerel çözümlerin kalitesindeki artışın yavaş olması gibi dezavantajlar da vardır (Çolak, 2010: 426).

GA her kuşakta mutasyon ve çaprazlama yöntemlerini kullanarak yeni popülasyonlar oluştururlar. Birkaç kuşak sonra problemin çözümüne ilişkin daha uygun değerlere sahip yeni kuşaklar ortaya çıkar (Pakkan, 2010: 80).

GA, organizmalardaki kalıtım ve evrim süreci boyunca seçim, geçiş ve mutasyon prensiplerini taklit ederek, optimal çözüm için uyarlanabilir arama prosedürünün gerçekleştirilebilmesini sağlar. GA karmaşık optimizasyon problemlerini çözmek için genel bir çerçeve sunar. Genetik Algoritmaların, fonksiyon optimizasyonu, çizelgeleme, mekanik öğrenme, tasarım, hücresel üretim gibi alanlarda başarılı uygulamaları bulunmaktadır (Emel, 2002: 130).

Genetik algoritmalarda crossover ve mutasyon kullanılır. Her bir çözümün kalitesi bir fitness değeri ile gösterilir. Bu değer, popülasyondan çoğaltılacak ve çözeltiler popülasyondan hariç tutulduğunda bir çözüm seçmek için kullanılır. Nüfusun ortalama kalitesi, bir taraftan yeni ve daha iyi çözümler üretilirken diğer taraftan da daha kötü çözümler kaldırıldıkça, kademeli olarak iyileşir (Razali, 2015: 1925).

3.2.6. Parçacık Sürü Optimizasyonu

Parçacık Sürü Optimizasyonu (PSO), Kennedy ve Eberhart (1995) tarafından doğada balık ve kuş gibi büyük gruplar halinde yaşayan hayvanların sürü davranışlarından esinlenilerek geliştirilmiş bir sürü zekâsı algoritmasıdır (Seyfi, 2018: 12).

Kuşlar besin ararken hedefe en yakın kuşun peşinden giderler. Benzer şekilde bu algoritmada olası muhtemel çözümler parçacık olarak adlandırılır ve parçacıklar aralarındaki bilgi paylaşımı sayesinde o anki en iyi parçacığı izleyerek çözüm uzayında dolaşırlar. Uygulamaya herhangi bir çözümle başlanır ve değişimlerle en iyi çözüm bulunmaya çalışılır. Algoritmada kullanılan parametre sayısının az olması nedeniyle parçacık sürü optimizasyonunun uygulanması kolaydır (Keskintürk, 2015: 96).

3.3. KESİN ÇÖZÜM YÖNTEMLERİ

Kesin Çözüm Yöntemleri adından da anlaşılacağı üzere, söz konusu problemin çözümü var ise kesin olarak optimal sonucu bulabilen yöntemlerdir.

3.3.1. Minimum Karar Ağacı Yöntemi

Bu yöntemde K-Ağaç, n + k kenar setinin, G grafiğini n + 1 nokta ile kapsayacak şekilde tanımlanır. ARP'de kapasite kısıtı ve her bir noktanın bir kez ziyaret edilmesi kısıtı sağlanarak, minimum K-Ağaç yönteminde maliyeti bulacak şekilde model kurulur (Demircioğlu, 2009: 56).

3.3.2. Dinamik Yaklaşım

Dinamik Programlama, Richard Ernest Bellman tarafından 1950 yılında geliştirmiş ve isimlendirilmiştir. Bu isim çok aşamalı karar süreçlerini ifade etmek adına da kullanılmaktadır (Patır, 2009: 64). Dinamik Programlamada problem birbirinden bağımsız alt problemlere ayrılır. Bu alt problemlerin çözümleri saklanır ve ihtiyaç olduğu zaman bu çözümler kullanılarak Araç Rotalama Probleminde çözüme ulaşılır (Keskintürk, 2015: 88).

3.3.3. Çok Yüzlü Yaklaşım

GSP çözmedeki çok yüzlü (polyhedral) yaklaşımın başarısı, ARP uygulanmasına ilham kaynağı olmuştur. Bu yöntem ile literatürde yer alan ve şu ana kadarki çözülebilen en büyük ARP olan 134 müşterilik problem çözülmüştür (Demircioğlu, 2009: 56).

3.3.4. Dal Kesme Yöntemi

Dal Kesme Yöntemi, Dal ve Sınır ile kesme düzlemi yöntemlerinin bir birleşimidir. Araç rotalama problemlerinin çözümü için ilk önce problemin doğrusal Programlama ile çözümü yapılır. Bu aşamada amaç fonksiyonu ve kısıtlar yazılarak

model oluşturulur. Modelin çözümünde oluşabilecek olan alt turlar sıfıra eşitlenerek problem dallara ayrılır. Daha sonraki dallarda ise araç sayısı gibi kısıtları sağlayabilmek için modele alt tur engelleme kısıtlayıcısı eklenerek optimal sonuca ulaşılmaya çalışılır (Keskintürk, 2015: 88).

3.3.5. Dal ve Sınır Yöntemi

İlk olarak 1950 li yıllarda A.H.Land ve A.G. Doing tarafından geliştirilen Dal ve Sınır Yöntemi (D & S) matematiksel programlama problemlerinin en iyi çözümlerinin bulunması ile ilgili pek çok yöneylem araştırması probleminde uygulanmaktadır. Doğrusal Programlama problemlerinin tamsayı karar değişkenleri ile çözülmesi bağlamında Land ve Doig (1960) Dal ve Sınır metodunu geliştirilmiştir. Tamsayılı Programlama modelini standart Doğrusal Programlama teknikleri (örneğin, simpleks metodu) kullanarak çözmeye çalışırken, tamsayı değerlerini almak için gerekli olan değişkenlerden bir veya daha fazlası, gerçek Doğrusal Programlama çözümünde kesirli olabilir. Dallanma Land ve Doig'ın algoritma dallarındaki süreçleri bu kesirli değişkenlerden alır. Bir dal, kesirli değişkenin, en büyük tamsayıya eşit veya daha küçük bir değere sahip olmasını gerektirir, diğer dal ise, değişkenin, daha büyük olan en küçük tam sayıdan büyük veya ona eşit bir değer almasını gerektirir. Algoritma, dalların düğümleri için doğrusal programlama alt problemlerinin çözümü ile devam eder. Nihai amaç, Doğrusal Programlama problemine en uygun çözümü üretmektir (Brusco, 2005: 4).

Dal ve Sınır yöntemi bir sorunla ilgili türlü aşamaları sistemli bir şekilde analiz ederek en iyi çözümü araştırır. Dal ve Sınır (D & S) algoritmaları, operasyonel araştırma, kombinasyonel optimizasyon ve yapay zeka konularında çok çeşitli problemleri çözmek için yaygın olarak kullanılır. Ancak, hangi algoritmaların D & S kategorisine girdiğine dair birçok fikir ayrılığı vardır. "Dal ve Sınır" terimi, genel, Karma-Tamsayı, Doğrusal Programlama problemlerini çözmek için bir yönteme başvurmak için ilk olarak operasyonel araştırma alanında kullanılmıştır. Tartışmalı olarak, son yıllarda, yapay zekâ alanında kullanılan bir dizi AND / OR grafik arama prosedürünün aslında D & S algoritmaları olarak görülebileceği iddia edilmiştir (McKeown, 1991: 1).

Başlangıç adımda olanaklı çözümlerin toplam kümesi daha küçük alt kümelere ayrılır. Aynı anda her bir alt küme için üst veya alt sınır değerleri belirlenir. Daha sonra bu değerlere bağlı olarak bazı alt kümelerin çözümden atılması işlemi gerçekleştirilir (Sezen, 2017: 120).

D & S'de arama alanı, kökü asıl sorun olan bir ağaç olarak temsil edilir, iç düğümler kısmen alt problemleri çözer ve yapraklar potansiyel çözümlerdir. D & S, şimdiye kadar bulunan en iyi çözümün (üst sınır) giderek geliştirildiği birkaç yinelemeyle ilerler: Daha iyi bir çözüme götürecek ve karşılık gelen alt dalları kesecek olan alt problemleri ortadan kaldırmak için bir sınırlama mekanizması kullanılır. Bu, keşfedilen arama alanının boyutunu azaltır, ancak pratikte hala zaman alıcı olabilir ve örneğin paralel hesaplama kullanarak hızlanma gerektirir (Borisenko, 2017: 640). Dal ve Sınır algoritmasında, Gezgin Satıcı Problemi, alt tur (depoda başlayıp bitmeyen turlar) engelleyici kısıtları yok edilerek atama problemi haline dönüştürülür ve Macar Yöntemi ile çözüme başlanır. Satır ve sütün eleme yöntemiyle rotalar belirlenmeye çalışılır. Alt tur oluşursa; en kısa döngüyü engelleyecek kısıtlar ile dallandırılır. İstenilmeyen rotalara atama yapılmaması için maliyet matrisinde ceza katsayısı olarak büyük M sayısı atandıktan sonra matris tekrar baştan çözülerek tüm dallar için aday çözümler belirlenir. Tüm dallar için aynı iterasyonlar tekrarlandıktan sonra en iyi çözüme karar verilir (Keskintürk, 2015: 88).

Dallanma rutini, eğer bir alt uzay ümit vaat ediyorsa ve atılamazsa uygulanır, dolayısıyla bu alt alan, sonraki iterasyonlarda keşfedilecek diğer alt alanlara bölünür (Kadri, 2016: 44).

Dallanma var olan problemi iki ayrı alt probleme ayırma işlemidir. Böylece asıl problem yerine iki ayrı alt problem ortaya çıkmış olur. Oluşturulan bu alt problemler asıl problemin parçalanmış alt problemleri olmaktadır. Bu parçaları asıl problemin kısmen çözülmüş halleri şeklinde de düşünebiliriz. Problemi Dal ve Sınır yöntemi ile çözebilmek için her aşamada problem alt problemlere (dallara) ayrılır. Ayrılan bu dallar diğer aşamada tekrar alt problemlere (dallara) ayrılır. Dal ve Sınır yöntemi ile problemin çözümü için dallanma iki farklı şekilde olmaktadır. Bunlar permütasyonel dallanma ve kombinasyonel dallanma olarak isimlendirilir (Sezen, 2017: 121).

Sınırlama fonksiyonu, D & S algoritmasının verimliliğinin ana bileşenidir ve düğüm seçimi ve dallanması için uygulanan stratejilerle dengelenemez. En iyi durumda, belirli bir alt problem için bir sınırlama fonksiyonunun değeri, optimal çözümün değerine eşit olmalıdır, ancak bu amaca ulaşmak çoğu durumda zordur (Kadri, 2016: 44).

Dallarda sayımlamayı azaltmak ve optimal çözümü daha az sayımlama ile bulmak için sınırlama yapılır. Sınırlama yaparken sınırlama işlemi yapılacak alt problemin en iyi çözümü için önceden bir değer ya da aralık belirlenir. Bu aralığa ya da değere göre sınırlama yapılır. Yaratılan her bir alt problemde sınırlama yaparken genellikle alt problemin alabileceği en küçük ve en büyük değerler belirlenir. Belirlenen bu değerlerden en küçük olanı alt sınır, en büyük olanı da üst sınır olarak isimlendirilir. Alt ve üst sınırlar genellikle hesaplanarak bulunur. Sınırlama ile problemin alt çözümlerinden bazıları çözüm dışında bırakılarak sayımlama azaltılır. Bunun sonucunda bu probleme ait sınırlanmış alt dallar için gereksiz hesaplamalardan kaçınılmış olur. Başka bir deyişle sınırlama ile bir alt dalın dallandırılmasının verimli olup olmayacağı ortaya çıkmış olur. Bir alt dala konulan sınır ile problem çözümünün devamında bu dal üzerinden en iyi çözüme gidilip gidilemeyeceğine karar verilebilir (Sezen, 2017: 121).

Dal ve Sınır yönteminde optimal çözüm bulmak için iki farlı yol vardır. Bunlardan ilki Sıçramalı İzleme Yaklaşımı, diğeri de Geri İzleme Yaklaşımıdır.

3.3.5.1. Sıçramalı İzleme Yaklaşımı

Bu yöntemde başlangıçta çözüm için alt ve üst sınırlar belirlenebilir. Daha sonra çözüm kümelerinden birisi ile işe başlanır. Çözüm kümelerinin çözüm için uygun olup olmadıkları test edilir. Kümelerden hiçbirisi uygun değilse problemin uygun bir çözümü yoktur. En küçükleme tipi bir problemde uygun çözümler içerisinden en iyi çözüm problemin çözüm değeri olarak ele alınır. Sonra alternatif çözümler içerisinden bütün çözümler bu değerle karşılaştırılır. Daha sonra daha iyi bir çözüme ulaşmak için bütün değerler eldeki bu en iyi çözüm değerinin üst sınırı ile karşılaştırılır. Daha küçük değere sahip bir çözüm var ise yeni bulunan çözüm en iyi çözüm olarak seçilir. Daha büyük değere sahip diğer çözümler ise işlem yarıda kesilerek problem çözümünden atılır.

Çünkü daha büyük değere sahip olduğundan, dallanmalara devam edildiğinde bulunan değer daha da büyüyecek ve en uygun çözümden daha da uzaklaşılacaktır. Geriye kalan çözümlerden amaç fonksiyonunun alt sınır değeri belirlenir. Belirlenen bu alt sınır değeri geriye kalan tüm çözümler için alt sınır değeri olarak alınır. Bu alt sınır değerinin uygun bir alt sınır değeri olması zorunlu değildir. Yeni çözümler araştırılırken dallandırma işleminde bu alt sınır değeri kullanılır. Sayımlama alt sınır değeri ile üst sınır değeri eşitlenene kadar devama eder. Eşitlendiğinde en iyi çözüm bulunmuş olur (Sezen, 2017: 130).

3.3.5.2. Geri İzleme Yaklaşımı

Geri İzleme (Backtracking) Yaklaşımının temelinde yatan düşünce; ilk önce hızlı bir şekilde deneme çözümü bulunup daha sonra geriye doğru daha iyi çözüm olup olmadığının araştırılmasıdır. Diğer bir deyişle ağaç yapısı üzerinde bir deneme çözümü bulunup daha sonra bu çözümün adım adım iyileştirilmesidir. Bir problemde sayımlamayı azaltmanın yollarından biri deneme çözümü kullanmaktan geçer. Bir deneme çözümü bulunduktan sonra, en iyi çözüm olarak; bulunan deneme çözümü kabul edilir. Deneme çözümünün bulunmasında herhangi bir yöntem kullanılabilir. Deneme çözümü bulunmazsa Dal ve Sınır yöntemi ile çok fazla sayımlama yapmak gerekir. Sayımlamayı şu şekilde azaltabiliriz: Deneme çözümü bulduktan sonra alternatif çözümler için alt dallar araştırılırken daha kötü bir alt dala rastlandığında bu dalın sonraki aşamalarına (alt dallar) bakılmaksızın yarıda kesilerek çözümden atılır. Böylece daha az alt dal araştırılmış olup sayımlama azaltılır. Alt dallar araştırılırken ağacın en alt sınırına kadar gelindi ise ve bulunan çözüm deneme çözümünden daha iyi bir çözüm ise mevcut deneme çözümü yerine yeni bulunan çözüm geçer. Bu işlemler ağacın başlangıcından geriye doğru gidilerek tekrar edilir. Araştırılacak hiç alt dal kalmayıncaya kadar işlemlere devam edilir. Araştırılacak alt dal kalmayınca o ana kadar bulunan en son deneme çözümü en iyi çözüm olmuş olur (Sezen, 2017: 140).

Geri İzleme Yaklaşımı ile enküçükleme probleminin çözüm adımları şu şekildedir:

1. Adım: Asıl problem için alt sınır değerlerini belirle ve problemi alt problemlere (dallara) ayır.

2. Adım: Yeni yaratılan problemler için alt sınır değerlerini belirle.

3. Adım: En küçük sınır değerine sahip düğümü seçip alt problemlere parçalayıp dallandır.

4. Adım: Dallandırılacak düğüm kaldı ise 2.Adım'a git.

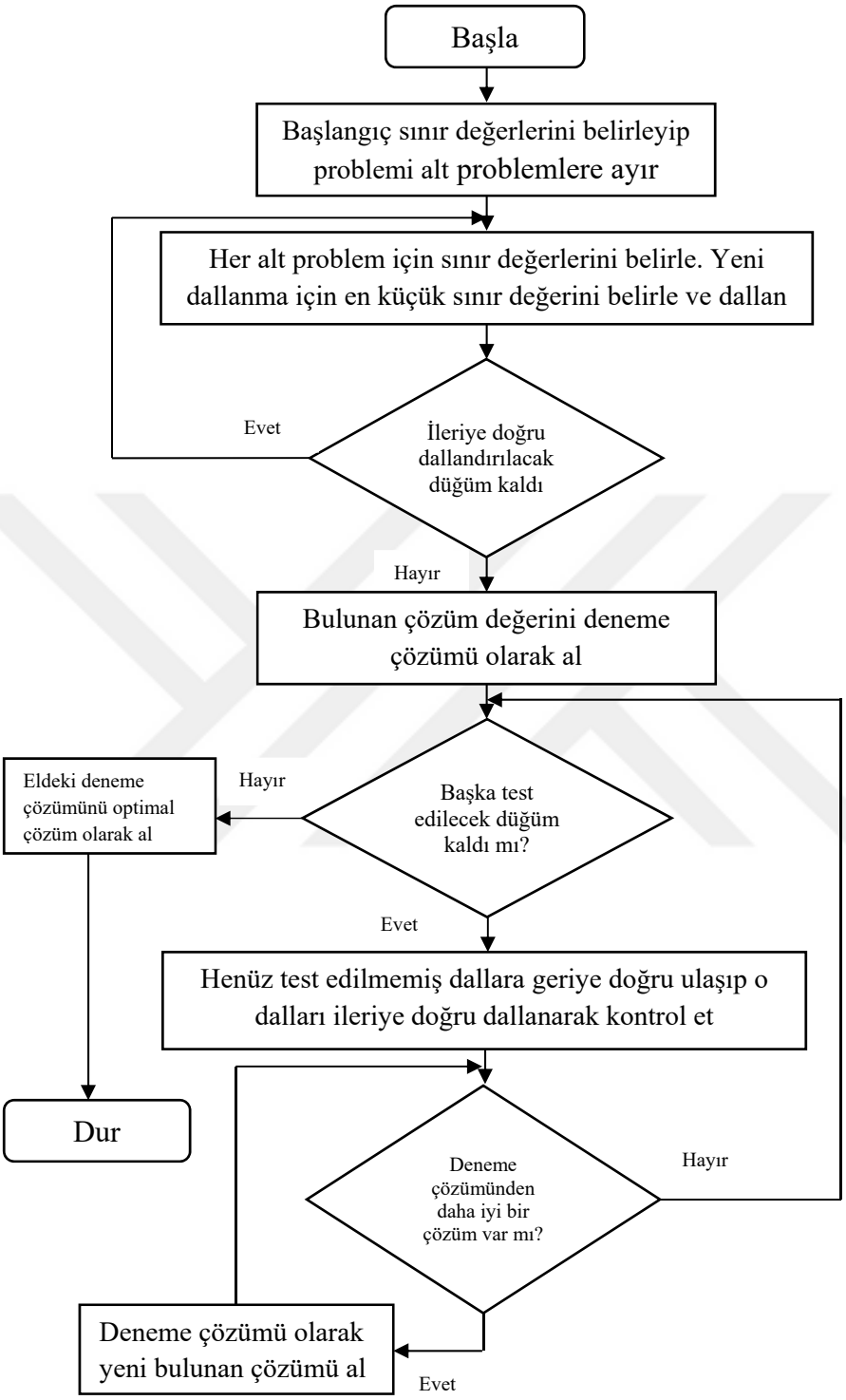
5. Adım: Dallandırma işleminin sonuna gelindiğinde bulunan çözüm değerini deneme çözümü olarak al.

6. Adım: Elde edilen deneme çözümü ile son aşamadan geriye doğru karşılaştırmalar yapılarak devam edilir. Yapılan karşılaştırmalarda sınır değeri daha büyük olan düğümler elimine edilir. Daha sonraki aşamalarda bu düğümler üzerinde işlem yapılmaz.

7. Adım: Elimine edilemeyen düğüm değeri deneme çözümü değerinden küçük olduğu sürece en son aşamaya gelene kadar (tam bir çözüm bulununcaya kadar) dallandırmaya devam edilir. Yapılan karşılaştırmalarda daha küçük değerler bulundu ise deneme çözümü yerine daha küçük değerli yeni çözüm geçer.

8. Adım: Dallanma işlemi elimine edilmeyen tüm düğümlere uygulandıktan sonra optimal çözüm bulunmuş olur (Sezen, 2017: 141).

Geri İzleme Yaklaşımı ile enküçükleme probleminin çözüm adımları için akış diyagramı aşağıda yer alan şekil Şekil 3.1'deki gibidir:



Şekil 3.1 Geri İzleme Yaklaşımı Akış Diyagramı

DÖRDÜNCÜ BÖLÜM

UYGULAMA

4.1. GİRİŞ

Çalışmada yer alan problem araçların boşta kalma (aylak kalma) sürelerinin enküçüklemeye çalışıldığı, karayolunda yolcu taşıması yapan bir otobüs şirketine ilişkindir. Otobüs şirketinde kullanılan belirli bir sefer topluluğu için aylak zaman toplamını enküçükleyen rota belirlenmesine ilişkin optimal çözüm araştırılacaktır.

Uygulamada rota dairesel olarak hesaplanmaya çalışılmıştır. Diğer bir deyişle rota başladığı şehirde son bulacak şekilde hesaplanmaya çalışılmıştır. Çözümde alt turlar oluşabilmektedir. Bu durumda alternatif rotalar taranarak çözüme sadece dairesel olan rotalar dahil edilmiştir. Aynı şekilde çözümü imkânsız ya da çıkmaz rotalama durumlarıyla karşılaşılabilmektedir. Yine bu durumda da alternatiflere bakılıp çözümsüz rotalar çözümden çıkarılmış, dikkate alınmamıştır.

Uygulamada aynı şehre birden çok kez uğranabilmektedir. Problemde tanımlanan şekilde her şehre tanımlanan sefer sayısı kadar uğranılmıştır. Bu nedenle var olan problemdeki düğüm sayısı şehir sayısı kadar olmayıp sefer sayısı kadardır. Problemde bir şehre farklı zamanlarda tanımlanan her bir sefer için tekrar uğranacağından o şehir-zaman bileşimi ayrı bir düğüm olarak alınmalıdır.

4.2. VERİLER

Uygulamada aşağıdaki iller ve seferler için aylak zamanı en küçük kılacak şekilde optimal çözüm araştırılmıştır. Tablo 4.1'de görüldüğü gibi 24 sefer için sefer numarası, seferin başlayacağı şehir, seferin sona ereceği şehir, kalkış zamanı ve yolculuk süresi gibi bilgiler eldeki problem için tanımlanmıştır.

Sefer No.	Başlangıç Şehri	Bitiş Şehri	Kalkış Saati	Yol. Süresi
1	Datça	Ankara	19:00	13:50
2	Ankara	Datça	21:30	13:50
3	Datça	İzmir	10:00	08:05
4	İzmir	Marmaris	19:00	06:15
5	Marmaris	İzmir	08:30	06:05
6	İzmir	Fethiye	16:00	08:00
7	Fethiye	İzmir	05:00	08:05
8	İzmir	Fethiye	14:00	08:00
9	Fethiye	İzmir	23:30	08:15
10	İzmir	İstanbul	09:00	11:50
11	İstanbul	İzmir	21:30	11:20
12	İzmir	Fethiye	09:00	07:50
13	Fethiye	İstanbul	17:00	17:50
14	İstanbul	Marmaris	15:00	15:50
15	Marmaris	İstanbul	10:30	15:50
16	İstanbul	Marmaris	07:00	15:50
17	Marmaris	İstanbul	00:00	15:50
18	İstanbul	Fethiye	17:00	17:50
19	Fethiye	İzmir	12:30	08:05
20	İzmir	Alanya	23:00	10:50
21	Alanya	İzmir	10:00	12:20

22	İzmir	Fethiye	00:00	08:00
23	Fethiye	İzmir	08:30	08:05
24	İzmir	Datça	18:00	07:35

Tablo 4.1 Uygulama Verileri

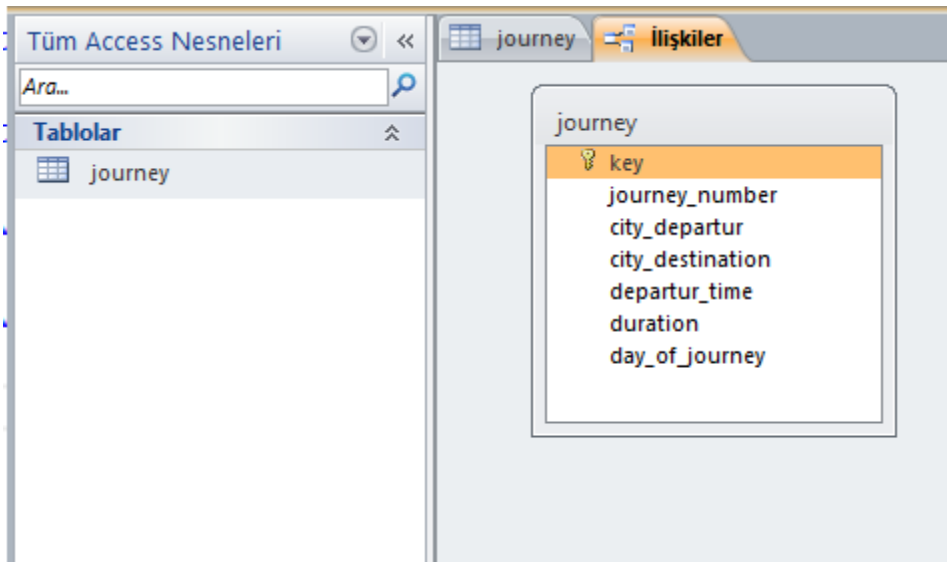
4.3. GERİ İZLEME ÇÖZÜMÜ İÇİN YAZILIM

Uygulama için hazırlanan yazılım Microsoft Visual Studio 2012 paket programının altında yer alan C# programlama dilinde yazılmıştır.

4.3.1. Uygulamada Kullanılan Veri Tabanı

Uygulamada Microsoft Access 2010 veri tabanı kullanılmıştır. Veri tabanı yerine düz metin belgesi şeklinde bir liste de kullanılabilir fakat bu liste üzerinde program aracılığı ile değişiklik yapmak program içerisinde istenmeyen problemlere sebep olabileceği için veri tabanı kullanılması uygun görülmüştür. Veri tabanı yerine yine de verileri saklamak için Microsoft Excel de alternatif olarak kullanılabilir.

Veri tabanında şekil 4.1'de görüldüğü gibi sadece bir tablo kullanılmıştır.



Şekil 4.1 Veri Tabanı Tablosu

Tablo adı journey olarak geçmektedir. Journey tablosu altında 7 tane alan bulunmaktadır. Bu alanlar aşağıda yer alan Tablo 4.2'de gösterilmiştir.

Alan Adı	**Alan ile ilgili açıklama**
key	Anahtar değer
Journey_number	Sefer numarası
city_departur	Kalkış şehri
city_destination	Hedef şehir
departur_time	Kalkış zamanı
duration	Süre
day_of_journey	Yolculuk günü

Tablo 4.2 Journey Tablosunda Yer Alan Alanlar

Journey tablosunda yer alan her bir alanın değişken türü aşağıda yer alan Tablo 4.3'teki gibidir.

Alan Adı	**Veri Türü**
key	Uzun tamsayı (Otomatik)
Journey_number	Metin
city_departur	Metin
city_destination	Metin
departur_time	Metin
duration	Metin

day_of_journey	Metin

Tablo 4.3 Journey Tablosu Değişken Türleri

4.3.2. Yazılımın Algoritması

Problemin çözümü için geliştirilen yazılımın özet halindeki algoritması aşağıdaki gibidir. Programda bu algoritmayı temsil eden program kodu *geriİzleme()* (*backTracking()*) fonksiyonu olarak geçer.

geriİzleme() fonksiyonu için algoritma:

1. Başla
2. **Değişkenler:**

   şehirİndeksi = 0

   turSayısı = 0

   seferSayısı = 24

   aylakZaman = 0

   enKüçükAylakZaman = 0

   rota = “”

   enİyiRota = “”

   Fonksiyonlar:

   ileriAdım()

   geriAdım()
3. Her bir şehir için küme yapısını (her bir ilden gidilebilecek iller) oluştur.

4. Oluşturulan kümeyi alfabetik olarak sıralı hale getir (işlemleri kolaylaştırdığından dolayı).

5. **Döngü başlangıcı** (şehirİndeksi < > -1 veya turSayısı < > 0) şartları doğru olduğu sürece döngüye devam et

   şehirİndeksi = *ileriAdım()* (Bir sonraki şehir indeksini bulan fonksiyon)

   Eğer şehirİndeksi = -1 veya aylakZaman > enKucukAylakZaman **ise**

      geriAdım()

      git Adım 5

   Eğer turSayısı = seferSayısı **ise**

      enKucukAylakZaman = aylakZaman

      enİyiRota = rota

      geriAdım()

   Döngü sonu

6. enİyiRota ve enKüçükAylakZaman'ı yazdır.

7. Dur

Programda en önemli fonksiyonlar *ileriAdım()* ve *geriAdım()* fonksiyonlarıdır. *ileriAdım()* fonksiyonu aracılığı ile bir sonraki gidilecek şehrin indeks numarası hesaplanıp bulunuyor. Eğer gidilebilecek şehir yok ise veya rota oluşturulurken çözümsüz bir rota oluştu ise fonksiyon geriye -1 değeri gönderiyor. Böylece ileriye gidilemeyeceği anlaşılıp *geriAdım()* fonksiyonu çalıştırılıyor. *geriAdım()* fonksiyonu aracılığı ile bir önceki şehre dönülür. Önceden ayarlanan rota ve aylak zamanlar bir adım önce ziyaret edilen şehirler çıkartılarak tekrardan güncellenir. Önceden gidilen şehre tekrar gidilmeyecek şekilde bir adım için geçerli olacak şekilde şehir indeksine engel konulur. Böylece *geriAdım()* fonksiyonu aracılığı ile geriye dönülünce aynı şehrin tekrar ziyaret edilmesi bir adım için engellenmiş olur. Böylece engellenmiş şehir yerine alternatif şehirler denenmiş olur.

Programda her ziyaret edilen şehir için şehir bilgilerinin yer aldığı kümeye işaret konulur. Böylece aynı şehir, tanımlı olduğu için tekrar ziyaret edilmez. *geriAdım()* fonksiyonu her çalıştırıldığında bu işaretler sırası ile kaldırılır.

4.3.2.1. Yardımcı Fonksiyonların Algoritması

Algoritmada yer alan *ileriAdım()* ve *geriAdım()* yardımcı fonksiyonlarının ayrıntıları aşağıdaki gibidir.

ileriAdım() (*moveForward()*) fonksiyonu için algoritma:

1. Başla
2. **Değişkenler:**

   indeks

   aylakZaman = 0

   rota = “”

   turSayısı = 0

   yolBilgisi

   Fonksiyonlar:

   ziyaretEdilmemişŞehirBul()

   yasaklıŞehriSerbestBırak()
3. İndeks = *ziyaretEdilmemişŞehirBul()*
4. **Eğer** indeks = -1 veya indeks = -2 **ise**

      geriAdım()

      git Adım 5

   değilse

yasaklıŞehriSerbestBırak()

aylakZaman değişkenine küme üzerinden aylak zamanı hesaplayıp gönder

rota değişkenine küme üzerinden rotayaı hesaplayıp gönder

turSayısı = turSayısı + 1

Bir önceki indeks, ve aylakZaman bilgilerini yolBilgisi değişkenine yaz

Bulunan yeni indeks numarasını üst fonksiyona gönder

5. Dur

ileriAdım() algoritmasında eğer gidilecek uygun bir şehir bulunamadı ise ya da çözümsüz bir tur oluştu ise algoritmada yer alan *ziyaretEdilmemişŞehirBul()* fonksiyonu geriye -1 ya da -2 değeri gönderir. Bu durumda algoritmada *geriAdım()* fonksiyonu çalıştırılır ve rota üzerinde bir adım geriye gidilir. *geriAdım()* fonksiyonu çalıştırıldığında *ileriAdım()* fonksiyonu burada sonlandırılır. Bu nedenle algoritmada bu aşamadan sonra son adıma gidilir.

Eğer uygun bir şehir bulundu ise o şehre ait indeks numarası *ziyaretEdilmemişŞehirBul()* fonksiyonu aracılığı ile bulunup indeks değişkenine aktarılır. Bu aşamadan sonra indeks değeri bir üst fonksiyona gönderilmeden önce rotada geri adım atılırsa, bazı kritik bilgiler gerekli olabileceğinden bu bilgiler yolBilgisi değişkenine yazılır. Rotada bir adım ilerlendiğinden turSayısı değişkeni bir artırılır. Bu işlemler tamamlandıktan sonra indeks değeri bir üst fonksiyona (*geriİzleme()*) gönderilir.

geriAdım() (*moveBackward()*) fonksiyonu için algoritma:

1. Başla
2. **Değişkenler:**

   indeks

   aylakZaman = 0

rota = “”

turSayısı = 0

yolBilgisi

Fonksiyonlar:

yasaklıŞehriSerbestBırak()

3. **Eğer** yolBilgisi = “” **ise**

   Geriye -1 gönder son adıma git

4. **Eğer** yasaklı şehir var **ise**

   yasaklıŞehriSerbestBırak()

5. rota değişkeninden en son girilen rotayı çıkart

   aylakZaman değişkeninden en son girilen aylak zamanı çıkart

   yolBilgisi değişkeninden en son girilen yolu çıkart

   turSayısı = turSayısı – 1

   indeks değerini küme üzerinden bulup hesapla ve üst fonksiyona gönder

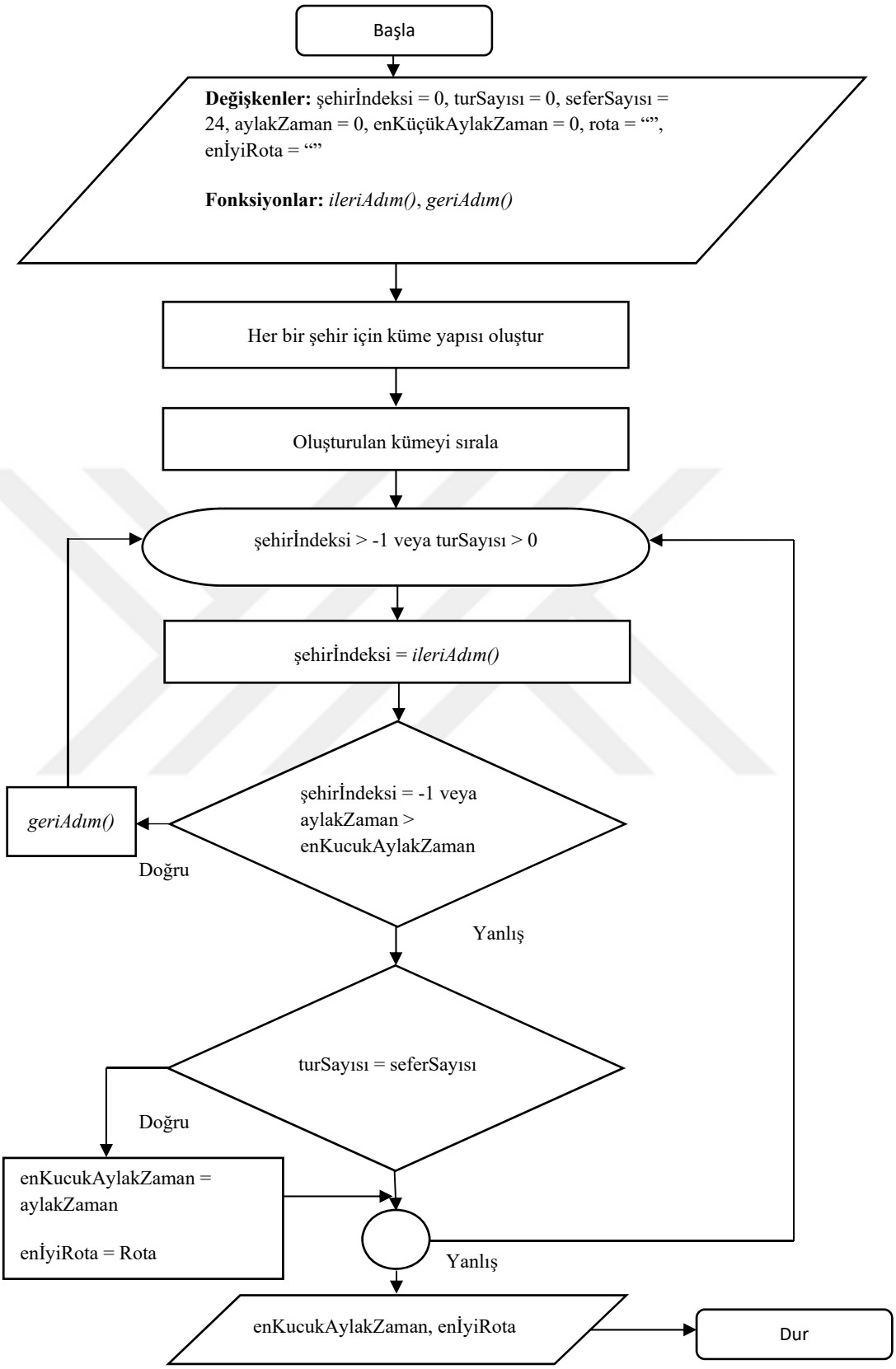
6. Dur

GeriAdım() algoritmasında eğer yolBilgisi değişkeninde hiçbir değer yoksa rota üzerinde en başa (ilk şehre) dönüldüğü anlamına gelir. Bu nedenle yolBilgisi değişkeni bir değer tutmuyorsa bunu üst fonksiyona bildirmek için geriye indeks numarası olamayacak bir numara olan -1 (indeks numaraları negatif olamaz) gönderilir. Üst fonksiyon -1 değerinden yararlanarak *gerAdım()* fonksiyonunun çalıştırılmayıp geri adım atılmadığını anlayacaktır. Bu durumda üst fonksiyon tur üzerinde başa dönüldüğünü ve geri adım atılamadığını anlayacak ve fonksiyonu sonlandıracaktır. O ana kadar bulunan en uygun çözüm ise problemin en iyi çözümü olarak geriye gönderilecektir.

GeriAdım() algoritmasının 4. Adımında, daha öncesinde de *geriAdım()* fonksiyonu çalıştırıldı ve geriye doğru adım atıldı ise ziyaret edilmemesi gereken yasaklı bir şehir olacağından *yasaklıŞehriSerbestBırak()* fonksiyonu aracılığı ile gidilmemesi gereken şehir ile ilgili yasak kaldırılır. Öncesinde de belirtildiği gibi ziyaret edilmemesi gereken şehre sadece bir adım için engel konulmuştu. Bu nedenle bir adım sonra böyle bir engel kaldı ise *yasaklıŞehriSerbestBırak()* fonksiyonu aracılığı ile bu engel kaldırılır.

4.3.3. Geri İzleme Çözümü için Akış Diyagramı

Yazılımda problemin çözümünü bulabilen fonksiyon Geri İzleme fonksiyonudur. Geri İzleme fonksiyonuna ilişkin akış diyagramı Şekil 4.2'deki gibidir.

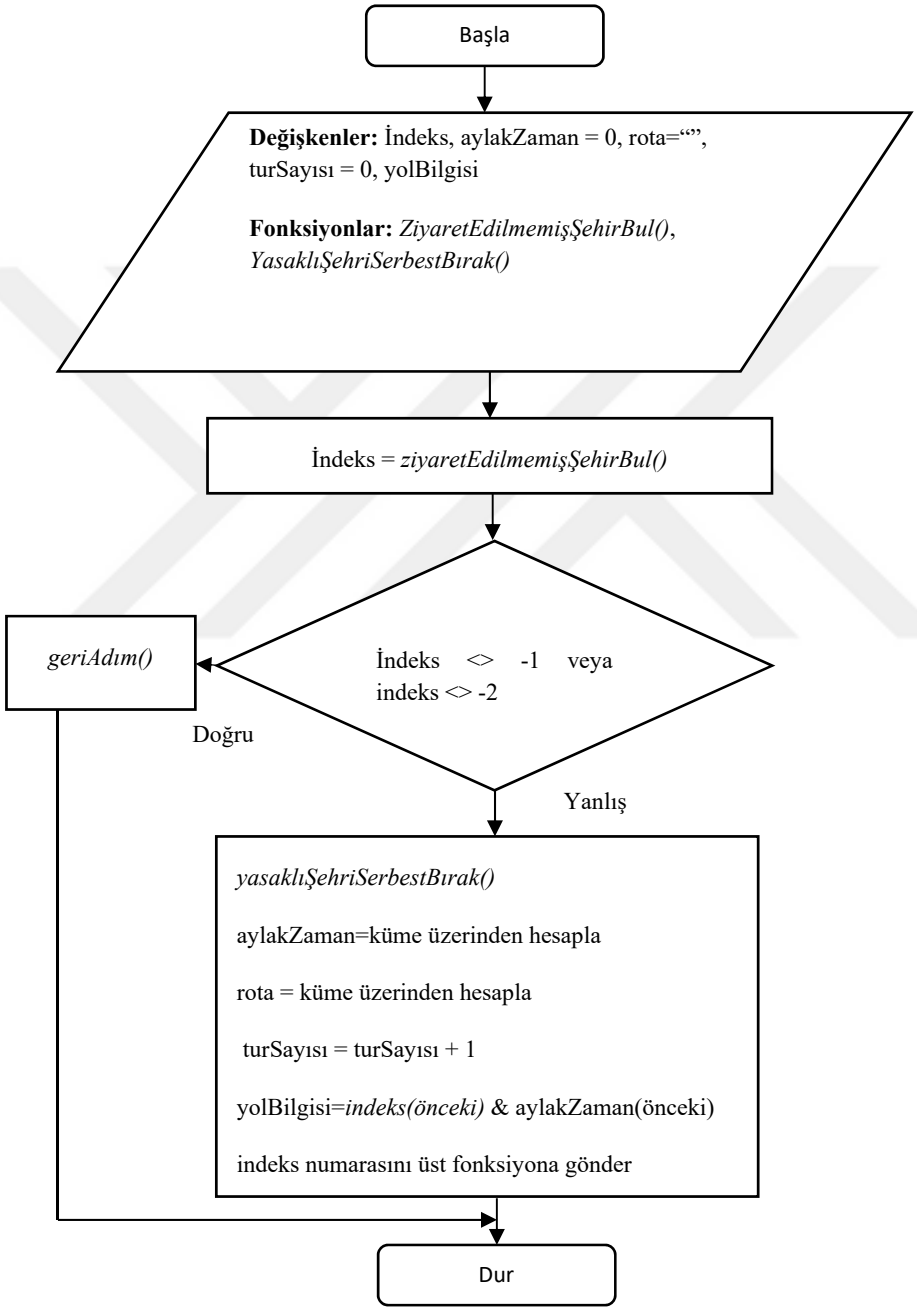


Şekil 4.2 Geri İzleme Fonksiyonu Akış Diyagramı

4.3.3.1. Yardımcı Fonksiyonların Akış Diyagramı

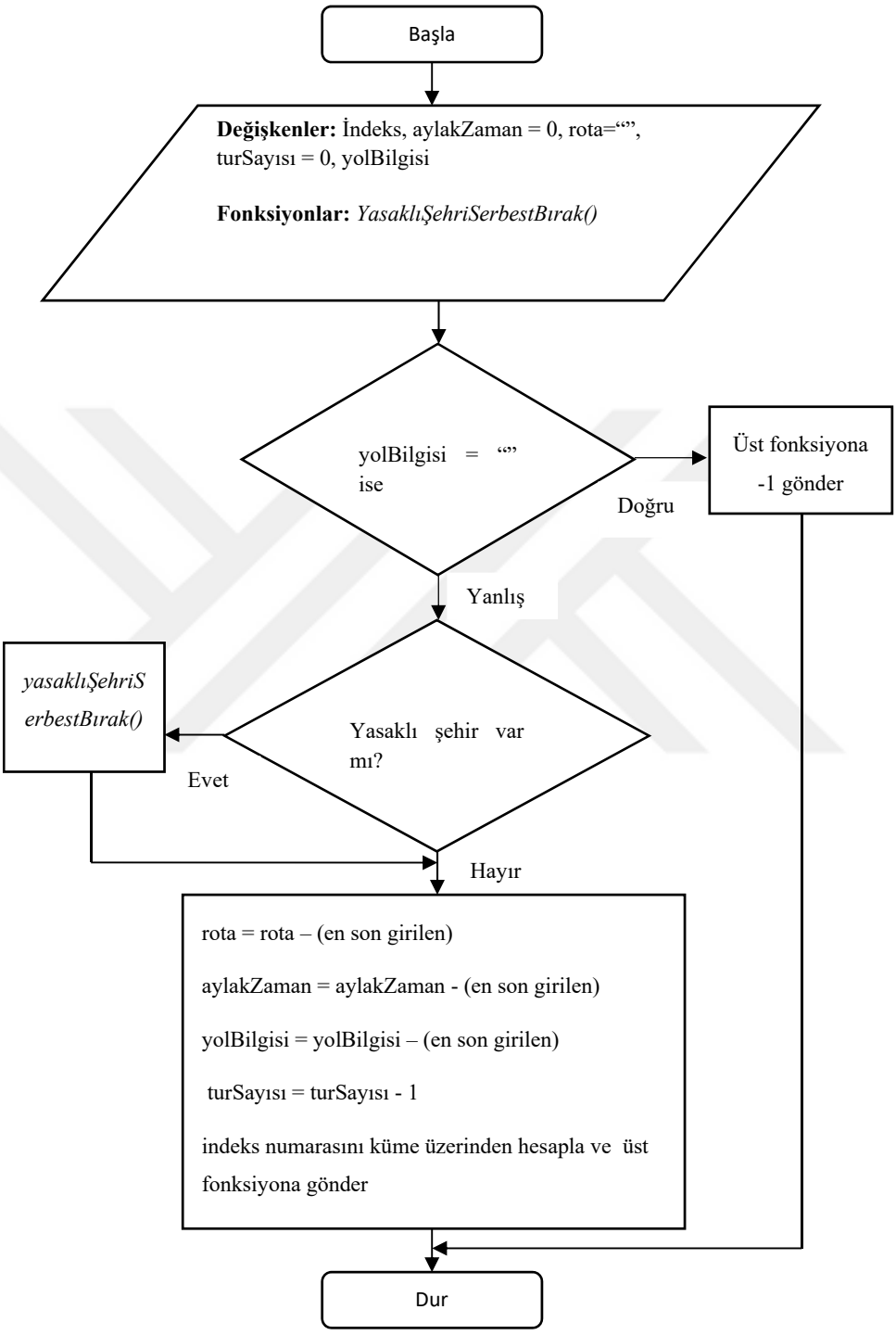
Şekil 1'de yer alan *ileriAdım()* ve *geriAdım()* fonksiyonlarına ait akış diyagramı ise aşağıda yer alan şekil 4.3 ve şekil 4.4'deki gibidir.

ileriAdım() fonksiyonu için akış diyagramı:



Şekil 4.3 İleri Adım Fonksiyonu Akış Diyagramı

geriAdım() fonksiyonu için akış diyagramı:



Şekil 4.4 Geri Adım Fonksiyonu Akış Diyagramı

4.3.4. Yazılımda Yer Alan Program Kodlarının Ayrıntıları

Programda yer alan veriler veri tabanından okunup bir ağaç yapısı şeklinde dizi oluşturularak tutulmuştur. Oluşturulan bu ağaç yapısı üzerinde ileri ve geri adımlar şeklinde Dal ve Sınır Geri İzleme yaklaşımına uygun olacak şekilde dolaşılarak problem için en iyi çözüm aranıp bulunmuştur. Çözüm araştırılırken program içerisinde birçok fonksiyon ve yardımcı fonksiyonlar kullanılmıştır.

4.3.4.1. Yazılımda Kullanılan Fonksiyonlar

Program kodlarında birçok temel fonksiyon ve yardımcı fonksiyon yazılıp kullanılmıştır. Bu fonksiyonlardan bir kısmı aşağıda yer alan Tablo 4.4'teki gibidir.

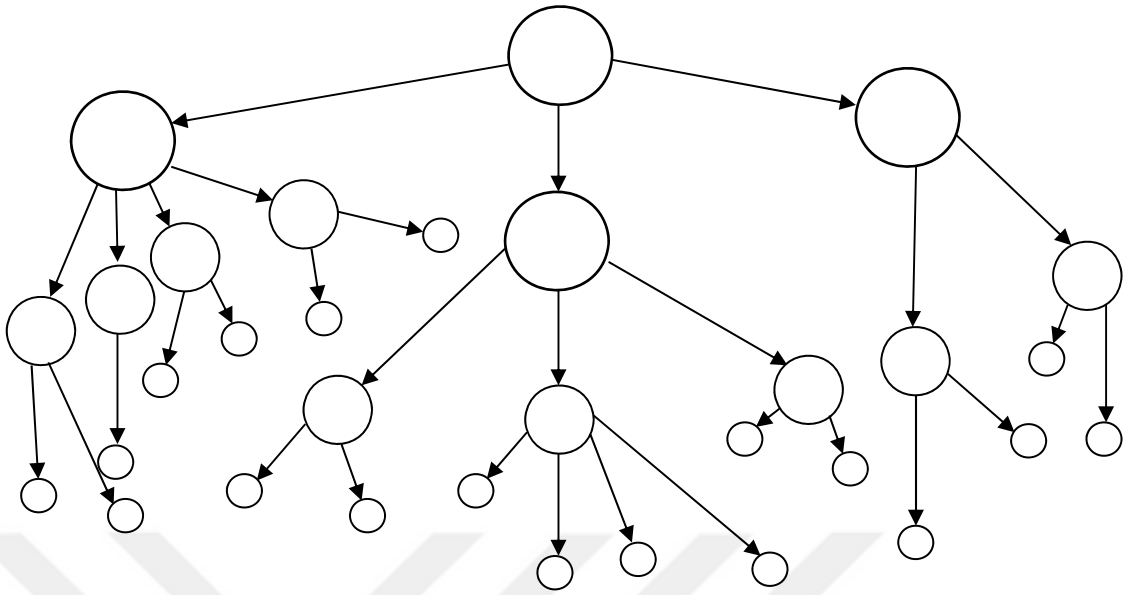
Fonksiyon Adı	**Açıklama**
backTracking()	Geri İzleme Yaklaşımı fonksiyonu (*geriİzleme()*)
createCluster()	Küme yapısı oluşturan fonksiyon
sortCluster()	Kümeyi sıralayan fonksiyon
createCityList()	Şehir listesi oluşturan fonksiyon
cityCount()	Şehir sayısını bulan fonksiyon
findCityIndex()	Şehir indeksi bulan fonksiyon
getNodeCount()	Ziyaret edilmemiş şehir sayısını veren fonksiyon
indexFormat()	İndeks verisini belirli bir formata dönüştüren fonksiyon
getTime()	Rotada o anki zamanı veren fonksiyon
bigString()	İki kelimeden büyük olanını veren fonksiyon

moveForward()	Rotada bir sonraki adıma gitmek için kullanılan fonksiyon (*ileriAdım()*)
moveBackward()	Rotada bir önceki adıma gitmek için kullanılan fonksiyon (*geriAdım()*)
releaseForbiddenCity()	Yasaklı şehri yasak listesinden kaldıran fonksiyon (*yasaklıŞehriSerbestBırak()*)
notForbiddenCity()	Yasaklı olmayan bir şehir bulan fonksiyon
findNonVisitedCity()	Daha önce gidilmemiş bir şehir bulan fonksiyon (*ziyaretEdilmemişŞehirBul()*)
findBestSlackTime()	En iyi aylak zamanı hesaplayan fonksiyon
changeString()	Ziyaret edilmiştir işareti koyan veya kaldıran fonksiyon
refreshTime()	O anki zamanı yenileyen fonksiyon

Tablo 4.4 Fonksiyonlar ve İşlevleri

4.3.5. Küme Yapısı

Programda veriler öncelikle küme haline dönüştürülmüştür. Diğer bir deyişle ağaç yapısı şeklinde bir dizi oluşturulup tutulmuştur. Küme yapısı aşağıda yer alan şekil 4.5'teki gibidir. Burada her bir halka bir düğümü temsil eder. Her bir halkanın bağlı olduğu diğer halkalar da gidilebilecek diğer düğümleri gösterir. Problemin çözümü araştırılırken düğümden düğüme, düğümler üzerinden geçiş yapılarak (ileri ya da geri adım atılarak) en iyi çözüm bulunmuştur.



Şekil 4.5 Küme Yapısı

Küme yapısına sahip bu dizi program içerisinde küme olarak geçmektedir. Küme içerisindeki veriler aşağıdaki şekilde yer almaktadır.

Küme[0] = 002000119:0013:50001000210:0008:05003

Burada küme[**0**] içerisindeki **0** şehir indeksini temsil etmektedir. Şehir listesi 0'dan başlayan indeks numaraları şeklinde aşağıdaki gibi tutulmaktadır.

Datça = küme[0], İzmir = küme[1], Aydın = küme[2], Alanya = küme[3] … gibi.

Kümenin sağ tarafında yer alan bilgiler kümenin içerdiği bilgilerdir. Bu verilerin anlamı kısaca aşağıda yer alan Tablo 4.5'teki gibidir.

Küme İçeriği	**Küme Verisi**	**Verinin Anlamı**
002000119:0013:50001000210:0008:05003	**002**	Verilen şehirden gidilebilecek olan şehir sayısı

002**0**00119:0013:50001000210:0008:05003	**0**	Gidilebilecek olan şehrin ziyaret edilip edilmediği
0020**001**19:0013:50001000210:0008:05003	**001**	Gidilebilecek şehrin indeksi
0020001**19:00**13:50001000210:0008:05003	**19:00**	Bulunan şehirden gidilebilecek şehre seferin başladığı zaman
002000119:00**13:50**001000210:0008:05003	**13:50**	Sefer süresi
002000119:0013:50**001**000210:0008:05003	**001**	Sefer numarası

Tablo 4.5 Küme Yapısı

Bu bilgilere göre Datça şehri İzmir ve Aydına bağlı görünüyor. Diğer bir deyişle Datça şehrinden sadece İzmir ve Aydına gidilebiliyor. Datça şehrini temsil eden küme[0]'a baktığımızda,

0020**001**19:0013:50001**002**10:0008:05003

001 = İzmir, **002** = Aydın olduğu görülüyor. Datça şehrinden gidilebilecek şehir sayısı ise,

002000119:0013:50001000210:0008:05003

en başta yer alan **002** sayısı ile iki şehir olarak belirtilmiştir. Ziyaret edilen her şehre karşılık bu sayı bir azaltılmaktadır.

İzmir ve Aydın şehirlerini temsil eden sayıların başlarında yer alan,

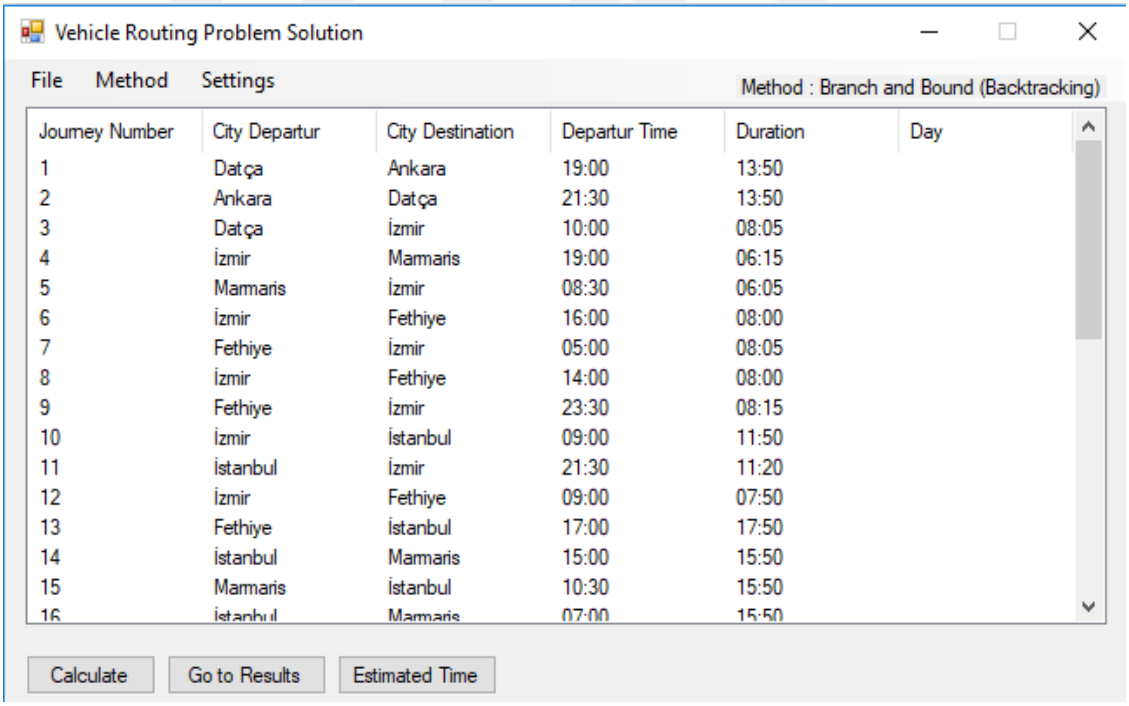
002**0**00119:0013:50001**0**00210:0008:05003

iki **0** rakamı ise şehirlerin ziyaret edilmediğini gösteriyor.

Program çalışırken bir şehir ziyaret edilirse küme üzerinde yer alan bilgilerdeki ziyaret edilip edilmediği ile ilgili sıfır değeri bir yapılmaktadır. Böylece programda bir olan düğümlere uğranmaz. Rota üzerinde geri adım atılırken ise bir değeri tekrar sıfır yapılır. Geri adım atılırken bir değeri hemen sıfır yapılmaz. Ancak bir adım (ileri ya da geri) daha atıldıktan sonra ziyaret edilmedi bilgisini temsil eden sıfır değeri bir değeri yerine yazılır. Eğer bir adım daha atılmasını beklemeden değişiklik yapılırsa program geri adımdan sonra ileri adım attığında tekrar aynı şehri ziyaret etmeye kalkacaktır. Bu durumda alternatif şehirler hiçbir zaman araştırılamayacak ve program sonsuz döngüye girecektir.

4.3.6. Yazılımın Ekran Görünümü

Geliştirilen yazılımda üç *Form* penceresi yer almaktadır. İlk pencerenin ekran görünümü şekil 4.6'daki gibidir.



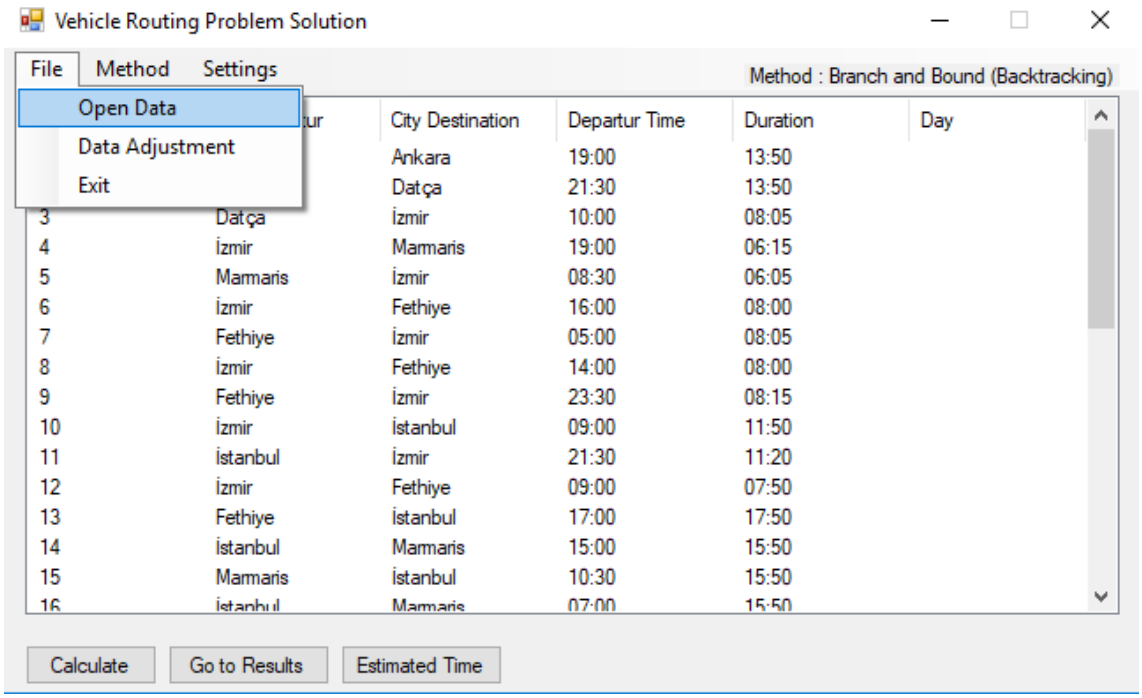
Şekil 4.6 Yazılımın Ekran Görünümü

Şekil 7'de otobüs seferleri ile ilgili ayrıntılı bilgiler görünmektedir. Bu bilgilerin anlamları aşağıda yer alan Tablo 4.6'da gösterilmiştir.

Ekran Görünümündeki İsim	**Anlamı**
Journey Number	Sefer Numarası
City Departur	Kalkış Şehri
City Destination	Hedef Şehir
Departur Time	Kalkış Zamanı
Duration	Süre
Day	Gün

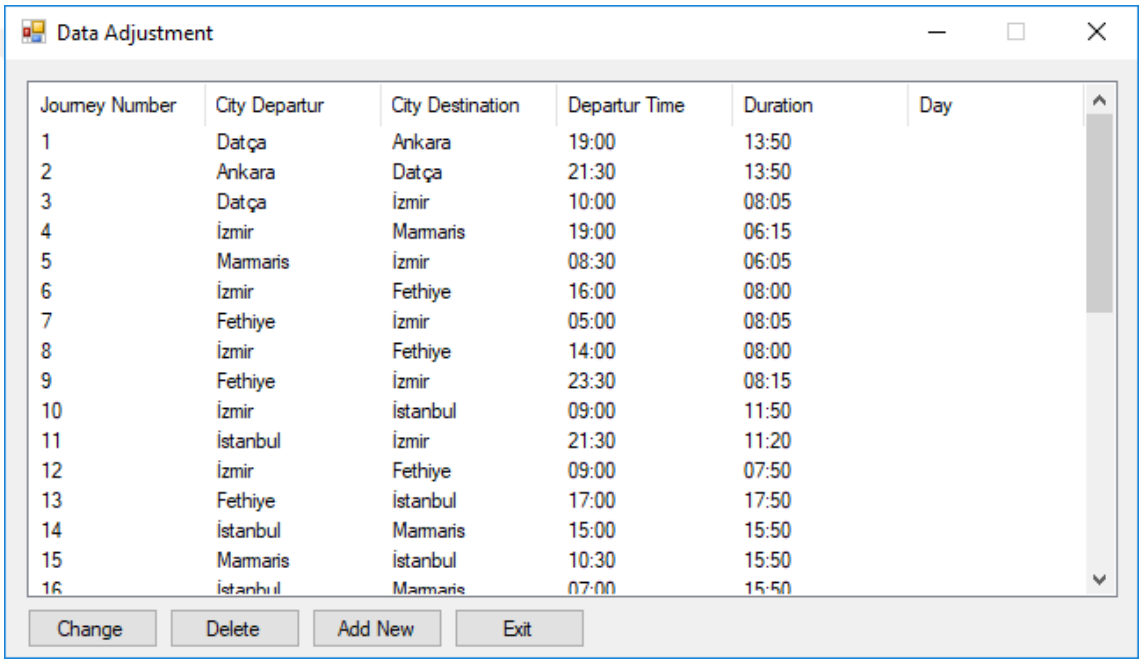
Tablo 4.6 Ekran Görünümündeki Bilgiler

şeklindedir. Buradaki veriler Access veri tabanından çekilmektedir. Farklı bir veri tabanından veri çekmek istediğimizde Şekil 4.7'deki gibi *File* menüsünden *Open Data* seçeneğini seçiyoruz. Buradan açılan pencere ile veri tabanını seçip verileri ekrana yansıtıyoruz.



Şekil 4.7 Ekran Görünümündeki File (Dosya) Menüsü

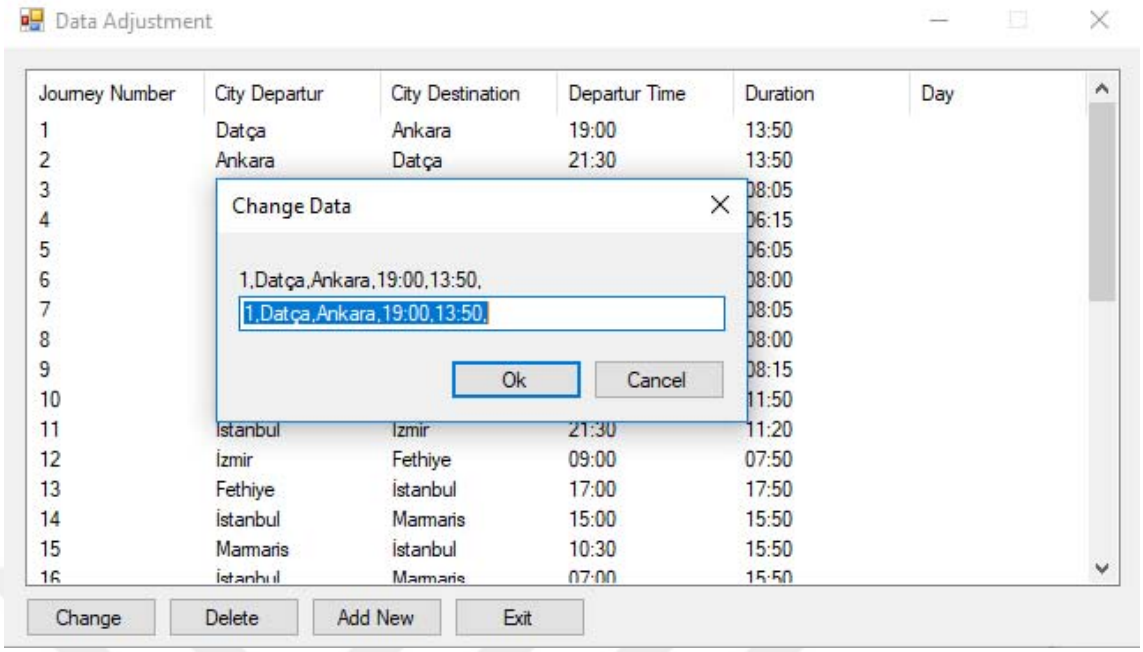
İkinci *Form* penceresi veriler üzerinde değişiklik yaptığımız penceredir. Bu pencereyi *File* menüsünden *Data Adjustment* seçeneğini seçerek açıyoruz. Bu seçenekten sonra şekil 9'daki gibi yeni bir pencere karşımıza çıkıyor. Bu pencereden sadece veriler üzerinde değişiklik yapılabiliyor. Bu değişikliler var olan veriyi değiştirme, var olan veriyi silme, yeni bir veri ekleme şeklindedir. Değişiklikleri şekil 4.7'de görüldüğü gibi *Change* butonuna basarak değiştirebileceğimiz gibi pencere üzerinde yer alan verinin üzerine çift tıklayarak da yapabiliriz. Verinin üzerine çift tıkladığımızda açılan pencere şekil 4.8'de görünmektedir.



Journey Number	City Departur	City Destination	Departur Time	Duration	Day
1	Datça	Ankara	19:00	13:50	
2	Ankara	Datça	21:30	13:50	
3	Datça	İzmir	10:00	08:05	
4	İzmir	Marmaris	19:00	06:15	
5	Marmaris	İzmir	08:30	06:05	
6	İzmir	Fethiye	16:00	08:00	
7	Fethiye	İzmir	05:00	08:05	
8	İzmir	Fethiye	14:00	08:00	
9	Fethiye	İzmir	23:30	08:15	
10	İzmir	İstanbul	09:00	11:50	
11	İstanbul	İzmir	21:30	11:20	
12	İzmir	Fethiye	09:00	07:50	
13	Fethiye	İstanbul	17:00	17:50	
14	İstanbul	Marmaris	15:00	15:50	
15	Marmaris	İstanbul	10:30	15:50	
16	İstanbul	Marmaris	07:00	15:50	

Şekil 4.8 Data Adjustment Ekran Görünümü

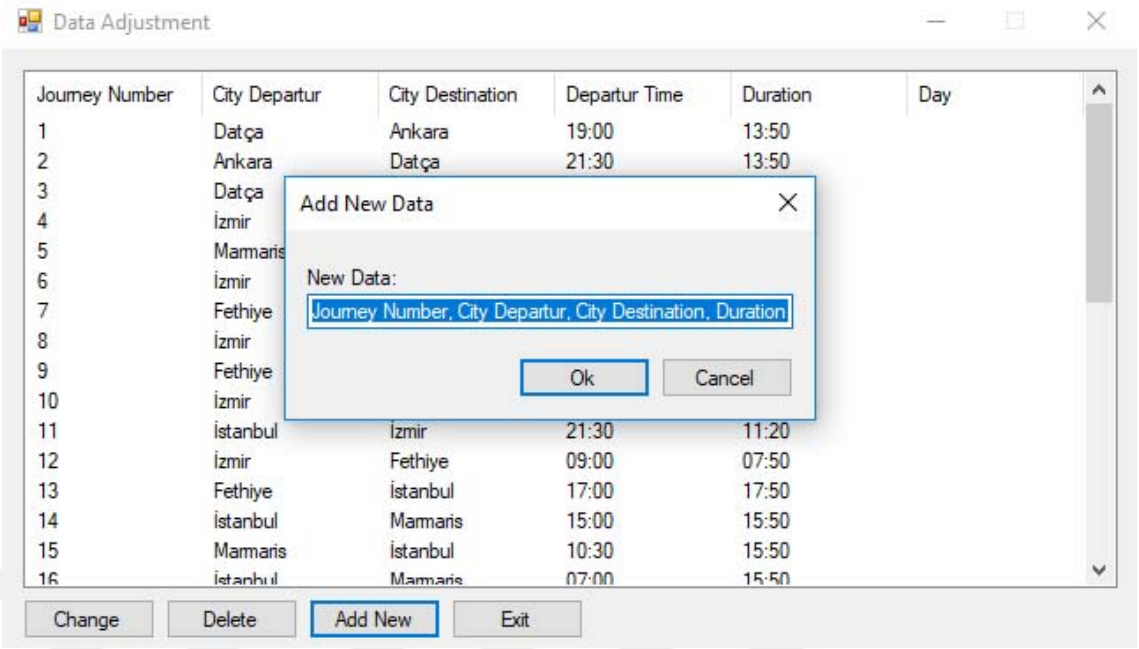
Bunların dışında *Delete* butonuna bastığımızda o an seçili olan veri silinir.



Şekil 4.9 Change Data Ekran Görünümü

Add New butonuna bastığımızda ise altta yer alan şekil 5.0'daki gibi küçük bir pencere açılmaktadır. Bu pencereye veri eklerken veriler arasında virgül bulunması gerekir. Buraya yazdığımız veriden sonra *Ok* butonuna bastığımızda pencerede yer alan veri satırı verilerin var olduğu veri tabanına yeni bir kayıt olarak eklenecektir. *Cancel* butonuna bastığımızda ise veri ekleme işlemi iptal edilecektir.

Data Adjustment penceresinde yer alan *Exit* butonuna bastığımızda ise pencere kapanır.

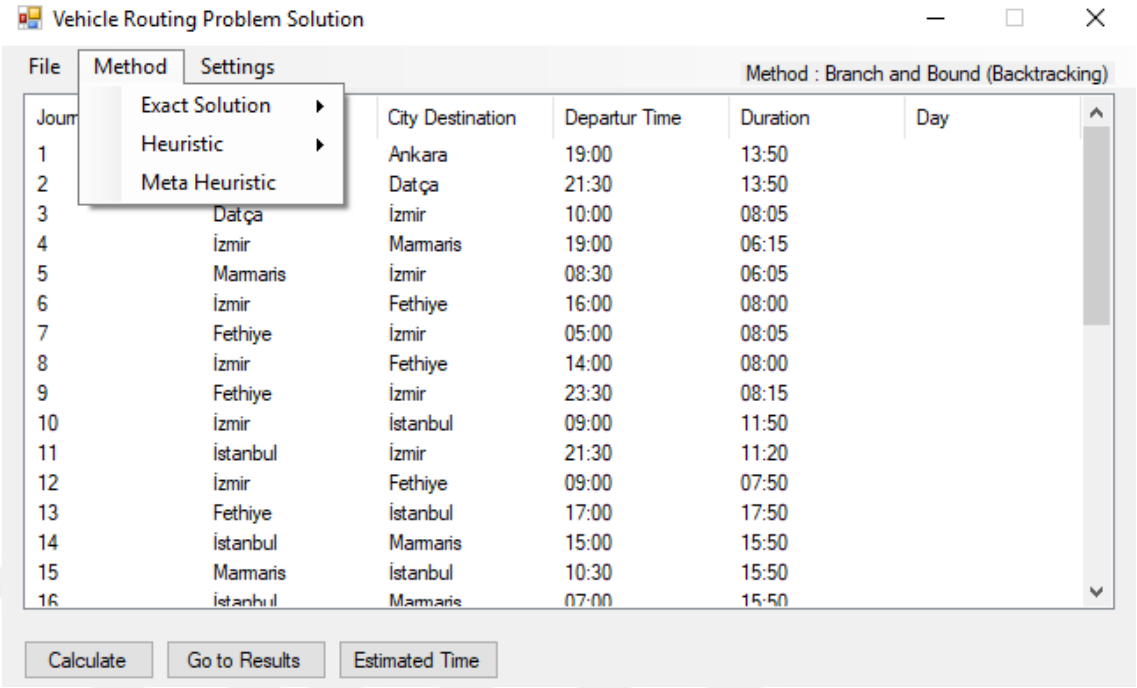


Şekil 5.0 Add New Data Ekran Görünümü

Herhangi bir verinin üzerine çift tıkladığımızda yukarıda yer alan şekil 6'daki gibi bir pencere karşımıza çıkmaktadır. Bu pencerede üzerine çift tıklanan verinin ayrıntıları görünmektedir. Burada veriler arasında virgül olacak şekilde istediğimiz kısımları silip değiştirdiğimizde ve ardından *Ok* butonuna bastığımızda değişiklikler veri tabanına kaydedilip *Data Adjustment* penceresine otomatik yansıtılacaktır. *Cancel* butonuna basıldığında ise pencerede değişiklik yapılsa da veri tabanına ekleme yapılmayıp herhangi bir değişiklik olmayacaktır.

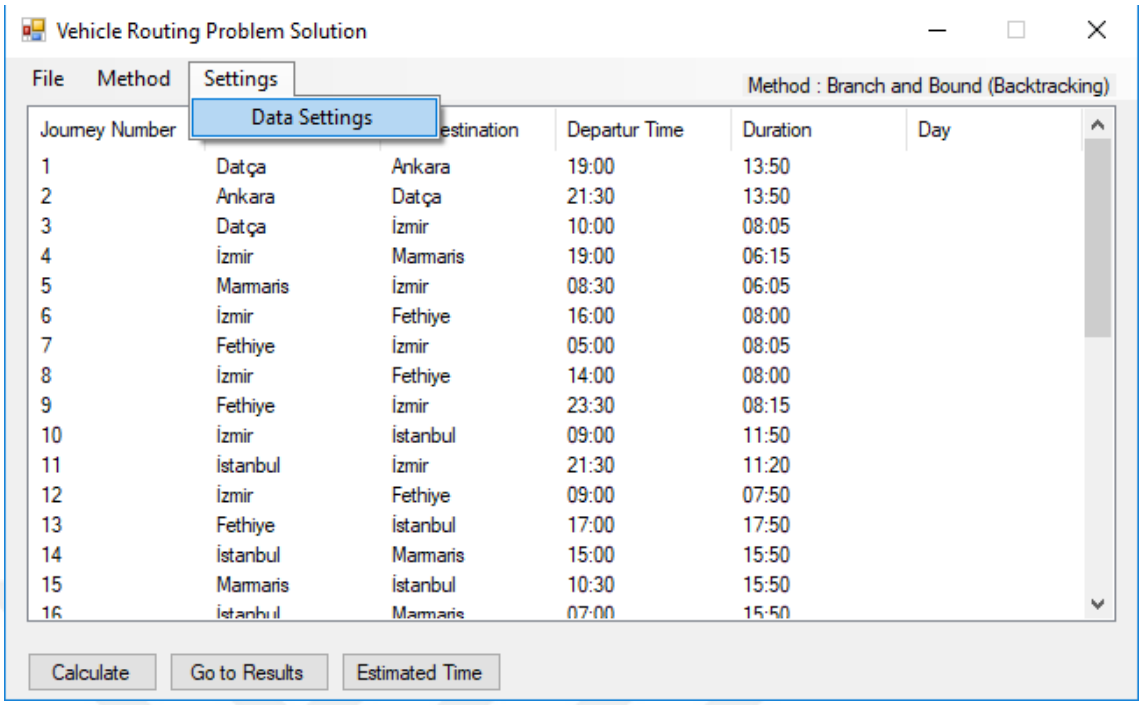
File menüsünde *Exit* seçeneği ise programdan çıkılmasını sağlar.

Aşağıda yer alan şekil 5.1'de görüldüğü gibi menü seçeneklerinden *Method* seçeneği seçildiğinde ise Araç Rotalama Problemlerinin (ARP) çözümü ile ilgili seçenekler görülmektedir. Buradan seçilen yönteme göre farklı çözüm yöntemleri ile ilgili seçenekler yer almaktadır. Buradan seçilen yöntem ilk pencerede sağ üstte görülmektedir. Örnek pencerede *Method: Branch and Bound (Backtracking)* (Yöntem: Dal ve Sınır (Geri İzleme)) olarak görülmektedir.



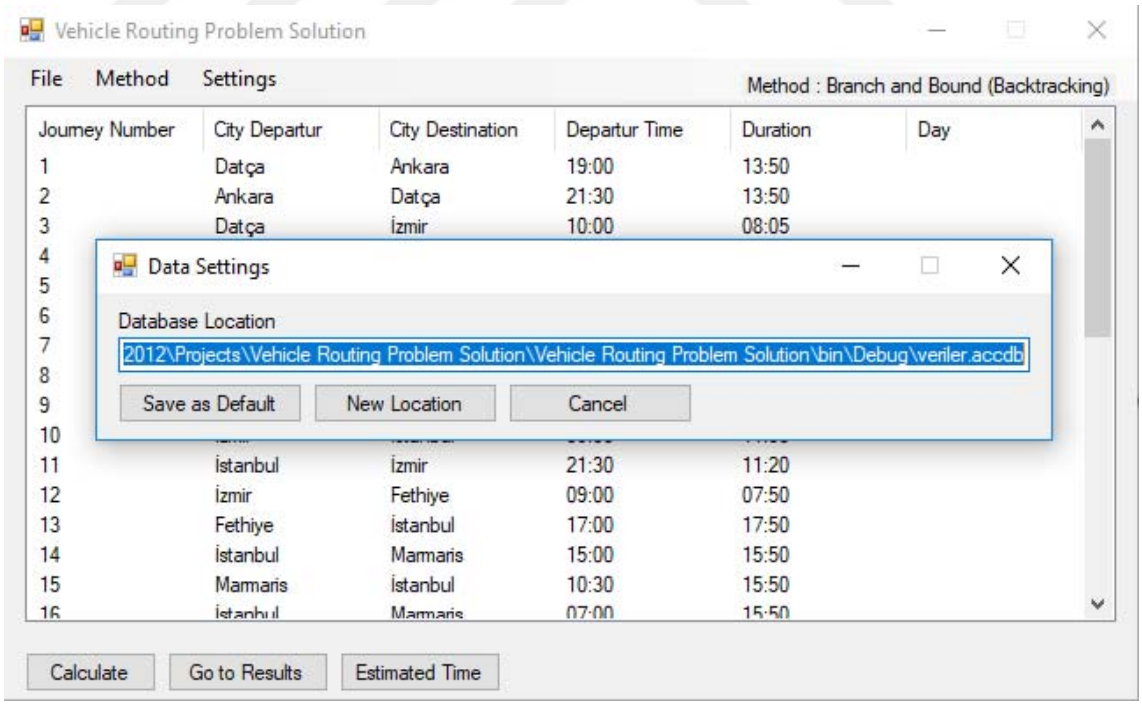
Şekil 5.1 Method Sekmesi Ekran Görünümü

Menü seçeneklerinden *Settings* altında yer alan *Data Settings* ise veri tabanı için geçerli konum ayarlamasını yapmak için kullanılmaktadır. Program tekrar açıldığında, hangi konum ayarlandı ise veri tabanın o konumundan verileri çekip programı açmaya çalışacaktır. Bu seçenek ile ilgili ayrıntılar şekil 5.2'deki gibidir.



Şekil 5.2 Data Settings Sekmesi Ekran Görünümü

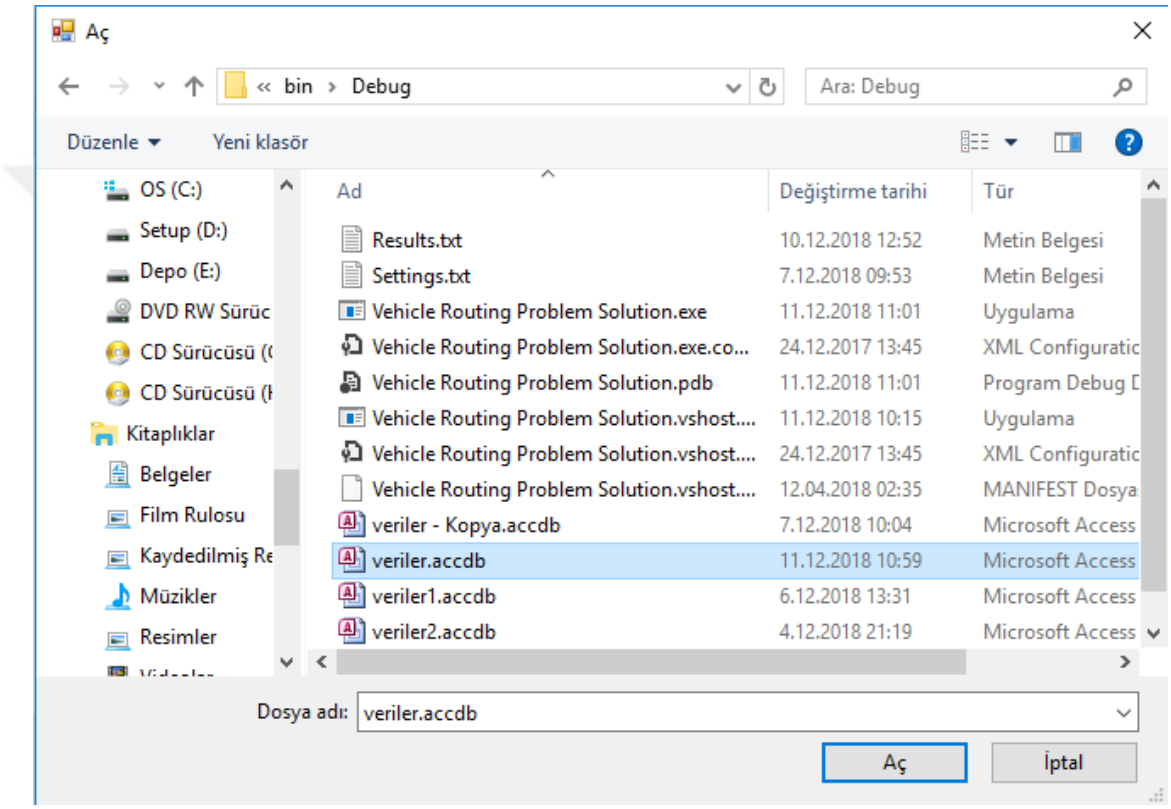
Data Settings'e tıklandığında şekil 5.3'deki pencere açılacaktır.



Şekil 5.3 Data Settings Ekran Görünümü

Burada *Database Location* olarak görünen veri tabanının geçerli konumunu değiştirebiliriz. *Save as Default* butonuna bastığımızda üstteki *Database Location*

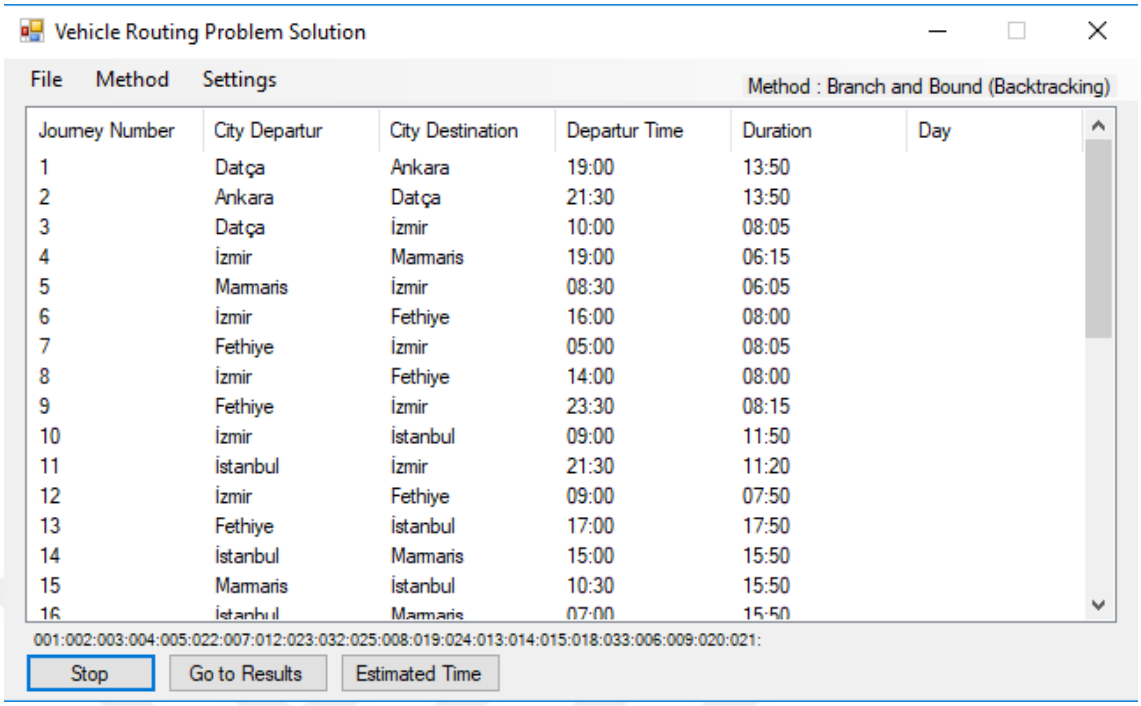
olarak yer alan adres ne ise o adres geçerli adres olarak kaydedilecektir. *New Location* butonuna bastığımızda aşağıda yer alan şekil 5.4'deki gibi yeni bir konum adresi için bir pencere açılacaktır. Bu açılan pencereden hangi veri tabanı dosyasını seçersek *Aç* butonuna bastığımızda o veri tabanı konumu *Database Location* olarak görünecektir. *İptal* butonuna bastığımızda ise konum ile ilgili değişiklik iptal edilmiş olacaktır. Daha sonra yine *Save as Default* butonuna basarak yeni veri tabanı dosyası için konumu kaydetmemiz gerekir.



Şekil 5.4 Adres Konumu Ekran Görüntüsü

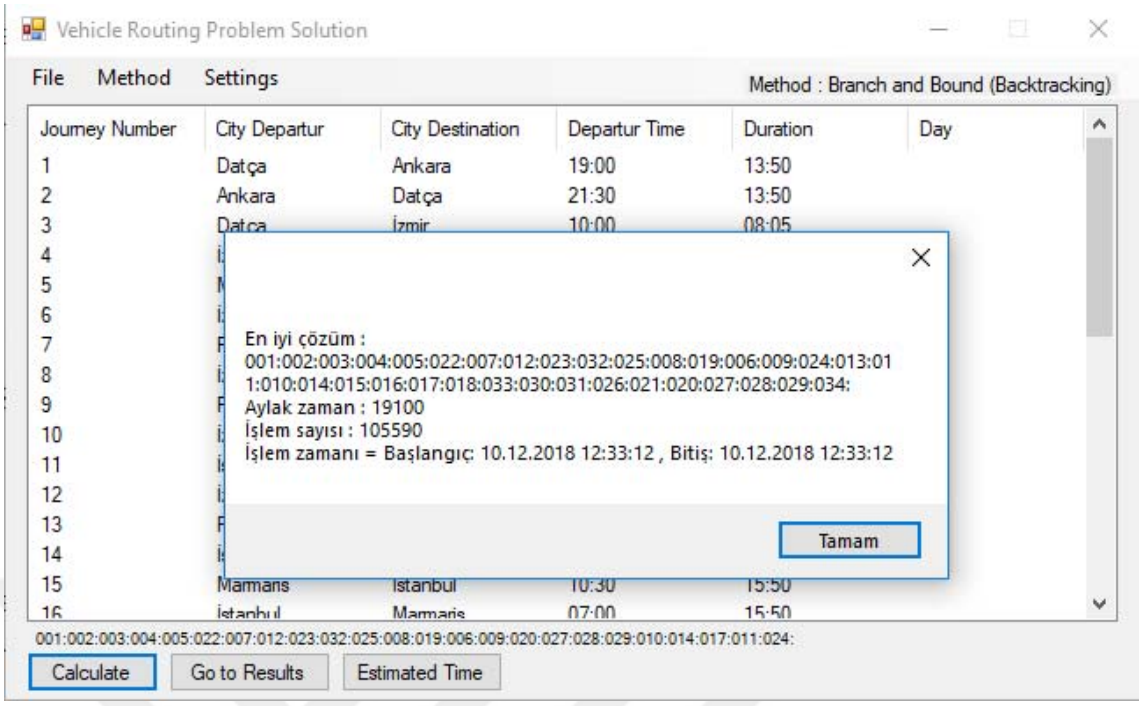
Cancel butonuna bastığımızda ise veri tabanı konumu ile ilgili değişiklik iptal edilmiş olacaktır. Veri tabanı konumu ile ilgili bilgiler programın bulunduğu konumda *Settings.txt* adlı metin belgesine kaydedilmektedir. Bu metin belgesinden konum ile ilgili bilgiyi metin belgesini açarak elle de değiştirebiliriz.

İlk pencerede şekil 5.5'deki gibi yer alan *Calculate* butonuna bastığımızda ise sağ üstte yer alan *Method: Branch and Bound (Backtracking)* olarak ne seçildi ise o yönteme göre program problemin çözümünü bulmaya çalışacaktır.



Şekil 5.5 Calculate Butonu Ekran Görünümü

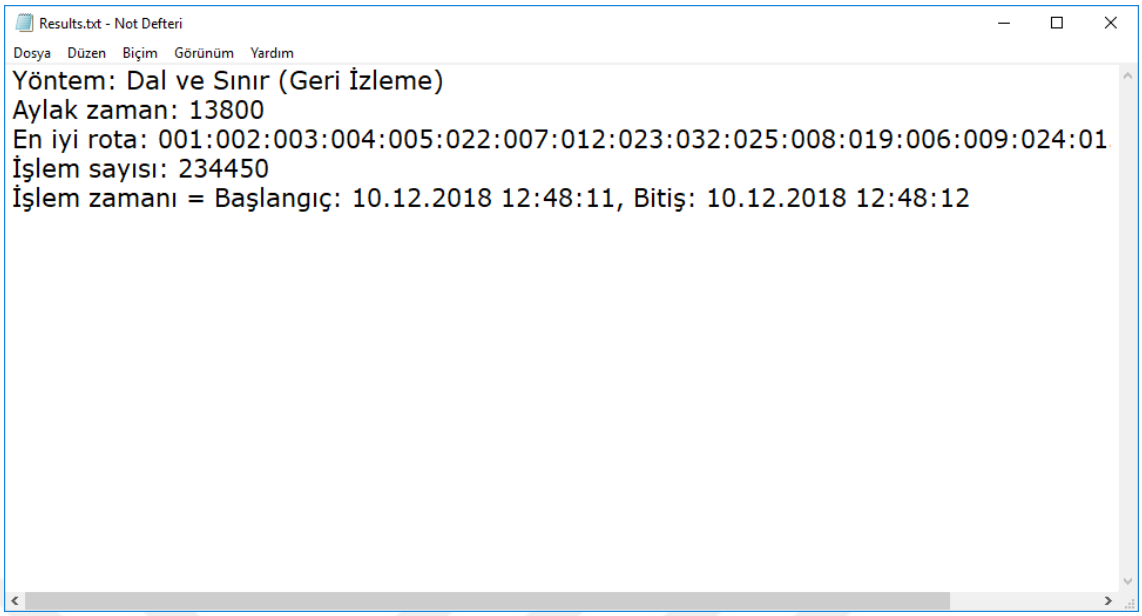
Şekil 16'de *Calculate* butonuna basıldığında değişiklikler görülmektedir. *Calculate* butonunun ismi *Stop* olarak değişmektedir. *Stop* butonuna basıldığında ise hesaplama yarıda kesilecek ve *Stop* butonu ismi tekrar *Calculate* olacaktır. Butonun hemen üstünde yer alan *001:002:003: ...* gibi bilgiler olası rotaları göstermektedir. Bu rotalardan büyük bir kısmı dairesel rota oluşturmamaktadır. Bir kısmı imkânsız rotalar oluşturmaktadır. Küçük bir yüzdesi ise problemin tanımına uygun bir rota oluşturmaktadır. Alternatif rotaların (imkânsız ve eksik rotalar dâhil) tümü içerisinden ekranda 1 milisaniye aralıklarla o an hangi rotada kalındı ise o rota bilgisi görünmektedir. *Stop* butonuna basıldığında o an bulunan en küçük aylak zamanlı en iyi rota ekranda mesaj kutusu ile şekil 17'deki gibi gösterilmektedir. İşlemler yarıda kesildiğinden bulunan bu rota en iyi rota olmayabilir. Problem çözümü çok uzun değil ise hesaplamalar bittikten sonra yine mesaj kutusu ile ekranda en küçük aylak zamanlı en iyi rota ekranda gösterilmektedir.



Şekil 5.6 Stop Butonu Ekran Görünümü

Şekil 5.6'da rota dışında mesaj kutusunda işlem sayısı (her bir ileri ya da geri adım 1 işlem olacak şekilde) görülmektedir. Mesaj kutusunda yer alan *İşlem zamanı* kısmında ise işlemlerin başladığı tarih ve bittiği tarih görülmektedir. İşlem sayısı ortalama olarak saniyede 230.000 işlemdir.

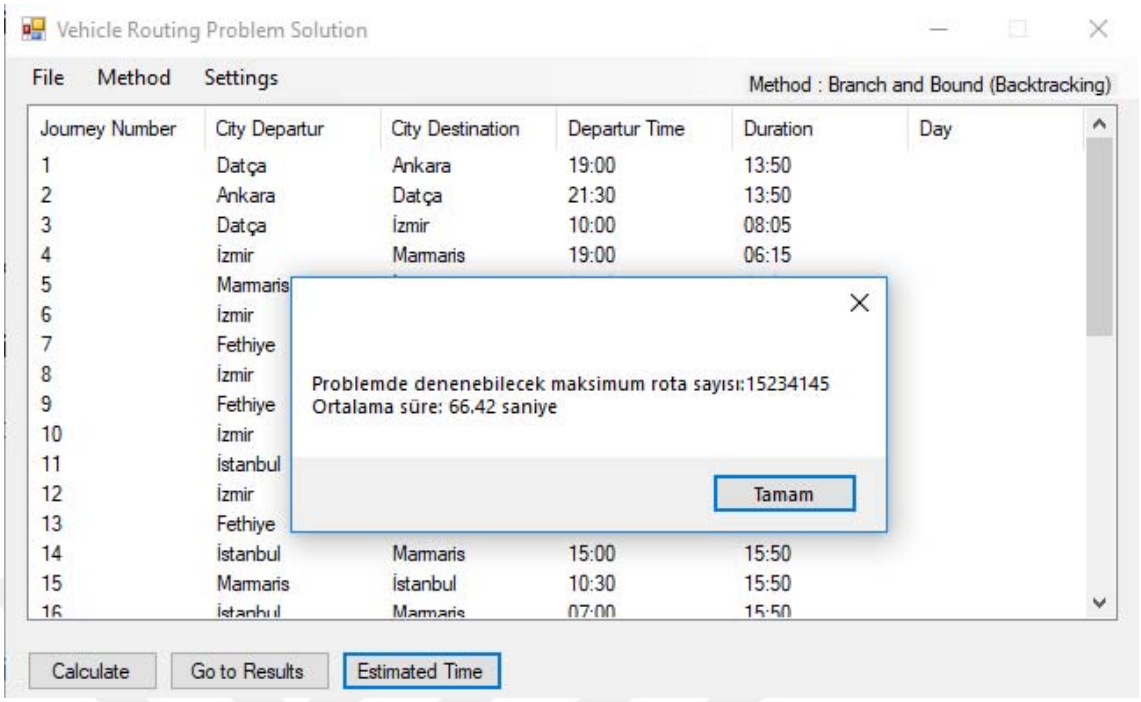
Yukarıda yer alan Şekil 5.7'deki gibi *Go to Result* butonuna basıldığında ise sonuçlarla ilgili *Result.txt* metin belgesinde yer alan sonuçlar açılmaktadır. En son bulunan sonuç bu metin belgesine eklenmektedir. Sonuçlar aşağıda yer alan şekil 18'deki gibi görünmektedir.



Şekil 5.7 Result.txt Ekran Görünümü

Her sonuç bulunduğunda önceki sonuç silinerek *Result.txt* dosyası değiştirilerek yenisi eklenmektedir.

Estimated Time butonuna bastığımızda ise problemin tahmini olarak ne kadar zamanda çözüleceği hesaplanıp gösterilmektedir. Bu bilgiler aşağıda yer alan şekil 5.8'deki gibi mesaj kutusunda gösterilmektedir. Buradaki tahmini süre bütün rotalar denendiğinde ortaya çıkabilecek en fazla süreyi göstermektedir. Gerçekte Geri İzleme Yaklaşımı sayesinde çoğu alternatif rota kötü çözüm üreteceğinden yarı yolda kesilip elenecektir. Böylece problemin çözümü tahmin edilen süreden daha kısa sürede bulunacaktır.



Şekil 5.8 Estimated Time Butonu Ekran Görünümü

4.3.7. Program Dosyaları ve Sistem Gereksinimleri

Programın çalıştırıldığı bilgisayar konfigürasyonu aşağıdaki gibidir;

İşlemci : Intel Core i5 6200U 2.3GHz

Bellek : 4 GB

İşletim Sistemi : Windows 10 (64 bit)

Grafik Kartı : NVIDIA GeForce 940M

Program dosyaları aşağıda yer alan Tablo 4.7'deki gibidir.

Dosya Adı	**Açıklama**
VehicleRoutingProblemSolution.exe	Programın çalıştırıldığı exe dosyası
Veriler.accdb	Microsoft Access 2010 veri tabanı dosyası
Settings.txt	Veri tabanı dosyasının geçerli konumunun ayarlandığı metin belgesi dosyası
Result.txt	Sonuçların görüntülendiği metin belgesi dosyası

Tablo 4.7 Yazılıma Ait Dosyalar

Dosya büyüklükleri aşağıda yer alan Tablo 4.8'deki gibidir.

Dosya Adı	**Dosya Büyüklüğü**
VehicleRoutingProblemSolution.exe	30 KB
Veriler.accdb	1.392 KB (Veri miktarı artarsa diskte kapladığı alan bir miktar daha artabilir)
Settings.txt	1 KB
Result.txt	1 KB

Tablo 4.8 Yazılıma Ait Dosya Büyüklükleri

Programın bellekte kapladığı yer 8.2 MB civarıdır. İşlem yaparken 9.5 MB civarına çıkabilmektedir.

Program işlem yaparken işlemci performansının yaklaşık %30'unu kullanmaktadır.

Yukarıdaki konfigürasyona sahip bir bilgisayarda program her saniyede yaklaşık 230.000 adımı kontrol etmektedir.

SONUÇ

Araç Rotalama Problemleri genellikle mesafeyi enküçüklemeye yöneliktir. Bu çalışmada; mesafe yerine toplam aylak (boşta bekleme) zamanı enküçüklemeye yönelik bir problemin, optimali garanti eden bir yöntemle çözümü bulunmuştur. Literatürde henüz yeni tanımlanmış bir problem olan aylak zamanı en küçüklemeye yönelik Araç Rotalama Probleminin Geri İzleme yaklaşımı ile çözümüne yönelik yapılan bir çalışma yoktur. Çalışma bu eksiği gidermiştir.

Program çalıştırıldıktan sonra 24 düğüm için hesaplamalar 1 dakika kadar sürmüştür. Düğüm sayısı çok büyük seçilmemiştir. Toplam aylak zaman 5715 dakika, en küçük aylak zamana ait rota; 1→ 2 → 3 → 20 → 21 → 10 → 11 → 12 → 13 → 14 → 15 → 16 → 17 → 18 → 19 → 22→ 23 → 4 → 5 → 6 → 7 → 8 → 9 → 24 olarak bulunmuştur. Bulunan sonuç kesin çözüm yöntemlerinden Geri İzleme Yaklaşımı ile çözüldüğünden bir en iyi (optimal) çözümdür.

Düğüm sayısı çok fazla olursa işlem süresi katlanarak (üssel) artacağından makul bir sürede sonuç bulunamayabilecektir. Bu durumda sezgisel ya da metasezgisel algoritmalar kullanılması gerekebilirdi. Ancak optimale yakın bir çözüm bulunabiliyor olsa hatta optimal çözüm dahi bulunmuş olsa bunun optimal olduğundan kesin olarak emin olunamayacaktır.

Problemin çözüm süresi yukarda da bahsedildiği gibi düğüm sayısı ve her düğümden gidilebilecek alternatif düğümler ile ilgilidir. Her bir düğüm için her düğümden gidilebilecek alternatif düğüm sayısının faktöriyeli kadar sıralama yapılabilmektedir. Problemin tümü için ise her düğümden gidilebilecek yerlerin faktöriyellerinin çarpımı kadar farklı rota denemeleri anlamına gelmektedir.

Bunu formüle edersek;

Her düğümden gidilebilecek düğüm sayısı di olsun (d1, d2, d3,…)

Bütün çözüm denemelerinin sayısı : ((d1!).(d2!).(d3!). …)/2 şeklinde hesaplanır.

Problemde 24 sefer yer almaktadır. Problemin çözümünde her bir sefer için yer adları (düğümler) aynı olsa dahi zamana bağlı farklı düğümler şeklinde düşünmek gerekir. Şehir sayısı ise 9 tanedir. Her bir şehirden gidilebilecek şehirlerin sayısı ise;

1.Şehir: 2, 2. Şehir: 1, 3. Şehir: 8, 4. Şehir: 3, 5. Şehir: 5, 6. Şehir: 4, 7. Şehir: 1, şeklindedir.

Toplam alternatif rota sayısı ise;

Toplam Rota Sayısı = (2!)(1!)(8!)(3!)(5!)(4!)(1!) / 2 =116.121.600. dır.

Problem için tüm dalları içeren bir tam sayımlama yapılsaydı çözüm süresinin 1 dakika yerine çok daha uzun olması gerekirdi. çözümü 1 dakika sürmüştür. Tam sayım yapılsaydı çözüm süresinin çok daha uzun olması gerekirdi. Geri İzleme yaklaşımı kullanıldığından sayımlama büyük oranda azalmıştır. Daha fazla düğümün olduğu problemlerde, verilerin değerlerine de bağlı olarak, tam sayım ile Geri İzleme Yaklaşımı süreleri arasındaki fark daha da açılabilecektir.

Lojistik ile ilgili birçok firma açısından bakıldığında araç rotalama problemleri ile boşta bekleme zamanını (aylak zamanı) kısaltmak önemli görünmektedir. Çünkü aylak zamanın kısaltılması yoluyla daha az işgücü (personel) daha az araç ile aynı işin yapılması sağlanıp aynı zamanda araç sayısı azaldığından yakıt tasarrufu, bakım, onarım gibi konularda da azalma olacağından çok daha yüksek tasarruflar elde edilebilir. Diğer bir deyişle eldeki personel araç-gereç daha verimli kullanılmış olur.

Geliştirilen çözüm yöntemi birçok farklı araç rotalama problemi türünde kullanılabilir. Havayolu şirketlerinden deniz yolu şirketlerine kadar birçok alanda kullanım alanı bulabilir. Bunların dışında çözüm yöntemi geliştirilerek atama problemleri, çizelgeleme problemleri gibi birçok farklı problem türlerinin çözümü için de kullanılabilir.

KAYNAKÇA

ATASEVEN Burçin, "Yapay Sinir Ağları ile Öngörü Modellemesi", *Sosyal Bilimler Enstitüsü Makale Koleksiyonu*, 2013, C. 10, S. 39, ss.101-115.

ATMACA Ediz, "Bir Kargo Şirketinde Araç Rotalama Problemi ve Uygulaması", 2012, *Tübav Bilim Dergisi*, C. 5, S. 2, ss.12-27.

BARKAOUİ M., BERGER J., BOUKHTOUTA A. (2015), "Customer Satisfaction in Dynamic Vehicle Routing Problemwith Time Windows", Applied Soft Computing, Vol 35, ss.423-432.

BELFİORE Patrícia, YOSHİZAKİ Hugo T.Y., "Heuristic Methods for the Fleet Size and Mix Vehicle Routing Problem with Time Windows and Split Deliveries", *Computers & Industrial Engineering*, 2013, C. 64, ss.589-601.

BELL John E., MCMULLEN Patrick R., "Ant Colony Optimization Techniques for the Vehicle Routing Problem", *Advanced Engineering Informatics*, 2004, C. 18, ss.41-48.

BELL John E., MULLEN Patrick R., "Ant colony optimization techniques for the vehicle routing problem", *Advanced Engineering Informatics*, 2004, C. 18, ss. 41-48.

BORİSENKO Andrey, HAİDL Michael, GORLATCH Sergei, "A GPU Parallelization of Branch-and-Bound for Multiproduct Batch Plants Optimization", *Springer Science*, 2017, C. 73, ss.639-651.

BOZYER Zafer, ALKAN Atakan, FIĞLALI Alpaslan, "Kapasite Kısıtlı Araç Rotalama Probleminin Çözümü için Önce Grupla Sonra Rotala Merkezli Sezgisel Algoritma Önerisi", *Bilişim Teknolojileri Dergisi*, 2014, C. 7, S. 2, ss.29-37.

BRUSCO Michael J., STAHL Stephanie, "Statistics and Computing", "Branch and Bound Applications in Combinatorial Data Analysis", *Springer Science*, 2005, ss.4-8.

BULLNHEİMER Bernd, HART Richard F., STRAUSS Christine, "Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization", "Applying the Ant System to the Vehicle Routing Problem", *Kluwer Academic Publishers*, 1999, C. 20, ss.285-286.

CARIC Tonci, CALIC Ante, FOSIN Juraj, GOLD Hrvoje, REINHOLZ Andreas, "A Modelling and Optimization Framework for Real-World Vehicle Routing Problems", *Vehicle Routing Problem*, Intechopen, 2008

ÇALIŞKAN Kamil, "*Karınca Kolonisi Optimizasyonu ile Araç Rotalama Probleminin Maliyetlerinin Kümeleme Tekniği ile İyileştirilmesi*", (Yüksek Lisans Tezi), Ankara: TOBB Ekonomi Ve Teknoloji Üniversitesi Fen Bilimleri Enstitüsü, 2011.

ÇAM Ömer Nuri, SEZEN H. Kemal, "Toplam Bekleme Süresini Enküçükleme Amaçlı Bir Araç Rotalama Problemi", *International Journal of Social Inquiry*, 2018, C. 11, S. 2, ss.47-60.

ÇETİN Mustafa, "Gezgin Satıcı Örnek Problemlerinin Optimum Sonuçlarının Grid Aracılığı İle Hesaplanması", (Yüksek Lisans Tezi), Balıkesir: Balıkesir Üniversitesi Fen Bilimleri Enstitüsü, 2007.

ÇETİN Suna, GENCER Cevriye, "Heterojen Araç Filolu Zaman Pencereli Eş Zamanlı Dağıtım-Toplamalı Araç Rotalama Problemleri: Matematiksel Model", *International Journal of Research and Development*, 2011, C. 3, S. 1, ss.19-27.

ÇOLAK Selçuk, "Genetik Algoritmalar Yardımı ile Gezgin Satıcı Probleminin Çözümü Üzerine Bir Uygulama", *Ç.Ü. Sosyal Bilimler Enstitüsü Dergisi*, 2010, C. 19, S. 3, ss.423-438.

DANTZİG G. B., RAMSER J. H., "The Truck Dispatching Problem", *Management Science*, 1959, C. 6, S. 2, ss. 80-91.

DASTGHAİBİFARD G.H., ANSARİ E., SHEYKHALİSHAHİ S.M., BAVANDPOURİ A., ASHOOR E., "A Parallel Branch and Bound Algorithm for Vehicle Routing Problem", *Proceedings of the International MultiConference of Engineers and Computer Scientists*, 2008, C. 2, ss.1-6.

DEMİRCİOĞLU Mert, "*Araç Rotalama Probleminin Sezgisel Bir Yaklaşım ile Çözümlenmesi Üzerine Bir Uygulama*", (Doktora Tezi), Adana: Çukurova Üniversitesi Sosyal Bilimler Enstitüsü, 2009.

DİKMEN Hasan, DİKMEN Hüseyin, ELBİR Ahmet, EKŞİ Ziya, ÇELİK Fatih, "Gezgin Satıcı Probleminin Karınca Kolonisi ve Genetik Algoritmalarla Eniyilemesi ve Karşılaştırılması", *Fen Bilimleri Enstitüsü Dergisi*, 2014, C.18, S. 1, ss.8-13.

DÜZAKIN Erkut, DEMİRCİOĞLU Mert, "Araç Rotalama Problemleri ve Çözüm Yöntemleri", *Çukurova Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, 2009, C. 13, S. 1, ss.66-87.

EMEL Gül Gökay, TAŞKIN Çağatan, "Genetik Algoritmalar ve Uygulama Alanları", *İktisadi ve İdari Bilimler Fakültesi Dergisi*, 2002, C. 21, S. 1, ss.129-152.

EREN Tamer, GÜNER Ertan, "Çok Ölçütlü Akış Tipi Çizelgeleme Problemleri İçin Bir Literatür Taraması", *Mühendislik Bilimleri Dergisi*, 2004, C. 10 S. 1, ss.19-30.

ERYAVUZ Mehmet, GENCER Cevriye, "Araç Rotalama Problemine Ait Bir Uygulama", *Süleyman Demirel Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, 2001, C. 6, S. 1, ss.139-155.

GÖKAY EMEL Gül, TAŞKIN Çağatan, "Araç Rotalama Problemlerinin İki Aşamalı Çözümünde Genetik Algoritma Kullanımı", *Gazi Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, 2005, C. 7, S. 1, ss.1-17.

HOKAMA Pedro, MIYAZAWA Flávio K., XAVIER EduardoC., "Abranch and cut approach for the vehicle routing problem with loading constraints*", Expert SystemsWithApplications*, 2016, C. 47, ss. 1-13.

HOOSHMAND F. Khaligh, MİRHASSANİ S.A., "A mathematical model for vehicle routing problem under endogenous uncertainty", *International Journal of Production Research*, 2016, C. 54, S. 2, ss.579-590.

KADRİ Ahmed Abdelmoumene, KACEM Imed, LABADİ Karim, "A Branch-and-Bound Algorithm for Solving the Static Rebalancing Problem in Bicycle-Sharing Systems", 2016, *Computers & Industrial Engineering*, C. 95, ss.45-52.

KEÇECİ Barış, ALTIPARMAK Fulya, KARA İmdat, "Heterojen Eş-Zamanlı Topla-Dağıt Araç Rotalama Problemi: Matematiksel Modeller ve Sezgisel Bir Algoritma", *Journal of the Faculty of Engineering and Architecture of Gazi University*, 2015, C.30, S.2, ss.185-195.

KESKİNTÜRK Timur, TOPUK Nihan, ÖZYEŞİL Okan, "Araç Rotalama Problemleri ile Çözüm Yöntemlerinin Sınıflandırılması ve Bir Uygulama", *İşletme Bilimi Dergisi*, 2015, C. 3, S. 2, ss.77-107.

KESKİNTÜRK Timur, UZ Emin, TOPA Mehmet, "Kapasite ve Mesafe Kısıtlı Periyodik Gezgin Satıcı Problemi ve Genetik Algoritma ile Çözümü: Türk Hava Kuvvetlerine Ait Kargo Uçaklarının", *Akademik Yaklaşımlar Dergisi*, 2016, C. 7, S. 1, ss.53-68.

KİREMİTCİ Barış, "Zaman Pencereli Çok Araçlı Dağıtım Toplamalı Rotalama Problemi İçin Gerçek Değerli Genetik Algoritma Yaklaşımı", *Istanbul University Journal of the School of Business*, 2014, C. 43, S. 2, ss.391-40.

KOÇ Çağrı, KARAOĞLAN İsmail, "Zaman Bağımlı Araç Rotalama Problemi İçin Bir Matematiksel Model", *Journal of the Faculty of Engineering and Architecture of Gazi University*, 2014, C. 29, S.3, ss.549-558.

KOSİF Burak, EKMEKÇİ İsmail, "Araç Rotalama Sistemleri v e Tasarruf Algoritması Uygulaması", *İstanbul Ticaret Üniversitesi Fen Bilimleri Dergisi*, 2012, S. 21, ss.41-45.

KUZU Sultan, ÖNAY Onur, ŞEN Uğur, TUNÇER Mustafa, YILDIRIM Bahadır Fatih, KESKİNTÜRK Timur, "Gezgin Satıcı Problemlerinin Metasezgiseller ile Çözümü", *İstanbul Üniversitesi İşletme Fakültesi Dergisi*, 2014, C. 43, S. 1, ss.1-27.

LAPORTE G., NOBERT Y., "A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem", *OR Spektrum*, 1983, C. 5, ss. 77-85.

LAPORTE Gilbert, "The Vehicle Routing Problem: An overview of exact and approximate algorithms", *European Journal of Operational Research*, 1992, C. 59, ss.345-358.

LİU S. B., NG K. M., ONG H. L., "Branch-And-Bound Algorithms For Simple Assembly Line Balancing Problem", *International Journal of Advanced Manufacturing Technology*, 2008, C. 36, ss.169-177.

McKEOWN G.P., RAYWARD-SMITH V.J., TURPIN H.J., "Branch-And-Bound as a Higher-Order Function", *Annals of Operations Research*, 1991, C. 33, ss.379-402.

MONTOYA Alejandro, GUÉRET Christelle, MENDOZA Jorge E., VİLLEGAS Juan G., "A Multi-Space Sampling Heuristic for the Green Vehicle Routing Problem", *Transportation Research Part C*, 2016, C. 70, ss.113-128.

PAKKAN Bahar, Ermiş MURAT, "İnsansız Hava Araçlarının Genetik Algoritma Yöntemiyle Çoklu Hedeflere Planlanması", *Havacılık ve Uzay Teknolojileri Dergisi*, 2010, C.4 S.3, ss.77-84.

PATIR Sait, "Dinamik Programlama ve Bir Ecza Deposunun Şehir İçi İlaç Dağıtımına Alternatifli Bir Çözüm Önerisi", *Atatürk Üniversitesi İktisadi ve İdari Bilimler Dergisi*, 2009, C. 23, S. 2, ss.63-79.

RAZALİA Noraini Mohd, "An Efficient Genetic Algorithm for Large Scale Vehicle Routing Problem Subject to Precedence Constraints", *Procedia - Social and Behavioral Sciences*, 2015, C.195, ss.1922-1931.

SEYFİ Gökhan, "*Metasezgisel Algoritmalar Kullanılarak Sınav Çizelgeleme*", (Yüksek Lisans Tezi), Konya: Selçuk Üniversitesi Fen Bilimleri Enstitüsü, 2018.

SEZEN H.Kemal, "*Yöneylem Araştırması*", Bursa: Dora Yayınevi, 2017.

SONG Byung Duk, KO Young Dae, "A vehicle routing problem of both refrigerated- and general-type vehicles for perishable food products delivery", *Journal of Food Engineering*, 2016, C. 169, ss.61-71.

ŞAHİN Yusuf, EROĞLU Abdullah, "Kapasġte Kısıtlı Araç Rotalama Problemi İçin Metasezgisel Yöntemler: Bilimsel Yazın Taraması", *Süleyman Demirel Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi,* 2014, C. 19 S. 4, ss.337-355.

TEKTAŞ Mehmet, AKBAŞ Ahmet, TOPUZ Vedat, "Yapay Zeka Tekniklerinin Trafik Kontrolünde Kullanılması Üzerine Bir İnceleme", *Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi*, 2006, C. 22, S. 4, ss.753-758.

TEZER Tuba, "*Toplama ve Dağıtım Zaman Pencereli Araç Rotalama Problemi İçin Kesin Çözüm Yaklaşımı ve Örnek Uygulamalar*", (Yüksek Lisans Tezi), Balıkesir: Balıkesir Üniversitesi Fen Bilimleri Enstitüs, 2009.

WOODCOCK Andrew J., WİLSON John M., "A Hybrid Tabu Search/Branch & Bound Approach to Solving the Generalized Assignment Problem", *European Journal of Operational Research*, 2010, C. 207, ss.566-578.

YILDIRIM Tevfik, "*Simetrik Gezgin Satıcı Problemi İçin Yeni Bir Meta-Sezgisel: Kör Fare Algoritması*", (Yüksek Lisans Tezi), Denizli: Pamukkale Üniversitesi Fen Bilimleri Enstitüsü, 2014.

EKLER

Ek.1: Yazılımın Kaynak Kodları

Program altında yer alan kaynak kodu dosyaları şunlardır;

- AssemblyInfo.cs
- Resources.Designer.cs
- Settings.Designer.cs
- Program.cs
- Form1.cs
- Form2.cs
- Form3.cs

AssemblyInfo.cs dosyası için kaynak kodları:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;
// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Vehicle Routing Problem Solution")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Vehicle Routing Problem Solution")]
[assembly: AssemblyCopyright("Copyright ©  2017")]
```

```
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]
// The following GUID is for the ID of the typelib if this project is exposed to COM
[assembly: Guid("32a13719-8947-4209-9917-cac87f07cc9b")]
// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Resources.Designer.cs dosyası için kaynak kodları:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
```

```
// </auto-generated>
//------------------------------------------------------------------------------

namespace Vehicle_Routing_Problem_Solution.Properties
{
    /// <summary>
    ///   A strongly-typed resource class, for looking up localized strings, etc.
    /// </summary>
    // This class was auto-generated by the StronglyTypedResourceBuilder
    // class via a tool like ResGen or Visual Studio.
    // To add or remove a member, edit your .ResX file then rerun ResGen
    // with the /str option, or rebuild your VS project.
    [global::System.CodeDom.Compiler.GeneratedCodeAttribute("System.Resources.Tools
.StronglyTypedResourceBuilder", "4.0.0.0")]
    [global::System.Diagnostics.DebuggerNonUserCodeAttribute()]
    [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
    internal class Resources
    {
        private static global::System.Resources.ResourceManager resourceMan;
        private static global::System.Globalization.CultureInfo resourceCulture;
[global::System.Diagnostics.CodeAnalysis.SuppressMessageAttribute("Microsoft.Perfo
rmance", "CA1811:AvoidUncalledPrivateCode")]
        internal Resources()
        {
        }
        /// <summary>
        ///   Returns the cached ResourceManager instance used by this class.
        ///</summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.Compone
ntModel.EditorBrowsableState.Advanced)]
        internal static global::System.Resources.ResourceManager ResourceManager
        {
```

```
            get
            {
                if ((resourceMan == null))
                {
                    global::System.Resources.ResourceManager temp = new global::System.Resources.ResourceManager("Vehicle_Routing_Problem_Solution.Properties.Resources", typeof(Resources).Assembly);
                    resourceMan = temp;
                }
                return resourceMan;
            }
        }
        /// <summary>
        ///   Overrides the current thread's CurrentUICulture property for all
        ///   resource lookups using this strongly typed resource class.
        ///</summary>
[global::System.ComponentModel.EditorBrowsableAttribute(global::System.ComponentModel.EditorBrowsableState.Advanced)]
        internal static global::System.Globalization.CultureInfo Culture
        {
            get
            {
                return resourceCulture;
            }
            set
            {
                resourceCulture = value;
            }
        }
    }
```

```
}
```

Settings.Designer.cs için kaynak kodları:

```
//------------------------------------------------------------------------------
// <auto-generated>
//     This code was generated by a tool.
//     Runtime Version:4.0.30319.42000
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </auto-generated>
//------------------------------------------------------------------------------

namespace Vehicle_Routing_Problem_Solution.Properties
{
   [global::System.Runtime.CompilerServices.CompilerGeneratedAttribute()]
[global::System.CodeDom.Compiler.GeneratedCodeAttribute("Microsoft.VisualStudio.Editors.SettingsDesigner.SettingsSingleFileGenerator", "11.0.0.0")]
   internal sealed partial class Settings : global::System.Configuration.ApplicationSettingsBase
   {
      private static Settings defaultInstance = ((Settings)(global::System.Configuration.ApplicationSettingsBase.Synchronized(new Settings())));
      public static Settings Default
      {
         get
         {
            return defaultInstance;
         }
      }
   }
```

```
}
```

Program.cs dosyası için kaynak kodları:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace Vehicle_Routing_Problem_Solution
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

From1.cs dosyası için kaynak kodları:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
```

```
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.OleDb;
using System.IO;
namespace Vehicle_Routing_Problem_Solution
{
   public partial class Form1 : Form
   {
      public static string databaseAddress = "";
      public static int method = 0;
      string[] journeyNumber;
      string[] cityDepartur;
      string[] cityDestination;
      string[] departurTime;
      string[] duration;
      string[] day;
      string result = "",
            route  = "";
      public Form1()
      {
         InitializeComponent();
      }
      private void exitToolStripMenuItem_Click(object sender, EventArgs e)
      {
         Close();
```

```
}
private void openToolStripMenuItem_Click(object sender, EventArgs e)
{
   try
   {
      openFileDialog1.ShowDialog();
      listView1.Columns.Add("Journey Number", 100);
      listView1.Columns.Add("City Departur", 100);
      listView1.Columns.Add("City Destination", 100);
      listView1.Columns.Add("Departur Time", 100);
      listView1.Columns.Add("Duration", 100);
      listView1.Columns.Add("Day", 100);
      listView1.View = View.Details;
      databaseAddress = openFileDialog1.FileName.ToString();
      string source = @"Provider=Microsoft.ACE.Oledb.12.0;Data Source=" +
openFileDialog1.FileName.ToString();
      OleDbConnection databaseConnection = new OleDbConnection(source);
      string sqlCommand = "SELECT * FROM journey";
      databaseConnection.Open();
      OleDbCommand command = new OleDbCommand(sqlCommand,
databaseConnection);
      OleDbDataReader print = command.ExecuteReader();
      while (print.Read())
      {
         string[] data = new string[6];
         for (int i = 0; i < 6; i++)
         {
            data[i] = print[i + 1].ToString();
         }
```

```
            ListViewItem add = new ListViewItem(data);
            listView1.Items.Add(add);
        }
        databaseConnection.Close();
    }
    catch { }
}
private void setDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form2 dataAdjustment = new Form2();
    dataAdjustment.Show();
}
private void dataSettingsToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form3 dataSettings = new Form3();
    dataSettings.Show();
}
private void Form1_Load(object sender, EventArgs e)
{
    listViewLoad();
}
private void backtrackingToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Text = "Method : Branch and Bound (Backtracking)";
    this.label1.Location = new Point(633 - label1.Size.Width, 9);
    method = 1;
}
private void antColonyToolStripMenuItem1_Click(object sender, EventArgs e)
```

```
{
    label1.Text = "Method : Heuristic (Ant Colony)";
    this.label1.Location = new Point(633 - label1.Size.Width, 9);
    method = 2;
}
private void geneticToolStripMenuItem_Click(object sender, EventArgs e)
{
    label1.Text = "Method : Heuristic (Genetic)";
    this.label1.Location = new Point(633 - label1.Size.Width, 9);
    method = 3;
}
private void Form1_Activated(object sender, EventArgs e)
{
    listViewLoad();
}
private void button1_Click(object sender, EventArgs e)
{
    journeyNumber   = new string[listView1.Items.Count];
    cityDepartur    = new string[listView1.Items.Count];
    cityDestination = new string[listView1.Items.Count];
    departurTime    = new string[listView1.Items.Count];
    duration        = new string[listView1.Items.Count];
    day             = new string[listView1.Items.Count];
    if (button1.Text == "Calculate")
    {
        timer1.Enabled = true;
        button1.Text = "Stop";
    }
```

```
        else
        {
            timer1.Enabled = false;
            button1.Text = "Calculate";
            MessageBox.Show(result);
            return;
        }

        for (int i = 0; i < listView1.Items.Count; i++)
        {
            journeyNumber[i]   = listView1.Items[i].SubItems[0].Text;
            cityDepartur[i]    = listView1.Items[i].SubItems[1].Text;
            cityDestination[i] = listView1.Items[i].SubItems[2].Text;
            departurTime[i]    = listView1.Items[i].SubItems[3].Text;
            duration[i]        = listView1.Items[i].SubItems[4].Text;
        }
        switch (method)
        {
            case 0:
MessageBox.Show(VRP.exactSolution.branchAndBound.backTracking(journeyNumbe
r, cityDepartur, cityDestination,
                departurTime, duration, day, ref result, ref route));
                updateData();
                break;
        }
    }
    void listViewLoad()
    {
        listView1.Clear();
```

```
            listView1.Columns.Add("Journey Number", 100);
            listView1.Columns.Add("City Departur", 100);
            listView1.Columns.Add("City Destination", 100);
            listView1.Columns.Add("Departur Time", 100);
            listView1.Columns.Add("Duration", 100);
            listView1.Columns.Add("Day", 100);
            listView1.View = View.Details;
            StreamReader read = new StreamReader("settings.txt");
            Form1.databaseAddress = read.ReadLine();
            read.Close();
            try
            {
               string source = @"Provider=Microsoft.ACE.Oledb.12.0;Data Source=" +
Form1.databaseAddress;
               OleDbConnection databaseConnection = new OleDbConnection(source);
               string sqlCommand = "SELECT * FROM journey";
               databaseConnection.Open();
               OleDbCommand command = new OleDbCommand(sqlCommand,
databaseConnection);
               OleDbDataReader print = command.ExecuteReader();
               while (print.Read())
               {
                  string[] data = new string[6];
                  for (int i = 0; i < 6; i++)
                  {
                     data[i] = print[i + 1].ToString();
                  }
                  ListViewItem add = new ListViewItem(data);
                  listView1.Items.Add(add);
```

```
            }
            databaseConnection.Close();
        }
        catch { }
    }
    void updateData()
    {
        string source = @"Provider=Microsoft.ACE.Oledb.12.0;Data Source=" + Form1.databaseAddress;
        OleDbConnection databaseConnection = new OleDbConnection(source);
        databaseConnection.Open();
        for (int i = 0; i < listView1.Items.Count; i++)
        {
            string sqlCommand = "update journey set journey_number='" + journeyNumber[i] + "', city_departur='" +
                cityDepartur[i] + "', city_destination='" + cityDestination[i] + "', departur_time='" +
                departurTime[i] + "', duration='" + duration[i] + "', day_of_journey='" + day[i] +
                "' where journey_number='" + journeyNumber[i] + "'";
            OleDbCommand command = new OleDbCommand(sqlCommand, databaseConnection);
            command.ExecuteNonQuery();
        }
        databaseConnection.Close();
        listViewLoad();
    }
    private void timer1_Tick(object sender, EventArgs e)
    {
        label2.Text = route;
```

```
        }
        private void button3_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Problemde denenebilecek maksimum rota sayısı:" +
"15234145\r\n" + "Ortalama süre: 66.42 saniye");
        }
    }
    public class VRP
    {
        public class exactSolution
        {
            public class branchAndBound
            {
                public static string backTracking(string[] journeyNumber, string[]
cityDepartur, string[] cityDestination,
                    string[] departurTime, string[] duration, string[] day, ref string result, ref
string route)
                {
                    int cityNumber = cityCount(cityDepartur, cityDestination);
                    string[] cluster = new string[cityNumber];
                    string[] cityList = new string[cityNumber];
                    string solution = "",
                           bestSolution = "",
                           chainIndex = "",
                           forbiddenCityIndex = "";
                          double slackTime = 0.0,
                           bestSlackTime = 0.0;
                    int clusterIndex = 0,
                       nextCityIndex = 0,
```

```
            estimatedTime = 0,
            tourCount = journeyNumber.Length,
            tourcount2 = journeyNumber.Length,
            tourNumber = 0;
        int time;

        string timePrevieus = DateTime.Now.ToString(), timeNext = "";
        createCityList(cityDepartur, cityDestination, cityList);
        createCluster(cluster, cityList, journeyNumber, cityDepartur,
cityDestination, departurTime, duration);
        sortCluster(cluster);
        time = -1;
        //Form1 frm1 = new Form1();
        //time = getTime(cluster[0].Substring(7, 5));
        //cluster[0] = "002700119:0013:50001600210:0008:05003";
        //MessageBox.Show(findBestSlackTime(cluster[0], "4", ref
time).ToString());
        while (!(clusterIndex == 0 && nextCityIndex == -1 && tourCount ==
journeyNumber.Length))
        {
            /*MessageBox.Show("cluster[0]" + cluster[0] + "\r\n" + "cluster[1]" +
cluster[1] + "\r\n" + "cluster[2]" + cluster[2] + "\r\n" + "cluster[3]" + cluster[3] + "\r\n"
+ "cluster[4]" + cluster[4] + "\r\n" + "cluster[5]" + cluster[5] + "\r\n" + "cluster[6]" +
cluster[6] + "\r\n" + "cluster[7]" + cluster[7] + "\r\n" + "cluster[8]" + cluster[8] +
                "\r\n" + "\r\n"+ "Cluster Index : " + clusterIndex.ToString() + "\r\n" +
                "forbidden City Index : "+ forbiddenCityIndex.ToString()+"\r\n" +
                "\r\n" + "Solution : " + solution.ToString() + "\r\n" +
                "Journey Number : " + tourCount.ToString() + "\r\n" +
                "Time : " + time.ToString() + "\r\n" +
                "Slack Time : " + slackTime.ToString() + "\r\n" +
                "Best Solution : " + bestSolution + "\r\n" +
```

```
                "Best Slack Time : " + bestSlackTime.ToString() + "\r\n" +
                "Chain Index : " + chainIndex.ToString());*/
            Application.DoEvents();
            route = solution;
            /*
            frm1.label2.Text = solution;
            frm1.Show();
            frm1.label2.Refresh();*/
            estimatedTime++;
            nextCityIndex = moveForward(cluster, ref clusterIndex, ref forbiddenCityIndex, ref solution, ref chainIndex, ref slackTime, ref tourCount, ref time, ref tourcount2);
            if (nextCityIndex == -1)
            {
                moveBackward(cluster, ref solution, ref slackTime, ref forbiddenCityIndex, ref tourCount, ref clusterIndex, ref chainIndex);
                continue;
            }
            if (slackTime > bestSlackTime && bestSlackTime != 0)
            {
                moveBackward(cluster, ref solution, ref slackTime, ref forbiddenCityIndex, ref tourCount, ref clusterIndex, ref chainIndex);
                continue;
            }
            if (tourCount == 0)
            {
                bestSolution = solution;
                bestSlackTime = slackTime;
                moveBackward(cluster, ref solution, ref slackTime, ref forbiddenCityIndex, ref tourCount, ref clusterIndex, ref chainIndex);
```

```
                    tourNumber++;
                    timeNext = DateTime.Now.ToString();
                    result = "En iyi çözüm : " + bestSolution + "\r\n" +
                                "Aylak zaman : " + bestSlackTime.ToString() + "\r\n" +
                                "İşlem sayısı : " + estimatedTime.ToString() + "\r\n" +
                                "İşlem zamanı = Başlangıç: " + timePrevieus + " , Bitiş: "
+ (timeNext);
                }
            }
            return bestSolution + "#" + bestSlackTime.ToString();
        }
    }
}

    private static void createCluster(string[] cluster, string[] cityList, string[]
journeyNumber,
        string[] cityDepartur, string[] cityDestination, string[] departurTime, string[]
duration)
    {
        int k = 0;
        int clusterCount = 0;
        for (int i = 0; i < cluster.Length; i++)
        {
            k = 0;
            clusterCount = 0;
            while (k < cityDepartur.Length)
            {
                if (cityList[i] == cityDepartur[k])
                {
```

```
                cluster[i] = cluster[i] + "0" + findCityIndex(cityList, cityDestination[k])
+                   departurTime[k] + duration[k] + indexFormat(journeyNumber[k], 3);
                clusterCount++;
            }
            k++;
        }
        cluster[i] = indexFormat(clusterCount.ToString(), 3) + cluster[i];
    }
}
private static void sortCluster(string[] cluster)
{
    string swap = "";
    for (int i = 0; i < cluster.Length; i++)
    {
        int subClusterCount = (cluster[i].Length - 3) / 17;
        string[] subCluster = new string[subClusterCount];
        for (int a = 0; a < subClusterCount; a++)
        {
            subCluster[a] = cluster[i].Substring(3 + a * 17, 17);
        }
        for (int x = 0; x < subClusterCount - 1; x++)
        {
            for (int y = x + 1; y < subClusterCount; y++)
            {
                if (bigString(subCluster[x], subCluster[y]))
                {
                    swap = subCluster[x];
                    subCluster[x] = subCluster[y];
                    subCluster[y] = swap;
```

```
                }
            }
        }
        cluster[i] = cluster[i].Substring(0, 3);
        int k = 0;
        while (k < subClusterCount)
        {
            cluster[i] = cluster[i] + subCluster[k];
            k++;
        }
    }
}
private static void createCityList(string[] cityDepartur, string[] cityDestination, string[] cityList)
{
    int endOfArray = 1;
    cityList[0] = cityDepartur[0];
    for (int i = 1; i < cityDepartur.Length; i++)
    {
        int k = 0;
        while (k < endOfArray)
        {
            if (cityDepartur[i] != cityList[k])
            {
                k++;
            }
            else
            {
                break;
```

```
            }
        }
        if (k == endOfArray)
        {
            cityList[k] = cityDepartur[i];
            endOfArray++;
        }
    }
}
private static int cityCount(string[] cityDepartur, string[] cityDestination)
{
    int endOfArray = 1;
    string[] cityList = new string[cityDepartur.Length];
    cityList[0] = cityDepartur[0];
    for (int i = 1; i < cityDepartur.Length; i++)
    {
        int k = 0;
        while (k < endOfArray)
        {
            if (cityDepartur[i] != cityList[k])
            {
                k++;
            }
            else
            {
                break;
            }
        }
```

```
        if (k == endOfArray)
        {
            cityList[k] = cityDepartur[i];
            endOfArray++;
        }
    }
    return endOfArray;
}
private static string findCityIndex(string[] cityList, string cityName)
{
    int i = 0;
    while (i < cityList.Length)
    {
        if (cityList[i] == cityName)
        {
            return indexFormat(i.ToString(), 3);
        }
        i++;
    }
    return indexFormat("?", 3);
}
private static int getNodeCount(string cluster)
{
    return Convert.ToInt32(cluster.Substring(0, 3));
}
private static string indexFormat(string index, int format)
{
    string newFormat = index;
```

```
        while (format > 0 && format > index.Length)
        {
            newFormat = "0" + newFormat;
            format--;
        }
        return newFormat;
    }
    private static int getTime(string time)
    {
        return Convert.ToInt32(time.Substring(0, 2)) * 60 + Convert.ToInt32(time.Substring(3, 2));
    }
    private static bool bigString(string str1, string str2)
    {
        int loopCount = 0;
        if (str1.Length > str2.Length)
        {
            loopCount = str2.Length;
        }
        else
        {
            loopCount = str1.Length;
        }
        for (int i = 0; i < loopCount; i++)
        {
            if((int)Convert.ToChar(str1.Substring(i, 1)) < (int)Convert.ToChar(str2.Substring(i, 1)))
            {
                return false;
```

```
            }
            if ((int)Convert.ToChar(str1.Substring(i, 1)) > (int)Convert.ToChar(str2.Substring(i, 1)))
            {
               return true;
            }
         }
         if (str1.Length < str2.Length)
         {
            return false;
         }
         else
         {
            return true;
         }
      }
      private static int moveForward(string[] cluster, ref int clusterIndex, ref string forbiddenCityIndex, ref string solution, ref string chainIndex, ref double slackTime, ref int tourCount, ref int time,  ref int tourCount2)
      {
         string index = findNonVisitedCity(cluster[clusterIndex], ref forbiddenCityIndex, ref time, ref tourCount, ref tourCount2);
         if (index == "-1" || index == "-2")
         {
            return moveBackward(cluster, ref solution, ref slackTime, ref forbiddenCityIndex, ref tourCount, ref clusterIndex, ref chainIndex);
         }
         if (index != "-1" && index != "-2")
         {
            if (forbiddenCityIndex != "")
```

```
            {
                releaseForbiddenCity(ref forbiddenCityIndex, cluster);
            }
            solution = solution + cluster[clusterIndex].Substring(Convert.ToInt32(index.Substring(0,4)) + 13,3) + ":";
            slackTime = slackTime + Convert.ToDouble(index.Substring(4, 4));
            chainIndex = chainIndex + indexFormat(clusterIndex.ToString(),3) + index;
            //time = refreshTime(cluster[clusterIndex], ref time, ref index);
            cluster[clusterIndex] = indexFormat((Convert.ToInt32(cluster[clusterIndex].Substring(0, 3)) - 1).ToString(), 3) + cluster[clusterIndex].Substring(3, cluster[clusterIndex].Length - 3);
            cluster[clusterIndex] = changeString(cluster[clusterIndex], "1", (Convert.ToInt32(index.Substring(0, 4)) -1).ToString());
            clusterIndex = Convert.ToInt32(cluster[clusterIndex].Substring(Convert.ToInt32(index.Substring(0, 4)),3));
            tourCount--;
            return clusterIndex;
        }
        else
        {
            return -1;
        }
    }
    private static int moveBackward(string[] cluster, ref string solution, ref double slackTime, ref string forbiddenCityIndex, ref int tourCount, ref int clusterIndex, ref string chainIndex)
    {
        if (chainIndex == "")
        {
            return -1;
        }
```

```
        if (forbiddenCityIndex != "")
        {
            releaseForbiddenCity(ref forbiddenCityIndex, cluster);
        }

        solution = solution.Substring(0, solution.Length - 4);
        slackTime = slackTime - Convert.ToDouble(chainIndex.Substring(chainIndex.Length - 4, 4));
        forbiddenCityIndex = chainIndex.Substring(chainIndex.Length - 11, 7);
        //////////
        // Cluster içindeki eleman sayısını bir artır
        if (chainIndex != "")
        {
            clusterIndex = Convert.ToInt32(chainIndex.Substring(chainIndex.Length - 11, 3));
        }
        else
        {
            clusterIndex = 0;
        }
        chainIndex = chainIndex.Substring(0, chainIndex.Length - 11);
        cluster[clusterIndex] = indexFormat((Convert.ToInt32(cluster[clusterIndex].Substring(0, 3)) + 1).ToString(), 3) + cluster[clusterIndex].Substring(3, cluster[clusterIndex].Length - 3);
        //time = (getTime(cluster[clusterIndex].Substring(Convert.ToInt32(subClusterIndex) + 3, 5)) + getTime(cluster.Substring(Convert.ToInt32(subClusterIndex) + 8, 5))) % 1440;
        tourCount++;
        return clusterIndex;
    }
```

```
    private static void releaseForbiddenCity(ref string forbiddenCityIndex, string[] cluster)
    {
        string part1 = "", part2 = "";
        int ClusterIndex;
        //MessageBox.Show(cluster[6]);
        ClusterIndex = Convert.ToInt32(forbiddenCityIndex.Substring(0, 3));
        part1 = cluster[ClusterIndex].Substring(0, Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)) - 1);
        part2 = cluster[ClusterIndex].Substring(Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)), cluster[ClusterIndex].Length - Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)));
        cluster[ClusterIndex] = part1 + "0" + part2;
        //MessageBox.Show(cluster[6]);
        forbiddenCityIndex = "";
    }
    public static void testParameter(string[] cluster, ref string solution, ref string bestSolution, ref string chainIndex, ref double slackTime, ref double bestSlackTime, ref string forbiddenCityIndex, ref int clusterIndex, ref int tourCount)
    {
        MessageBox.Show("Cluster[" + clusterIndex.ToString() + "] : " + cluster[clusterIndex] + "\r\n" +
                        "Solution : " + solution + "\r\n" +
                        "Best Solution : " + bestSolution + "\r\n" +
                        "Slack Time : " + slackTime.ToString() + "\r\n" +
                        "Best Slack Time : " + bestSlackTime + "\r\n" +
                        "Chain Index : " + chainIndex + "\r\n" +
                        "Forbidden City Index : " + forbiddenCityIndex + "\r\n" +
                        "Tour count : " + tourCount.ToString());
    }
    private static string notForbiddenCity(string[] cluster, ref string forbiddenCityIndex)
```

```
    {
        int k;
        string clusterData = "", notForbidden = "", reference = "";
        clusterData = cluster[Convert.ToInt32(forbiddenCityIndex.Substring(0, 3))];
        k = (Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)) - 3) / 17;
        reference = clusterData.Substring(Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)) + 1, 3);
        for (int i = 0; i < ((clusterData.Length - 3) / 17) - k; i++)
        {
            notForbidden = clusterData.Substring(Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)) + 1 + i * 17, 3) + ":" + indexFormat((Convert.ToInt32(forbiddenCityIndex.Substring(3, 4)) + 1 + i * 17).ToString(), 4);

            if (reference == notForbidden.Substring(0, 3))
            {
                continue;
            }
            else
            {
                return notForbidden;
            }
        }
        return "-1";
    }
    private static string findNonVisitedCity(string cluster, ref string forbiddenCityIndex, ref int time, ref int tourCount, ref int tourCount2)
    {
        int loopEnd = (cluster.Length - 3) / 17;
        int i, forbidden = 0;
```

```
        if (cluster.Substring(0, 3) == "000")
        {
            return "-2";
        }
        if (forbiddenCityIndex != "")
        {
            forbidden = Convert.ToInt32(forbiddenCityIndex.Substring(3, 4));
            forbidden = (forbidden - 3) / 17;
        }
        for (i = forbidden; i < loopEnd; i++)
        {
            if (cluster.Substring(3 + 17 * i, 1) == "1")
            {
                continue;
            }
            else
            {
                return findBestSlackTime(cluster, (3 + 17 * i + 1).ToString(), ref time, ref tourCount, ref tourCount2);
            }
        }
        return "-1";
    }
    private static string findBestSlackTime(string cluster, string subClusterIndex, ref int time, ref int tourCount, ref int tourCount2)
    {
        int slackTime, nextTime;
```

```
        nextTime = getTime(cluster.Substring(Convert.ToInt32(subClusterIndex) + 3,
5));
        if (time == -1 || tourCount == tourCount2)
        {
           slackTime = 0;
        }
        else
        {
           if (time <= nextTime)
           {
              slackTime = nextTime - time;
           }
           else
           {
              slackTime = 1440 + (nextTime - time);
           }
        }
        //MessageBox.Show(clusterIndex.ToString());
        //MessageBox.Show("Slack time : " + slackTime.ToString() + "Time : " +
time.ToString() + "Next time : " + nextTime.ToString() + "  2.part : " +
getTime(cluster.Substring(Convert.ToInt32(subClusterIndex) + 8, 5)).ToString() + "
ClusterIndex");
        time = (nextTime +
getTime(cluster.Substring(Convert.ToInt32(subClusterIndex) + 8, 5))) % 1440;

        return indexFormat(subClusterIndex, 4) + indexFormat(slackTime.ToString(),
4);
    }
    private static string changeString(string text, string pieceOfText, string index)
    {
       string newText;
```

```
            newText = text.Substring(0, Convert.ToInt32(index)) + pieceOfText + text.Substring(Convert.ToInt32(index) + pieceOfText.Length, text.Length - Convert.ToInt32(index) - pieceOfText.Length);
            return newText;
        }
        private static int refreshTime(string cluster, ref int time, ref string subClusterIndex)
        {
            int index = Convert.ToInt32(subClusterIndex.Substring(0, 4));
            return (time + getTime(cluster.Substring(index + 8,5))) % 1440;
        }
    }
}
```

Form2.cs dosyası için kaynak kodları:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Data.OleDb;
using System.IO;
namespace Vehicle_Routing_Problem_Solution
{
    public partial class Form2 : Form
    {
```

```
public Form2()
{
   InitializeComponent();
}
private void Form2_Load(object sender, EventArgs e)
{
   listViewLoad();
}
private void button1_Click(object sender, EventArgs e)
{
   dataChange();
}
private void button2_Click(object sender, EventArgs e)
{
   foreach (ListViewItem item in listView1.SelectedItems)
   {
      item.Remove();
   }
}
private void button3_Click(object sender, EventArgs e)
{
   string data = "";
   InputBox("Add New Data", "New Data: ", "Journey Number, City Departur, City Destination, Duration", ref data);
}
private void button4_Click(object sender, EventArgs e)
{
   this.Close();
}
```

```
private void button4_Click_1(object sender, EventArgs e)
{
   Close();
}
private void button5_Click(object sender, EventArgs e)
{
}
private void listView1_DoubleClick(object sender, EventArgs e)
{
   dataChange();
}
public void listViewLoad()
{
   listView1.Clear();
   listView1.Columns.Add("Journey Number", 100);
   listView1.Columns.Add("City Departur", 100);
   listView1.Columns.Add("City Destination", 100);
   listView1.Columns.Add("Departur Time", 100);
   listView1.Columns.Add("Duration", 100);
   listView1.Columns.Add("Day", 100);
   listView1.View = View.Details;
   StreamReader read = new StreamReader("settings.txt");
   Form1.databaseAddress = read.ReadLine();
   read.Close();
   try
   {
      string source = @"Provider=Microsoft.ACE.Oledb.12.0;Data Source=" +
Form1.databaseAddress;
      OleDbConnection databaseConnection = new OleDbConnection(source);
```

```
            string sqlCommand = "SELECT * FROM journey";
            databaseConnection.Open();
            OleDbCommand command = new OleDbCommand(sqlCommand, databaseConnection);
            OleDbDataReader print = command.ExecuteReader();
            while (print.Read())
            {
               string[] data = new string[6];
               for (int i = 0; i < 6; i++)
               {
                  data[i] = print[i + 1].ToString();
               }
               ListViewItem add = new ListViewItem(data);
               listView1.Items.Add(add);
            }
            databaseConnection.Close();
         }
         catch { }
      }
      void dataChange()
      {
         string changedData = "", newData = "";
         ListViewItem item = listView1.Items[listView1.SelectedIndices[0]];
         changedData =
         item.SubItems[0].Text + "," +
         item.SubItems[1].Text + "," +
         item.SubItems[2].Text + "," +
         item.SubItems[3].Text + "," +
         item.SubItems[4].Text + "," +
```

```
            item.SubItems[5].Text;
            InputBox("Change Data", changedData, changedData, ref newData);
            string sqlCommand = "update journey set journey_number='" +
               partOfString(newData, 0) + "', city_departur='" +
               partOfString(newData, 1) + "', city_destination='" +
               partOfString(newData, 2) + "', departur_time='" +
               partOfString(newData, 3) + "', duration='" +
               partOfString(newData, 4) + "', day_of_journey='" +
               partOfString(newData, 5) + "' where journey_number='" + item.SubItems[0].Text + "'";
            string source = @"Provider=Microsoft.ACE.Oledb.12.0;Data Source=" + Form1.databaseAddress;
            OleDbConnection databaseConnection = new OleDbConnection(source);
            databaseConnection.Open();
            OleDbCommand command = new OleDbCommand(sqlCommand, databaseConnection);
            command.ExecuteNonQuery();
            databaseConnection.Close();
            listViewLoad();
        }
        public static string partOfString(string data, int index)
        {
            string part = "";
            string character = "";
            int count = 0;
            for (int i = 0; i < data.Length; i++)
            {
               character = data.Substring(i, 1);
               if (character == ",")
```

```
            {
                count++;
                if ((count - 1) == index)
                {
                    return part;
                }
                else
                {
                    part = "";
                    continue;
                }
            }
            part = part + character;
        }
        if (count < index) return "";
        return part;
    }
    public static DialogResult InputBox(string title, string promptText, string defaultText, ref string value)
    {
        Form form = new Form();
        Label label = new Label();
        TextBox textBox = new TextBox();
        Button buttonOk = new Button();
        Button buttonCancel = new Button();
        form.Text = title;
        label.Text = promptText;
        textBox.Text = value;
        buttonOk.Text = "Ok";
```

```
            buttonCancel.Text = "Cancel";
            buttonOk.DialogResult = DialogResult.OK;
            buttonCancel.DialogResult = DialogResult.Cancel;
            label.SetBounds(9, 20, 372, 13);
            textBox.SetBounds(12, 36, 372, 20);
            buttonOk.SetBounds(228, 72, 75, 23);
            buttonCancel.SetBounds(309, 72, 75, 23);
            label.AutoSize = true;
            textBox.Anchor = textBox.Anchor | AnchorStyles.Right;
            buttonOk.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;
            buttonCancel.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;
            form.ClientSize = new Size(396, 107);
            form.Controls.AddRange(new Control[] { label, textBox, buttonOk,
buttonCancel });
            form.ClientSize = new Size(Math.Max(300, label.Right + 10),
form.ClientSize.Height);
            form.FormBorderStyle = FormBorderStyle.FixedDialog;
            form.StartPosition = FormStartPosition.CenterScreen;
            form.MinimizeBox = false;
            form.MaximizeBox = false;
            form.AcceptButton = buttonOk;
            form.CancelButton = buttonCancel;
            textBox.Text = defaultText;
            DialogResult dialogResult = form.ShowDialog();
            value = textBox.Text;
            return dialogResult;
        }
    }
}
```

Form3.cs dosyası için kaynak kodları:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;
namespace Vehicle_Routing_Problem_Solution
{
   public partial class Form3 : Form
   {
      public Form3()
      {
         InitializeComponent();
      }
      private void Form3_Load(object sender, EventArgs e)
      {
         textBox1.Text = Form1.databaseAddress;
      }
      private void button1_Click(object sender, EventArgs e)
      {
         StreamWriter write = new StreamWriter("Settings.txt");
         write.Write(textBox1.Text);
         write.Close();
```

```
            Close();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            openFileDialog1.ShowDialog();
            textBox1.Text = openFileDialog1.FileName.ToString();
        }
        private void button3_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

ÖZGEÇMİŞ

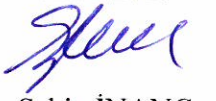
Adı-Soyadı	Şahin		İNANÇ
Doğum Yeri ve Yılı	Kırcaali / Bulgaristan		02.12.1975
Bildiği Yabancı Diller	İngilizce, Bulgarca		
Eğitim Durumu	**Başlama - Bitirme Yılı**		
Lise	1989	1992	Ertuğrulgazi Lisesi
Lisans	1994	2002	Trakya Üniversitesi Bilg.Mühendisliği
Yüksek Lisans	2005	2010	Uludağ Üniversitesi Sos.Bil. Enstitüsü
Doktora			
Çalıştığı Kurumlar	**Başlama - Ayrılma Yılı**		**Çalışılan Kurumun Adı**
1. Demisaş	2003	2004	Demisaş
2. Hipotez Bilg.	2004	2005	Hipotez Bilg.
3. ULUDAĞ ÜNİVERSİTESİ TBMYO	2005	2006	ULUDAĞ ÜNİVERSİTESİ
4. ULUDAĞ ÜNİVERSİTESİ KMYO	2006	2018	ULUDAĞ ÜNİVERSİTESİ
Üye Olduğu Bilimsel ve Meslekî Kuruluşlar			
Katıldığı Proje ve Toplantılar			

Yayınlar:	Kitap ve Bölüm Yazarlığı: 1. Nicel Karar Yöntemlerinde Güncel Konular:Teori ve Uygulama, Bölüm Adı: Dinamik Programlama İle Otomatik Kılavuzlu Araç Rotalama, EREN ŞENARAS ARZU,İNANÇ ŞAHİN,SEZEN HAYRETTİN KEMAL, Yayın Yeri:Gazi Kitabevi, Editör:Hasan Türe,Deniz Koçak, Basım sayısı:1, ISBN:21554, 2. Nicel Karar Yöntemlerinde Güncel Konular:Teori ve Uygulama, Bölüm Adı: Fiyat İndirimli Envanter Modellerinde VBA Uygulaması, İNANÇ ŞAHİN,EREN ŞENARAS ARZU,SEZEN HAYRETTİN KEMAL, Yayın Yeri:Gazi Kitabevi, Editör:Hasan Türe, Deniz Koçak, Basım sayısı:1, Sayfa sayısı:165, ISBN:978-605-344-767-2, Bölüm Sayfaları:165 -176 Makaleler: 1. C Programlama Dilinde Geliştirilen Program İle Euler Sayısının Rasgeleliğinin Sınanması, ARZU EREN ŞENARAS,ŞAHİN İNANÇ , Yayın Yeri:Anemon Muş Alparslan Üniversitesi Sosyal Bilimler Dergisi , 2018 2. DİNAMİK PROGRAMLAMA İLE AGV HATTI İÇİN VBA UYGULAMASI, ARZU EREN ŞENARAS,ŞAHİN İNANÇ , Yayın Yeri:Journal of Life Economics , 2018 3. GSP Çözümü İçin Karınca Kolonisi Optimizasyonu, ARZU EREN ŞENARAS, Unvan: Araştırma Görevlisi, Yazar türü: Author, Şu anki unvan: Araştırma Görevlisi, ŞAHİN İNANÇ , Yayın Yeri:Sosyal Bilimler Metinleri Dergisi , 2017 4. pi Sayısının Rasgeleliğinin Sınanması, ŞAHİN İNANÇ, Unvan: Öğretim Görevlisi, Yazar türü: Author, Şu anki unvan: Öğretim Görevlisi ,HAYRETTİN KEMAL SEZEN , Yayın Yeri:Journal of

	Life Economics , 2017 5. Testing Randomness Of Generated Numbers Based On The Henon Map, ARZU EREN ŞENARAS, Unvan: Araştırma Görevlisi, Yazar türü: Author, Şu anki unvan: Araştırma Görevlisi, ŞAHİN İNANÇ , Yayın Yeri:International Refereed Journal of Humanities and Academic Sciences , 2016 6. C Programlama Dilinde Geliştirilen Program İle Rasgeleliğin Sınanması, ARZU EREN ŞENARAS, Unvan: Araştırma Görevlisi, Yazar türü: Author, Şu anki unvan: Araştırma Görevlisi, ŞAHİN İNANÇ,HAYRETTİN KEMAL SEZEN , Yayın Yeri:Uludağ Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi , 2015 7. Analyzing Randomness Of Numbers Generated By Logistic Map, ARZU EREN ŞENARAS, Unvan: Araştırma Görevlisi, Yazar türü: Author, Şu anki unvan: Araştırma Görevlisi, ŞAHİN İNANÇ,HAYRETTİN KEMAL SEZEN , Yayın Yeri:Uluslararası Hakemli Ekonomi Yönetimi Araştırmaları Dergisi , 2014 Bildiriler: 1. En Kısa Yol Problemlerinin Dinamik Programlama ile VBA Uygulaması ARZU EREN ŞENARAS Unvan: Araştırma Görevlisi Yazar türü: Author. Şu anki unvan: Araştırma Görevlisi ,ŞAHİN İNANÇ,HAYRETTİN KEMAL SEZEN (29.09.2018 -30.09.2018) , Yayın Yeri:IX. Uluslararası Balkan ve Yakın Doğu Sosyal Bilimler Kongreler Serisi , 2018 2. Organizasyonel İnovasyon ve İşletme İçi Motivasyon Unsurları İlişkisi ŞAHİN İNANÇ,KÜBRA İNANÇ (27.10.2018 - 28.10.2018) , Yayın Yeri:X. Uluslararası Balkan ve Yakın Doğu Sosyal Bilimler Kongreler Serisi , 2018

	3. Monte Carlo Simülasyonu İçin VBA Uygulaması ARZU EREN ŞENARAS,ŞAHİN İNANÇ (29.09.2018 - 30.09.2018) , Yayın Yeri:IX. Uluslararası Balkan ve Yakın Doğu Sosyal Bilimler Kongreler Serisi , 2018 4. Makine Arızalarının Monte Carlo Benzetimi İle Tahmin Edilmesi ŞAHİN İNANÇ,ARZU EREN ŞENARAS (27.10.2018 -28.10.2018) , Yayın Yeri:X. Uluslararası Balkan ve Yakın Doğu Sosyal Bilimler Kongreler Serisi , 2018 5. Elektrik Enerjisi Piyasasının İncelenmesi ARZU EREN ŞENARAS,ŞAHİN İNANÇ (27.10.2018 -28.10.2018) , Yayın Yeri:X. Uluslararası Balkan ve Yakın Doğu Sosyal Bilimler Kongreler Serisi , 2018 6. Araç Rotalama Probleminin Çözümü için Karınca Kolonisi Optimizasyonu ŞAHİN İNANÇ,ARZU EREN ŞENARAS (20.05.2017 -21.05.2017) , Yayın Yeri:International Congress of Management Economy and Policy , 2017 7. Elektrik Enerjisi Üretimine Bağlı Co2 Salınımının Sistem Dinamiği Yaklaşımı İle Modellenmesi ARZU EREN ŞENARAS,ŞAHİN İNANÇ,HAYRETTİN KEMAL SEZEN (17.11.2017 -18.11.2017) , Yayın Yeri:International Congress of Management, Economy and Policy , 2017 8. Rasgelelik Testleri ve Bir Uygulma: Altın Oran Sayısı Örneği HAYRETTİN KEMAL SEZEN,ARZU EREN ŞENARAS,ŞAHİN İNANÇ (24.05.2013 - 28.05.2013) , Yayın Yeri:14.Uluslararası Ekonometri Yöneylem Araştırması ve İstatistik Sempozyumu , 2013 9. C Programlama Dilinde Geliştirilen Program İle Euler Sayısının Rasgeleliğinin Sınanması ARZU EREN ŞENARAS,ŞAHİN İNANÇ (20.05.2017 - 21.05.2017) , Yayın Yeri:International Congress of Management Economy amd

	Policy , 2017 10. Türkiye'de Nüfusun Markov Analizi İle İncelenmesi ARZU EREN ŞENARAS,ŞAHİN İNANÇ (17.11.2017 - 18.11.2017) , Yayın Yeri:International Congress of Management, Economy and Policy , 2017
Diğer:	
İletişim (e-posta):	sahininanc@uludag.edu.tr

Tarih	02.01.2019
İmza	
Adı-Soyadı	Şahin İNANÇ

BURSA ULUDAĞ ÜNİVERSİTESİ

TEZ ÇOĞALTMA VE ELEKTRONİK YAYIMLAMA İZİN FORMU

Yazar Adı Soyadı	**Şahin İNANÇ**
Tez Adı	**Aylak Zamanı Enküçükleyen Tur Oluşturma Problemlerinin Geri İzleme Yöntemi ile Çözümüne İlişkin Bir Yazılım Geliştirme Uygulaması**
Enstitü	**Sosyal Bilimler**
Anabilim Dalı	**Ekonometri Anabilim Dalı**
Tez Türü	**Doktora**
Tez Danışman(lar)ı	**Prof. Dr. H.Kemal SEZEN**
Çoğaltma (Fotokopi Çekim) İzni Kısıtlama	☐ Patent Kısıt (2 yıl) ☐ Genel Kısıt (6 ay) ☒ Tezimin elektronik ortamda yayımlanmasına izin veriyorum.

Hazırlamış olduğum tezimin belirttiğim hususlar dikkate alınarak, fikri mülkiyet haklarım saklı kalmak üzere Bursa Uludağ Üniversitesi Kütüphane ve Dokümantasyon Daire Başkanlığı tarafından hizmete sunulmasına izin verdiğimi beyan ederim.

Tarih : 02/01/2019

İmza :