ZAMAN PENCERELİ ARAÇ ROTALAMA PROBLEMİ İÇİN BİR KARAR DESTEK SİSTEMİ

Mehmet Aydın TOKAYLI

YÜKSEK LİSANS TEZİ
ENDÜSTRİ MÜHENDİSLİĞİ

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KASIM 2005
ANKARA

Mehmet Aydın TOKAYLI tarafından hazırlanan ZAMAN PENCERELİ ARAÇ ROTALAMA PROBLEMİ İÇİN BİR KARAR DESTEK SİSTEMİ adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

Prof.Dr.Serpil EROL
Tez Yöneticisi

Bu çalışma, jürimiz tarafından Endüstri Mühendisliği Anabilim Dalında Yüksek Lisans tezi olarak kabul edilmiştir.

Başkan: : Prof.Dr.Serpil EROL

Üye : Doç.Dr. Ümit YÜCEER

Üye : Yrd.Doç.Dr. Bahar ÖZYÖRÜK

Bu tez, Gazi Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygundur.

ZAMAN PENCERELİ ARAÇ ROTALAMA PROBLEMİ İÇİN BİR KARAR DESTEK SİSTEMİ

(Yüksek Lisans Tezi)

Mehmet Aydın TOKAYLI

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
Kasım 2005

ÖZET

Zaman pencereli araç rotalama problemi (ZPARP), merkez bir ana depodan, coğrafi olarak dağılmış müşterilerin (talep merkezlerinin) ihtiyaçlarını belirtilen zaman aralığı içinde karşılamak için en az araçlı ve en kısa mesafeli rotaları bulmayı amaçlar. ZPARP, modelleme yapısı nedeniyle okul otobüsü rotalama, gazete, posta dağıtımı, akaryakıt dağıtımı ve kentsel atık toplama ve gibi gerçek hayat karar problemlerini çözmek için uygundur. Bu nedenle, ZPARP hakkında yapılmış bir çok çalışma vardır. ZPARP için bir karar destek sisteminin (KDS) oluşturulması, bu rota planlamalarının kolay ve gerçek zamanlı olarak yapılmasına imkan tanıyacaktır. Bu amaçla bu çalışmada, ZPARP çözüm algoritmalarını kullanan bir KDS ilk olarak tasarlanmış, uygulaması anlatılmış ve mimari yapısı sunulmuştur.

Bilim Kodu : **919**
Anahtar Kelimeler : **Zaman Pencereli Araç Rotalama Problemi, Karar Destek Sistemleri, Karınca Kolonisi Algoritması**
Sayfa Adedi : **129**
Tez Yöneticisi : **Prof. Dr. Serpil EROL**

A DECISION SUPPORT SYSTEM FOR THE VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

(M.Sc. Thesis)

Mehmet Aydın TOKAYLI

GAZI UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

November 2005

ABSTRACT

Vehicle routing problem with time windows (VRPTW), aims to design routes using possible minimum number of vehicles and possible shortest total distance from one main depot to a set of geographically scattered customers (demand points). Because of its modeling structure, VRPTW is suitable to solve real life decision problems, such as school bus routing, mail, newspaper delivery, fuel oil delivery and municipal waste collection. For this reason, there are a lot of research in the literature. Designing a decision support system (DSS) for VRPTW enables planning of routes in an easy way and in real time conditions. For this purpose, in this research a first DSS which uses VRPTW solving algorithms is designed, its implementation is explained and its architecture is presented.

Science Code : 919
Key Words : Vehicle Routing Problem with Time Windows, Decision Support Systems, Ant Colony Optimization
Page Number : 129
Adviser : Prof. Dr. Serpil EROL

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren Sayın Hocam Prof. Dr. Serpil EROL başta olmak üzere tüm "Gazi Üniversitesi Endüstri Mühendisliği Bölümü" Öğretim Üyelerine ve Araştırma Görevlisi arkadaşlarım Hakan ÇERÇİOĞLU ve Tahsin ÇETİNYOKUŞ başta olmak üzere "Endüstri Mühendisliği Bölümünün" tüm Araştırma Görevlilerine teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış bazı simgeler ve kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Simgeler	Açıklama
a_{ij}	i' den j' ye olan ark
a_{rl}	l günü r kombinasyonuna aitse 1, yoksa 0
$b_i(t)$	i düğümünde t anında bulunan karınca sayısı
c_i	i. müşteri düğümü ($i \in \{1,2,...,N\}$)
c_0	merkez depo düğümü
$\bar{c}$	zincir uzunluğu maksimum değeri
$C1$	(i,j) müşterileri arasına u müşterisini eklemenin maliyeti
$C2$	u müşterisini ise eklemenin faydası
C_i	i müşterisine servis yapılacak kombinasyonlar
C_{ij}	i ve j düğümleri arasındaki seyahatten doğan maliyet
$\bar{d}$	komşu müşteri/düğüm/rota yakınlık ölçütü
d_{ij}	i ve j düğümleri arasındaki (öklid) uzaklık
d_{ijkl}	k aracının i' den j' ye l gününde gitme süresi
e_i	i düğümüne en erken varış zamanı
$G(N,A)$	N düğümleri ve A arklarından oluşan graf
I^{rt}	rotalar kümesini
k	araç indisi
K	toplam araç sayısı
l	ziyaret günü indisi
l_i	i düğümüne en geç varış zamanı
ll	uzaklık artışına izin verilen son limit değeri
m	toplam karınca sayısı
η_{ij}	seçilebilirlik parametresi

Simgeler	Açıklama
N	toplam müşteri/düğüm sayısı
N_i^k	k. karıncanın tabu listesinde olmayan i şehirleri
$N(rt_i)$	rt_i ' ye komşu rotalar kümesi
$p_{ij}(t)$	t anında (i, j) hattına şeçmek olasılık değeri
q_i	i müşterisi/düğümü talebi ($q_0 = 0$)
Q_k	k aracı kapasitesi
r	kombinasyon indisi $(r \in C_i)$
rs_k	k aracı için izin verilen maksimum rota süresi
rt_k	k aracının rotası
s_i	i düğümündeki servis zamanı ($s_0 = 0$)
se_i	servis sıklığı
S	alt tur engelleme kısıtı düğüm kümesi
t	planlama ufku
t_i	i düğümüne varış zamanı
t_{ij}	i ve j düğümleri arasındaki seyahat süresi
x_{ij}	karar değişkeni (i ' den j ' ye giderse 1, yoksa 0)
x_{ijk}	karar değişkeni (k aracı, i ' den j ' ye gidiyorsa 1, yoksa 0)
x_{ijkl}	karar değişkeni (k aracı i ' den j ' ye l gününde giderse 1, yoksa 0)
w_i	i düğümünde bekleme zamanı
y_{ir}	$r \in C_i$ kombinasyonu i müşterisine atanmışsa 1, yoksa 0
$\tau_{ij}(t)$	t anında (i, j) hattına bırakılan feremon miktarı
ψ^m	m karıncasının oluşturduğu çözüm
ψ^{iyi}	o ana kadar olan en iyi çözüm
$J\psi^{iyi}$	o ana kadar olan en iyi çözüm uzunluğu
α	feremon maddesi nispi önemi
β	seçilebilirlik kriteri nispi önemi

Simgeler	Açıklama
∂_θ	seçim olasılığı (keşif/kullanma)
ρ	buharlaşma katsayısı

Kısaltmalar	Açıklama
ARP	Araç Rotalama Problemi
ARS	Araç Rotalama Sistemi
BT	Bilgi Teknolojileri
CBS	Coğrafi Bilgi Sistemi
CNV	Cumulative Number of Vehicle – Toplam Araç Sayısı
CTD	Cumulative Total Distance – Mesafeler Toplamı
ÇDARP	Çok Depolu Araç Rotalama Problemi
GRASP	Greedy Randomized Adaptive Search Procedure
GSP	Gezgin Satıcı Problemi
KDS	Karar Destek Sistemi
KFARP	Karışık Filolu Araç Rotalama Problemi
KKO	Karınca Kolonisi Optimizasyonu
MTYS	Model Tabanı Yönetim Sistemi
PARP	Periyodik Araç Rotalama Problemi
YBARP	Yer Bağımlı Araç Rotalama Problemi
YBS	Yönetim Bilgi Sistemleri
VTYS	Veri Tabanı Yönetim Sistemi
ZPARP	Zaman Pencereli Araç Rotalama Problemi
ZPARP-KDS	ZPARP Karar Destek Sistemi

1. GİRİŞ

Fayda sağlayan ekonomik değerler, her zaman insanların kolay elde edebileceği mekanlarda bulunmaz. Bu tür değerlerin, çok bulunduğu yerlerden az bulunan yerlere taşınarak insan ihtiyaçlarının karşılanması gerekir. Örneğin Güney Amerika ülkelerinde yetiştirilen kakao ve kahvenin bu ülkelerden, bu ürünlerin yetişmediği Türkiye gibi ülkelere taşınması gerekir. Ülkemiz açısından düşünürsek, Manisa'da üretilen bir beyaz eşyanın da ülkemiz içindeki veya yurtdışındaki dağıtıcı firmalara ulaştırılması müşteri ihtiyaçlarını karşılamak açısından önemlidir. Ayrıca bazı maddelerin bulunduğu yerlerden toplanarak ayrıştırma ve yeniden değerlendirme merkezlerinde toplanma durumu söz konusudur. Bu açıklamalar, taşıma faaliyetlerinin ekonomik gelişme açısından önemini vurgulamaktadır.

Bununla beraber gıda, tekstil veya atık gibi taşınması gereken malzemelerin taşınmasında da zaman faktörüne dikkat etmek gerekir. Gıdaların zamanında ulaştırılmaması bozulmasına, tekstil malzemelerinin dağıtımının gecikmesi pazar ve prestij kaybına, evsel atıkların zamanında toplanıp toplama merkezlerine getirilmemesi çevre kirliliği veya salgınlara da neden olabilir. Kısacası, taşıma faaliyetinin zamanında gerçekleşmesi gereklidir.

Yukarıda bahsedilen tüm konular ilgili kurumlar için planlı bir taşıma faaliyetini zorunlu kılar. Plansız ve düzensiz taşıma, maliyetleri çok fazla artırabilir. Taşıma faaliyetlerinin düzenli olarak, değişen şartlara göre mümkün olan en az maliyetle organize edilmesi gerekmektedir. Bu ise kolay kullanımlı otomasyon ve karar destek sisteminin (KDS) taşıma ve araç rotalama işlemlerini planlaması ile olur.

Bu nedenle bu çalışmada bir çok gerçek hayat problemine (otobüs rotalama, posta ve gazete dağıtımı v.b.) uyarlanabilen zaman pencereli araç rotalama problemi için kullanıcı-dostu bir karar destek sistemi tasarlanarak, taşıma ve lojistik faaliyetleri üzerinde çalışan, bilgisayar tabanlı sistemlere ihtiyaç duyan uzman personelin karar alma süreçlerinin kolaylaştırılması amaçlanmıştır.

Tasarlanan bu KDS yedi bileşenden oluşmuş olup, bunlar; dosya modülü, harita modülü, çözüm modülü, yönetim modülü, arayüz modülü, raporlama modülü ve kullanıcıdır. Dosya ve harita modülleri üzerinde karar verilecek ZPARP' nin verilerini sağlamakta, çözüm modülü istenen çözüm hassaslığına göre çeşitli deterministik ve meta-heuristik algoritmalardan yararlanarak sisteme çözüm alternatifleri sunmakta, yönetim modülü sistemdeki veri-bilgi akışını kontrol etmekte, arayüz modülü KDS ile kullanıcı arasında iletişim sağlamakta, raporlama modülü istenen raporları üretmekte ve kullanıcı ise bu karar destek sisteminden yararlanarak rota planlarını oluşturmaktadır.

Bu kapsamda, tasarlanan zaman pencereli araç rotalama problemi karar destek sistemi (ZPARP-KDS) için, 2. bölümde karar destek sistemleri hakkında genel bilgi verilmiş, 3. bölümde gezgin satıcı problemi (GSP), araç rotalama problemi ve problem türleri anlatılmış, 4. bölümde ise zaman pencereli araç rotalama probleminin tanımı, matematiksel modeli ve ilgili literatürü anlatılmıştır. 5. bölümde, çözüm modülünde kullanılabilen algoritmalardan biri olan *ANA_ZPARP-KDS* algoritmasında faydalanılan karınca kolonisi algoritması tanıtılmış, 6. bölümde ise tasarlanan ZPARP-KDS' nin mimarisi anlatılmıştır. Elde edilen sonuçlar ve gelecek çalışmalar için tavsiyeler sonuç ve öneriler bölümündedir.

2. KARAR DESTEK SİSTEMLERİ

İş dünyasında insanlar sıkça kapsamlı veya basit şekilde kararlar almak durumundadır. Bu kararlar yatırım yapma-yapmama, kapasite artırma-azaltma, işçi alma-çıkarma, çalışma saatlerini artırma, pazarlama stratejilerini geliştirme, ürün yelpazesini artırma-azaltma, üretim teknolojilerini değiştirme-değiştirmeme gibi faaliyetleri içerir. Bu kararları vermek her zaman kolay değildir ve bu kararları verdikten sonra her zaman doğru sonuçlar elde edilmeyebilir.

Bu nedenle karar, ondan beklenen sonucun varlığı ve çeşidine göre yapısal ve yapısal-olmayan, diye ikiye ayrılır. *Yapısal karar*, belli bir bilgiyi işledikten sonra sonunda her zaman doğru cevaba ulaşılan karardır. *Yapısal olmayan karar* ise, doğru cevabı bilmenin mutlak bir yolunun olmadığı ve ortada bir çok doğru karar olabildiği durumlarda söz konusudur (1). Karar yapı itibariyle bu iki durumun arasında *yarı-yapısal karar* olarak da adlandırılabilir.

Karar Destek Sistemleri, böyle yarı-yapısal ve yapısal olmayan kararlar alınması gerektiğinde devreye girer. Bir bilgi teknolojisi (BT) olan KDS, iş hayatında karar verme sürecini kolaylaştırır, karar zamanını kısaltır ve daha kesin sonuçlu kararlar alınmasına imkan tanır. Çünkü BT' lerinin ana unsuru olan bilgisayarlarda her türlü kararın hızlı bir şekilde alınması sağlayacak yüksek işlem hızı ve hafıza kapasitesi mevcuttur. Bu nedenle bilgisayarlar, karar vericinin kendi kapasitesiyle hemen elde etmesi zor olan sonuçlara kolay olarak ulaşarak karar vericilere yardımcı olur.

KDS'lerin ortaya çıkışı, örgüt yönetiminde kullanılan bilgilerin işlenmesini ve iletilmesini sağlayan Yönetim Bilgi Sistemlerinin (YBS) eksikliğinden kaynaklanmıştır (1). Bu eksiklikten dolayı 1970' lerde bazı firmalar, geleneksel YBS' lerden farklı bilgi sistemleri tasarlamaya başlamışlardır. Bu süreçte ortaya çıkan KDS' leri takip eden süre içerisinde akademik ve endüstriyel düzeyde konuyla ilgili araştırma-geliştirme ve uygulama çalışmaları hızla yaygınlaşmıştır (2).

2.1. Karar Destek Sistemlerinin Tanımı

Yöneticilerin, yönetsel problemlerin aşılması için kantitatif modelleri kullanma çabalarıyla ortaya çıkan karar desteği, ilk 1970 yılında J. D. Little çalışmasıyla ortaya konmuştur. Terim olarak KDS' nin kullanıldığı ilk çalışma ise Gorry ve Scott-Morton (1971)' a aittir (3). Bu çalışmada, *"yönetim karar sistemleri"* kavramı ortaya atılmıştır. İleri sürdükleri sistemi, karar vericilerin yapısal-olmayan problemleri çözerken veri ve modelleri kullanmasına imkan sağlayan, etkileşimli bilgisayara dayalı sistemler olarak tanımlamışlardır. Aynı araştırmacıların başka bir tanımında ise KDS, bireylerin enteklektüel kaynakları ile bilgisayarların karar iyileştirme yeteneklerini birleştiren ve yarı-yapısal problemlerle ilgilenen yönetim seviyesi karar vericiler için bilgisayara dayalı destek imkanı sağlayan sistemler olarak vurgulanmıştır (2).

Yukarıda geçen tanımlar, KDS' nin dört ana karakteristiğini vurgulamaktadır. Bunlar KDS' nin, veri ve modelleri içerisine alması, yönetici hükümlerinin yerine geçmekten ziyade onlara destek vermesi, yöneticilere yarı-yapısal (veya yapısal olmayan) görevlerde yardımcı olması ve hangi kararlarla verimlik sağlandığı değil, verilen kararlardaki etkinliğin artırılması amacıdır (2).

Bu kavramlar ışığında *KDS* için literatürde bulunan tanımlar aşağıdaki gibidir (3):

- Bir KDS, kullanıcıya yarı-yapısal ve yapısal olmayan karar verme işlemlerinde destek sağlamak amacıyla, karar modellerine ve verilere kolay erişimi sağlayan etkileşimli bir sistemdir.

- KDS, kararın yapısal olmadığı durumlarda karar alma işlemine yardımcı olmak için tasarlanmış, esnek ve etkileşimli bilişim teknolojisi sistemleridir.

- KDS, karmaşık problemleri çözebilmek için insan zekası, bilgi teknolojisi ve yazılımın etkileşim içerisinde olacak şekilde harmanlandığı bir sistemdir.

• Karar verme sürecinde, yönetime destek vermek için hedeflenen bilginin üretilmesi ve sunulması için kullanıcı etkileşimli yazılım ve donanım vasıtalarının bütünleşik kümesinden oluşan etkileşimli bilgi sistemleridir.

• Karar vericinin yerine geçmesinden ziyade onun kararlarını destekleyen, yarı-yapısal ve yapısal olmayan problemlerin çözümü için karar vericiye karar vermesinde yardımcı olan etkileşimli sistemlerdir.

Model-güdümlü ve veri-güdümlü olmak üzere temel iki tip KDS vardır:

Model-güdümlü KDS' ler aslında ana organizasyonun bilgi sisteminden bağımsız olarak çalışan ve "Şayet...ise (What ... if)" ve diğer farklı tip analizler için bir takım modeller kullanan sistemlerdir. Bu tür sistemler genellikle merkezi bilgi sistemi kontrolü altında olmayan son kullanıcı kısım veya gruplar tarafından tasarlanır (4). Bu sistemlerin analiz yetenekleri, modelin kullanımını kolaylaştıracak iyi bir kullanıcı ara yüzüyle birleştirilmesine bağlıdır.

İkinci tip KDS olan veri-güdümlü KDS' ler ise, ana organizasyonun geniş veri havuzunu analiz eder. Karar vericiye destek olmak için, büyük miktardaki veriler arasına gömülmüş faydalı bilgileri gün yüzüne çıkarırlar. Veri işleme sistemlerinden elde edilen veriler, bu amaç için veri ambarlarında toplanırlar (4).

2.2. Karar Destek Sistemlerinin Özellikleri ve Yetenekleri

KDS özellik ve yetenekleri aşağıdaki gibidir (2):

1) KDS, yarı-yapılandırılmış ve yapılandırılmamış durumlarda insan yargıları ve bilgisayar kaynaklı bilgileri bir araya getirerek karar vericiler için karar desteği sağlarlar.

2) KDS, veri inceleme ve çözüm üretmede analitik modeller kullanır.

3) Üst kademe yöneticilerden alt kademe yöneticilere kadar, tüm yönetim seviyelerine karar desteği verirler.

4) Gruplara olduğu kadar, bireylere de karar desteği verirler.

5) KDS' ler, bağımsız ve/veya sıralı kararlar için karar desteği sağlarlar.

6) KDS' ler karar verme süreçlerinin tüm aşamalarında karar desteği sağlar.

7) KDS' ler farklı karar verme süreçleri ve tiplerine destek verirler.

8) KDS, karar verici kişi ya da grubun karar verme yaklaşımındaki değişiklikleri göz önüne alarak buna uyumu sağlayacak nitelikte esneklik özelliğine sahip olabilmektedir.

9) KDS kolay kullanımlı bir yapıda olmalıdır. Esneklik, güçlü grafik yetenekler ve kullanıcıyla uyumlu diyalog dili KDS' nin etkisini büyük ölçüde artırmaktadır.

10) KDS, karar vermenin verimliliğinden (maliyet gibi) çok, karar vermenin etkinliğini (doğruluk, zamanlılık ve kalite gibi) iyileştirmeye çalışır.

11) KDS, karar verici yerine geçmekten ziyade, karar vericiye destek olmayı amaçlamaktadır.

12) KDS' nin kurulumu kolay olmalıdır.

2.3. Karar Destek Sistemlerinin Bileşenleri

Bir Karar Destek Sistemi dört alt sistemden oluşur (1);

- Veri Yönetimi: Veri tabanından oluşur ve bir Veri Tabanı Yönetim Sistemi (VTYS) tarafından yönetilir.

- Model Yönetimi: Mali, istatistiki, yönetim bilimi ve diğer kantitatif modellerden oluşan bir yazılım paketidir.

- İletişim (diyalog) alt sistemi: Kullanıcının KDS ile haberleştiği ve komutlar gönderdiği alt sistemdir.

- Kullanıcı: KDS' den beklenen faydalardan yararlanan karar vericilerdir.

2.3.1. Veri yönetimi alt sistemi

Veri yönetimi alt sistemi (2);

- KDS veri tabanı

- VTYS

- Veri Rehberi / Sözlüğü

- Sorgulama Sisteminden oluşur.

Veri tabanı

Veri tabanı, organizasyonun yapısını ve ihtiyaçlarını karşılamak üzere düzenlenmiş birbiriyle ilişkili veriler topluluğudur. Geniş KDS uygulamalarında "Veri Ambarı" büyüklüğünde olacağı gibi bazı özel amaçlı KDS uygulamalarında ihtiyaca göre büyüyen bir veri tabanı da olabilir.

Veri tabanındaki veriler dahili ve harici veri kaynaklarından elde edilebileceği gibi kullanıcılara ait kişisel verilerden de oluşabilirler. Dahili veri organizasyonun Kayıt/Veri İşleme Sistemlerinden elde edilir. (Örneğin maaş bordrosu) Harici veri, endüstriyel veriler, sayım sonuçları, kamu düzenlemeleri, milli ekonomik göstergelerden oluşabilir. Çevrimiçi servislerden elde edilebileceği gibi web tabanlı olarak da elde edilebilir. Kişisel veriler ise çeşitli karar vericiler tarafından hazırlanmış tahmin bilgileri veya rehber bilgilerdir.

Veri tabanı yönetim sistemi

Veri tabanı, oluşturma, erişim ve güncelleme işlemleri bir VTYS aracılığı ile yapılır.

Sorgulama sistemi

Veri tabanındaki veriye erişim, işleme ve sorgulama sistemi aracılığıyla yapılır. Sistem çeşitli KDS parçalarından gelen ihtiyaçları formüle ederek veri tabanına erişir ve sonuçları talep eden modüle gönderir.

Veri Rehberi / Sözlüğü

Katalog görevi görür. Veri tanımlarını kapsar ve ana görevi veriye erişim yolunu göstermektir.

2.3.2. Model yönetimi alt sistemi

Model yönetimi alt sistemi (2);

- Model tabanı,

- Model tabanı yönetim sistemi (MTYS),

- Modelleme dili,
- Model Rehberi,
- Modelin uygulaması, birleştirmesi ve komut işlemcisi

Model tabanı

Bir KDS sistemine analiz yetenekleri sağlayan çeşitli istatistik, finansal, tahmin, yönetim bilimi ve diğer kantitatif modellerden oluşan bileşendir. Eldeki problemin çözümüne yönelik ilgili modelin seçilebilmesi KDS' ni bilgisayar tabanlı bilgi sistemlerinden ayıran en önemli özelliktir. Model tabanındaki modeller, stratejik modeller, taktik modeller, operasyonel modeller ve model oluşturma rutinleri olmak üzere dört ana kategoriye ayrılırlar.

Stratejik modeller, üst yönetimin stratejik planlama görevlerini yerine getirmede yararlandığı modellerdir. Organizasyonel hedefler geliştirme, fabrika yeri seçimi, çevresel etki analizi, bütçeleme gibi faaliyetlerde kullanılırlar. Çoğunlukla harici veri kullanırlar.

Taktik modeller, ara yönetim birimleri tarafından kaynak tahsisi ve kontrolü faaliyetlerinde kullanılırlar. Bu modellerden, işgücü ihtiyaçları planlaması, satış promosyon planlaması, fabrika tasarımı ve rutin bütçe planlaması çalışmalarında yararlanılır. Bazı harici verinin yanı sıra büyük ölçüde dahili veri kullanan modellerdir.

Operasyonel modeller, günlük çalışma faaliyetlerini desteklemek amacıyla kullanılırlar. Üretim planlaması, stok kontrol, bakım planlaması, zaman planlaması ve kalite kontrol faaliyetlerinde yararlanılırlar. Alt seviye yöneticilerinin kararlarına destek olmada kullanılırlar ve dahili verilerle beslenirler.

Model oluşturma rutinleri, veri analizi gibi tek başına kullanılacağı gibi tüm modeller tarafından kullanılabilen rastsal sayı üreteci, eğri uydurma rutini, regresyon analizi gibi modüllerdir.

Model tabanı yönetim sistemi

Model oluşturan model sistemidir. Kullanıma yönelik gerekli altyordamları birleştiren, yeni rutinlerin ve raporların tanımlanmasını sağlayan, modellerin güncellenmesi, değiştirilmesi ve veri işleme ile raporlamada faydalanılan sistemdir.

*Modelleme dilleri***,** KDS' nin kullanıldığı yarı-yapısal ve yapısal olmayan problemlerin çözümünde kullanılacak modellerin oluşturulmasına imkan sağlayan üst seviye programlama dilleridir.

Model rehberi, model tabanında yer alan yazılım ve modellerin katalogudur.

Modelin uygulaması, çalışmakta olan modelin kontrolünü içerir. Model birleştirmesi, değişik modellerin bir kompozisyon içinde kullanılması (bir modelin çıktısının diğerinin girdisi olması) faaliyetidir. Komut işlemcisi, kullanıcı taleplerini model tabanı yönetim sistemine ileten ara yüzdür.

2.3.3. Kullanıcı arayüzü (diyalog) alt sistemi

Kullanıcı ile KDS' nin iletişimini sağlayan KDS bileşenidir. Kullanıcının sistemle doğrudan temasta bulunduğu tek bileşendir. Bu bileşen, temel olarak girdi-çıktı araçları, konuşma-sorgulama dili işleyicisi, diyalog üretme ve yöneltme araçlarını içerir (3).

2.3.4. Kullanıcı

Karar vericiler, KDS' yi kullanan ve yöneten kişilerdir. Karar vericiler, kullanıcı arayüzü yardımıyla KDS' yi yönlendirirler. Ele aldığı problemin gerekleri

doğrultusunda KDS' yi kullanarak sonuç raporlarında ve tablo analizlerinden hareketle, alternatif çözümler içerisinden en iyiyi bulmaya çalışır (3). Uygulamada KDS kullanıcıları her seviye yöneticiler ve konuyla ilgili uzmanlar oluşturmaktadır.

2.4. KDS Donanımı

KDS donanımı, bilgisayar donanımı ve yazılım teknolojilerinin gelişimiyle gün geçtikçe gelişmektedir. KDS ana donanım seçenekleri;

- Ana bilgisayar
- Web sunumcu sistemi
- Intranet
- Internet
- İş istasyonu
- Kişisel bilgisayardır.

2.5. Karar Destek Sistemlerinin Fonksiyonları

Karar Destek Sistemleri uygulamaları şu fonksiyonlara sahip olmalıdır (1).

Model yaratma

Bir karar verme probleminde model yaratmak, bir çok Karar Destek Sisteminin ana amacıdır. Bu model genelde iki veya daha fazla boyutlu bir tablo halindedir (Excel dokümanları gibi). Tablodaki ilk iki boyut gelirleri, üçüncü boyut çeşitli ürünleri ve dördüncü boyut da perakende satış mağazalarını gösterebilir. Bu durumda model

gelişimi, çeşitli satış ve harcama değişkenleri arasındaki ilişkileri matematiksel olarak kurmayı gerektirir. Örneğin buradaki modeli geliştirirken, satışın reklam harcamalarının bir fonksiyonu olduğunu varsayabiliriz. Matematiksel olarak bu fonksiyon Satış = 20 * (Reklam Harcamaları) şeklinde olabilir. Bu fonksiyon, reklama harcanan her bir YTL' nin, satışı 20 YTL artıracağı anlamına gelir.

Şayet .... olursa (what-if) analizleri

Karar Destek Sistemleri'nin en faydalı özelliği veriler ve varsayımlardaki değişimlerin etkisini gösterebilme kabiliyetidir. Örneğin Karar Destek Sistemleri, eğer satışlar yüzde 5 yerine yüzde 7 veya 10 artarsa bunun kâra etkisini gösterebilir. Birçok Karar Destek Sistemleri uygulamaları bu tarz değişimlerin sonuçlarını anında gösterebilmektedir.

Risk analizi

Risk analizi sonucu elde edilen olasılık dağılımı, bir çok karar verici için çok yararlıdır. Bu, kâr gibi bazı kritik ölçülerin belli bir seviyeye gelmesi olasılıklarının bilgisini sağlar. Örneğin, karın büyüme hızının sıfır veya yüzde 5 olması olasılıklarının bilinmesi son derece yararlı olacaktır.

İstatistiki Analiz ve Yöneylem Araştırması Modelleri

İyi bir Karar Destek Sistemi, regresyon ve zaman serileri analizleri gibi Yöneylem Araştırması modellerini içermelidir. Bu iki model, satışlar gibi eski verileri, geleceğe yansıtabilmek için kullanılabilir.

Ek Fonksiyonlar

Çok kullanılan bazı hesaplamalar önceden programlanmış olarak Karar Destek Sistemlerinde yerini alabilir. Bunlara örnek olarak vergi oranları hesaplamalarını verebiliriz.

Grafikler

Karar Destek Sistemlerindeki çok önemli özelliklerden biri de grafikler ve diyagramlardır. Sistem, içinde bulunan herhangi bir veriyi grafik haline dönüştürebilmelidir.

Donanım Kapasitesi

Karar Destek Sistemleri büyük sunucu veya mainframe şeklindeki bilgisayarlara kurulabildiği gibi küçük kişisel bilgisayarlara da kurulabilmektedir. Yeni bir eğilim, Karar Destek Sistemlerini kişisel bilgisayarlar ve mainframe' lerde birlikte kullanmaktır. Eğer veriler ve işlemler daha küçükse kişisel bilgisayarlar ana bilgisayara bağlıdırlar ve gerekli veriyi buradan alarak işlerler. Eğer veri ve işlem boyutları çok büyükse Karar Destek Sistemlerinin bir kısım işlemleri ana bilgisayarlarda gerçekleşebilir.

Veri Tabanları ve Yabancı Dosyalar

Karar Destek Sistemlerinin, organizasyonun içinde bulunan dosyalara erişmesi gereksinimi vardır. Karar Destek Sistemlerinin bilgi üretmek için kullandığı verilerin çoğu bu dosyalardadır. Bu erişim, ya veri tabanı yönetim sistemi aracılığıyla, ya da Karar Destek Sistemlerinin bu dosyalara erişim kabiliyetleri sayesinde gerçekleşir. Ayrıca bir çok Karar Destek Sistemlerinde bu dosyalara erişim sağlanıp veriler elde edildikten sonra, bunları kendi içinde dosya olarak saklama kabiliyeti de vardır.

2.6. Karar Destek Sisteminin Kurulması

Bir karar destek sisteminin kurulması aşağıdaki aşamalar takip edilerek gerçekleşmektedir. Bu aşamalar;

- Ön tasarım,

- Tasarım,
- Kurma,
- Geliştirmedir.

2.6.1. Ön tasarım aşaması

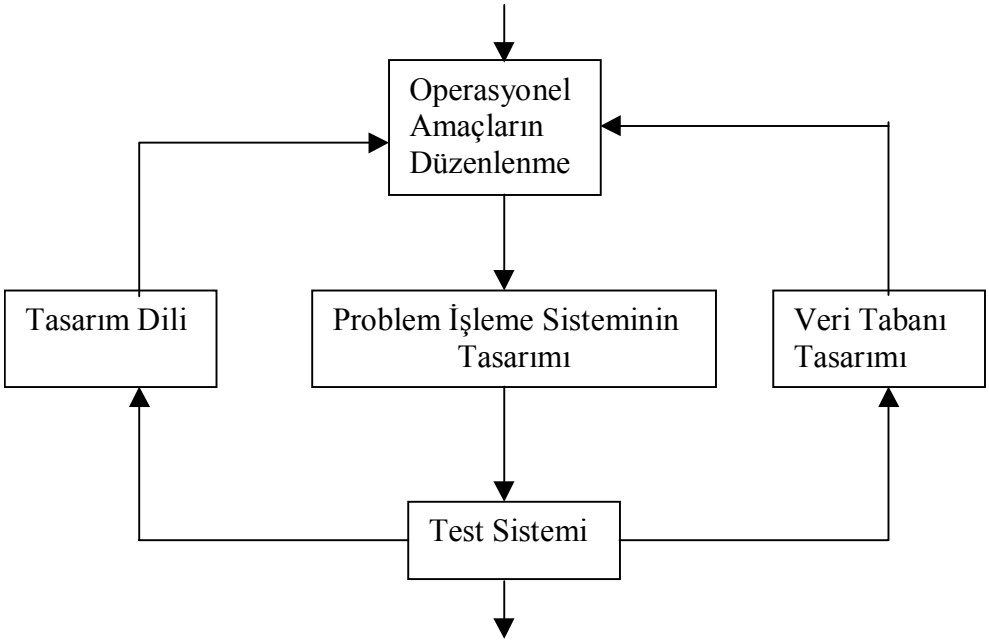
Ön tasarım aşamasında, karar destek çalışması için amaçların belirlenmesi gerekmektedir. Bu aşamada karar destek sistemini yönlendirecek amaçlar ayrıntılı olarak tanımlanmalıdır. Projenin sağlıklı bir şekilde sonuçlandırılabilmesi, problemle ilgili anahtar kararların doğru alınmasına bağlıdır. Anahtar kararlar doğrultusunda kullanılacak kaynakların belirlenmesi, sistem için marjinal yarar sağlayacak bilgilerin elde edilmesi önem taşımaktadır. Ön tasarımın son aşamasında belirlenen problemin yapısına uygun karar modellerinin seçilmesi gerekmektedir. Karar destek sisteminin kurulmasına ilişkin ön tasarım aşaması dört adımdan oluşur. Bunlar;

- Karar destek çalışmaları için amaçların tanımı,
- Mevcut kaynakların belirlenmesi,
- Anahtar kararların belirlenmesi,
- Modellerin tanımlanmasıdır.

2.6.2. Tasarım aşaması

Tasarım aşamasında bir önceki aşama olan ön tasarımda belirlenen amaçlar operasyonel özelliklerine göre düzenlenmektedir. Sistem amaçlarının güncel şartlarla sağlanabilecek şekilde belirlenmesi önem taşımaktadır. Bir sonraki adım ise belirlenen amaçlar doğrultusunda verileri işleyecek olan problem çözme sisteminin

tasarımıdır. Burada dil arabiriminin seçimi önem taşımaktadır. Seçilen dilin işlemsel olamayan bir dilden seçilmesi, tüm kullanıcılar tarafından kolayca kullanılmasını ve gerektiğinde esnek bir şekilde yönlendirilebilmesini sağlamaktadır. Problem çözme sisteminin başarılı olması, seçilen modellerle veri tabanından alınan verilerin yazılım dili aracılığıyla uyum içerisinde karar destek sistemini desteklemesine bağlıdır. Bu birleşmeden sonra sistem, tasarım aşamasının son adımı olan testten geçirilmektedir. Testten alınan sonuçların tutarlılığı sistemi kurulma aşamasına geçirmektedir. Şekil 2.1'de tasarım aşamalarının akışı gösterilmiştir.



Şekil 2.1. KDS tasarım aşamaları (1)

2.6.3. Kurma aşaması

Bu aşamada, daha önceki iki aşamada toplanan bilgiler ışığında sistemin kurulmasına çalışılacaktır. Sistemin kurulması aşamasında karar destek sistemi ile kullanıcının etkileşiminin ne şekilde sağlanacağı önem taşımaktadır. Kullanıcının sistem tasarımına aktif bir şekilde katılması, sistem kurulurken; sistemi tamamen tanımasını sağlayacaktır. Kullanıcının sistemi benimsemesi, sistemi organizasyon ve kişisel ihtiyaçları için kullanmasını sağlayacaktır. Böylelikle kullanıcı, güncel işlerinde çok fazla bir değişiklik yapmaksızın bütün işlerinde karar destek sistemini kullanmasını

sağlayacaktır. Karar destek sistemi için ideal durum, kullanıcının karar destek sistemini başlatması ve ara geçişler olmaksızın bilgisayar sistemi ile yönetmesidir.

2.6.4. Geliştirme aşaması

Karar destek sistemi yapısı gereği çeşitli uygulamaların geliştirilmesine açıktır. Bu yapı kullanıcının ve/veya organizasyonun istekleri doğrultusunda geliştirilebilmektedir. Gelişmenin sağlıklı bir şekilde sağlanabilmesi bazı kriterlere bağlıdır. Bu kriterler kurulan karar destek sisteminin kurulduğu işletmeye göre farklılık göstermektedir. Gelişme, bileşenlerin tam olarak oturtulmasına bağlıdır. Bu bileşenlerin birincisi düzenlemelerin tanımlanması, ikincisi ise bu düzenlemeye karşı sistemde meydana gelen değişmelerin tanımlanmasıdır. Son olarak da düzenlemelerin sağlanıp sağlanamadığının ölçüleceği test sisteminin geliştirilmesi gerekmektedir.

3. GEZGİN SATICI ve ARAÇ ROTALAMA PROBLEMLERİ

3.1. Gezgin Satıcı Problemi

3.1.1. Gezgin satıcı problemi tanımı

Gezgin Satıcı Problemi, üzerinde en geniş çalışılmış kombinatoral optimizasyon problemidir. GSP, bir satıcının bütün ziyaret noktalarına uğrayıp, yine başladığı yere dönmesini ve bir kere ziyaret ettiği noktadan bir daha geçmemesini ve dolaştığı toplam mesafenin minimum olmasını amaçlar. GSP, $G(N,A)$ grafı üzerinde tanımlanır. Bu problemde $d_{ij,}$ a_{ij} arkının ilişkili olduğu maliyeti (uzunluk, mesafe) ifade eder.

Hamilton Turu

Hamilton turu, $G(N,A)$ grafında $c_i \in N$ düğümlerinin her birini sadece bir defa içeren bir kapalı çevrimdir. Gezgin Satıcı Turu ise G grafı içinde d_{ij}' ye göre en az uzunluğu olan Hamilton Turudur.
GSP eğer $d_{ij} = d_{ji} \forall c_i, c_j \in N$ ise simetrik; aksi halde asimetrik olarak adlandırılır. Ayrıca d_{ij} maliyetleri de *üçgen eşitsizliği* diye adlandırılan $d_{ik} + d_{kj} \geq d_{ij} \forall c_i, c_j, c_k \in N$ şartını sağlamalıdır (5).

3.1.2. Gezgin satıcı problemi tamsayı programlama formulasyonu

GSP' nin (0-1) tamsayı programlama ile tanımlanan matematiksel modeli aşağıdaki gibidir.

Parametreler:

S : alt tur engelleme kısıtı için düğüm kümesi,

N : müşteri sayısı,

d_{ij} : i ve j müşterileri arasındaki uzaklık,

Değişkenler:

$x_{ij, i \neq j}$: araç i ' den j ' ye gidiyorsa 1, yoksa 0 olmak üzere;

Amaç Fonksiyonu

$$\text{Min Z} = \sum_{\forall ij, i \neq j} d_{ij}.x_{ij} \qquad [2.1]$$

s.t.

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \qquad i = 1,.....,n \qquad [2.2]$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad j = 1,.....,n \qquad [2.3]$$

$$\sum_{\substack{n_i, n_j \in S \\ i \neq j}} x_{ij} \leq |S| - 1 \qquad \forall S \subset N; \;\; 2 \leq |S| \leq n-2 \qquad [2.4]$$

$$x_{ij} = 0 - 1 \qquad \forall i, j = 1,....n; i \neq j \qquad [2.5]$$

Yukarıdaki kısıtlardan Eş.2.2 ve Eş.2.3 *derece kısıtlarıdır*: Bunlar G grafındaki her düğüme sadece bir kere girilmesini ve her düğümden sadece bir defa çıkılmasını sağlar. Eş.2.4 ise *alt tur eleme kısıtıdır*: Bu kısıt G grafınındaki S bağlı düğümler alt kümesinde sadece bir tane tur olmasını sağlar. Eş.2.5 ise x_{ij} 0-1 değişkenini ifade eder. Burada eğer x_{ij} = 1 ise, gezgin satıcı a_{ij} arkını kullanıyor, x_{ij} =0, ise a_{ij} arkını kullanmıyor demektir (5).

3.1.3. GSP' yi optimal çözen algoritmalar

GSP optimal olarak daha çok *dal-sınır* algoritması ile çözülmektedir. Dal-sınır algoritması öncelikle bazı kısıtların gevşetilmesiyle ve daha sonra fizibilitenin yeniden kazanılmasıyla GSP' ye uygulanır. Dal-sınır algoritmasının kalitesi gevşetme ile elde edilen alt-sınırın kalitesi ile doğrudan ilgilidir. GSP için başlangıç alt tur eleme kısıtının gevşetilmesiyle elde edilir. Sonuç problem, Carpaneto v.d. (1988)' in $O(n^3)$ zamanda çözülebileceğini gösterdiği standart atama problemidir. GSP' nin atama probleminde gevşetme ile dal-sınır algoritması geliştiren araştırmacılar; Eastman (1958), Little v.d. (1963), Shapiro (1966), Murty (1968), Bellmore ve Malone (1971), Garfinkel (1973), Smith v.d. (1977), Carpaneto ve Toth (1980), Balas ve Christofides (1981)'dir (5).

3.1.4. GSP için sezgisel yaklaşımlar

GSP' nin NP-hard bir problem olması nedeniyle, çözüm yöntemlerinde deterministik algoritmaların kullanılması, ya çözüm sürecinin çok uzun olmasına yada bilgisayar kapasitelerinden dolayı çözüm algoritmalarının çalışmamasına neden olmaktadır. Bu nedenlerden ötürü GSP çözümü için sezgisel yöntemler geliştirilmiştir. Bu sezgiseller GSP' de ortalama durumlar için iyi çözümler üretmek için tasarlanmışlardır. GSP sezgiselleri 3 kategoriye ayrılır. Bir turu baştan oluşturmak için, *inşaa sezgiselleri*; bilinen bir turu geliştirmek için, *geliştirme sezgiselleri*; alt tur bulunan durumlarda *alt tur birleştirme sezgiselleri*. Burada sadece kullanımı yaygın olan inşaa sezgiselleri ve iyileştirme sezgisellerinden bazıları hakkında genel bilgi verilecektir.

İnşaa sezgiselleri

Bir sonraki iyileştirme algoritmalarına en avantajlı uygun turu oluşturmayı amaçlarlar. En yakın komşuluk algoritması ve ekleme algoritması önemli inşaa sezgiselleridir.

En yakın komşuluk algoritması (5)

Metot, N^*; tura bağlı düğümler kümesini ifade etmek üzere aşağıdaki şekilde ifade edilir.

Adım 1: $d_{jl} = \min_{n_v \notin N^*} \{d_{jv}\}$ olan bir c_l düğümünü bul ve $N^* := N \cup \{c_l\}$ ve $c_j := c_l$ atamalarını yap.

Adım 2: Eğer $N^* = N$?; Evet ise: a_{ji} 'yi tura ekle ve dur; hayır ise: Adım 1' e git.

Ekleme Algoritması (5)

Diğer bir tip inşaa sezgiseli de ekleme algoritmasıdır. Genel ekleme algoritması keyfi bir alt tur olan (c_i, c_j)' den başlar. N^*, turdaki düğümlerin kümesini belirtir. Başlangıçta $N^* = \{c_i, c_j\}$ değerini alır. Ayrıca, hazırda bulunan turun içerdiği ark kümesini belirtmek üzere A^* kümesi tanımlanmıştır. Başlangıçta $A^* = \{a_{ij}, a_{ji}\}$ olarak algoritma başlamaktadır. Bu başlangıç atamaları yapıldıktan sonra algoritma aşağıdaki gibi devam eder.

Adım1: Hazırdaki tur bütün düğümleri içeriyor mu? Evet: Tur bulunmuştur, DUR. Hayır: Adım 2'ye git.

Adım2: $c_k = \arg\min_{\substack{c_i \notin N^* \\ a_{ij} \in A^*}} \{d_{il} + d_{lj} - d_{ij}\}$ olan bir c_k düğümü bul. $N^* := N \cup \{c_k\}, A^* := A \cup \{a_{ik}, a_{kj}\}$ atamalarını yap ve Adım 1' e dön.

İyileştirme algoritmaları

İnşaa algoritmalarıyla oluşturulmuş Hamilton döngülerinin amaç fonksiyon değerleri tatmin edici değildir. Bunların çeşitli yöntemlerle iyileştirilmesine ihtiyaç duyulur.

GSP iyileştirme algoritmalarından en önemlileri 2-opt değişim ve 3-opt değişim algoritmalarıdır.

2-opt değişim algoritması (6)

Bu iyileştirme algoritması, Hamilton döngüsünün kendini kesmesi durumunda toplam döngü uzaklığının kolaylıkla azaltılabileceği temeline dayanır. Birbirini kesen iki çift düğümün arkını silip, birbirini kesmeyen iki ark oluşturmak Hamilton döngüsünün amaç fonksiyonunda azalmaya yol açar. Eğer $a_{ij} - a_{kl}$ arkları kesişiyorsa, bunları $a_{ik} - a_{jl}$ ile değiştirmek Hamilton döngüsünün uzunluğunu azaltır.

Adım 1: Mevcut Hamilton döngüsünü Ψ olarak adlandır.

Adım 2: Her c_i düğümü için "hata" etiketi elde edene kadar aşağıdakileri tekrarla!

Adım 2.1: Bir c_i düğümü seç.

Adım 2.2. Döngüde c_i düğümü ve onu izleyen düğüm arasındaki köşelerde 2-opt hareketlerini test et. Eğer döngünün uzunluğu bu yöntem ile kısaltılabiliyorsa, bu hareket ile en iyisini seç; yoksa, c_i düğümünü "hata" diye etiketle.

Adım 3: Ψ' yi döndür.

3-opt sezgiseli (6)

Eldeki Hamilton döngüsünün değişiminde daha çok esneklik için döngü iki parça yerine (2-opt) üç parçaya bölünmekte, sonuç yollar daha uygun olarak birleştirilmektedir. Algoritma aşağıdaki şekilde işler.

Adım 1: Mevcut Hamilton döngüsünü Ψ olarak adlandır.

Adım 2: Her $c_i \in N$ için bir $N(i)$ düğüm kümesi tanımla!

Adım 3: Her c_i düğümü için "hata" etiketi elde edinceye kadar aşağıdaki adımları tekrarla!

Adım 3.1: Bir c_i düğümü seç

Adım 3.2: Her biri $N(i)$' de en az bir bitiş noktası olan üç köşeyi eleyen 3-opt hareketlerinin gerçekleştirme ihtimallerini test et! Eğer bu yolla bir azalma gerçekleşiyorsa bu hareketi en iyi olarak seç; yoksa, c_i düğümünü "hata" olarak etiketle!

Adım 4: Ψ' yi döndür.

3.2. Araç Rotalama Problemi ve Türleri

3.2.1. Klasik araç rotalama problemi

Klasik veya kapasitelendirilmiş diye adlandırılan araç rotalama problemi (ARP) şöyle tanımlanır: Q_k sabit kapasiteli araçların i müşterilerinden gelen $q_i\ (i = 1,....,n)$ taleplerini karşılamak üzere, merkez bir depodan $(i = 0)$ yapacakları dağıtımlar olarak adlandırılabilir. Problem, müşteriler i ve j $(i, j = 1,....,n \wedge i \neq j)$ müşteri arasındaki d_{ij} uzaklıklarının bilinmesi durumunda ARP' nin amacı her bir aracın yapacağı taşıma miktarının Q_k' yi aşmayacak şekilde, tüm araçlar tarafından müşterilere gerçekleştirilecek olan seyahat turlarının toplam mesafesini enazlanmasıdır. Burada her müşteriye bir defa servis yapılması zorunludur ve siparişlerin bölünmesi mümkün değildir (7).

Bu problemin çözümü için başlangıçta basit bir tamsayı programlama modelinden yararlanılmaktadır. Klasik ARP' nin çözümü için model şu şekilde tanımlanır.

Parametreler:

Q_k : araç kapasitesi,

N : müşteri veya durak sayısı,

q_i : $i(i>0)$ müşterisinin talebi,

d_{ij} : i ve j müşterileri arasındaki uzaklık,

Değişkenler:

$x_{ij,\,i\neq j}$: araç i' den j' ye gidiyorsa 1, yoksa 0 olmak üzere, ($i,j\in\{0,.....,n\}$ ve "0" başlangıç deposu iken)

Amaç Fonksiyonu

$$Min\sum_{i=0}^{n}\sum_{j=0,i\neq j}^{n}d_{ij}x_{ij} \qquad [2.6]$$

s.t.

$$\sum_{\substack{i=1\\ i\neq j}}^{n}x_{ij}=1, \qquad \forall j, j\in\{1,.....,n\}, \qquad [2.7]$$

$$\sum_{\substack{j=1\\ j\neq i}}^{n}x_{ij}=1, \qquad \forall i, i\in\{1,.....,n\}, \qquad [2.8]$$

$$\sum_{i=0}^{n}\sum_{\substack{j=0,\\ j\in S}}^{n}x_{ij}\leq |S|-1, \qquad [2.9]$$

$$\sum_{i=0}^{n} \sum_{\substack{j=0, \\ j \in T}}^{n} x_{ij} \leq |T| - n, \qquad [2.10]$$

Burada Eş.2.7 ve Eş.2.8 kısıtları ziyaret edilen müşteriden ayrılma kısıtlarıdır. Eş.2.9 kısıtı depodan başlamayan ve depoda bitmeyen turları elemekte kullanılır. Son olarak, Eş.2.10 kısıtı araçlardaki yük durumunu kontrol etmektedir. Bu kısıt, depo dahil olmak üzere her T müşteri kümesine eklenir. Bu kümelerin her biri $\sum_{i \in T} q_i < Q_k$ şartını sağlamaktadır. Ayrıca n ise aşırı yüklemeyi engellemek için T kümesinden çıkarılması gereken minimum sayıda müşteri sayısıdır (8).

3.2.2. Periyodik araç rotalama problemi

Periyodik Araç Rotalama Problemi (PARP), planlama periyodunu M güne çıkararak klasik ARP' yi genelleştirmektedir. $M - gün$ periyodunda, her müşteri belirtilen periyotta en az bir kere ziyaret edilmelidir. Klasik PARP' e homojen araç filosundan oluşur ve bu araçlar bir grup müşteriyi ziyaret edip tekrar başlangıç deposuna dönmek zorundadır. Her aracın aşılamayacak sabit bir kapasitesi ve her müşterinin de sadece bir ziyaret ile sadece bir araç ile tamamen karşılanması gereken günlük bilinen talepleri mevcuttur. Planlama periyodu M gündür. $M = 1$ olduğunda PARP, klasik ARP' nin bir örneği olur. PARP' deki her müşteri periyot içinde se_i defa ziyaret edilmelidir ($1 \leq se_i \leq M$) (7).

PARP' nin matematiksel modeli aşağıdaki gibidir (9).

Parametreler:

Q_k : araç kapasitesi,

q_i : i müşterisinin talebi ($q_0 = 0$),

k : araç indisi,

m : toplam araç sayısı,

rs_k : k aracına izin verilen rota süresi,

s_i : servis zamanı ($s_0 = 0$),

t : planlama ufku,

C_i : i müşterisine servis yapılacak kombinasyonlar,

se_i : servis sıklığı,

l : ziyaret günü,

d_{ijkl} : k aracının i ' den j ' ye l gününde gitme süresi,

r : kombinasyon indisi $(r \in C_i)$,

Değişkenler:

a_{rl} : l günü r kombinasyonuna aitse 1, yoksa 0,

y_{ir} : $r \in C_i$ kombinasyonu i müşterisine atanmışsa 1, yoksa 0,

x_{ijkl} : k aracı i ' den j ' ye l gününde giderse 1, yoksa 0 olmak üzere;

Amaç Fonksiyonu

$$\text{Min} \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{k=1}^{m} \sum_{l=1}^{t} d_{ijkl} x_{ijkl} \qquad [2.11]$$

s.t

$$\sum_{r \in Ci} y_{ir} = 1 \qquad (i = 1,.....,n) \qquad [2.12]$$

$$\sum_{j=0}^{n} \sum_{k=1}^{m} x_{ijkl} - \sum_{r \in Ci} a_{rl}\, y_{ir} = 0 \qquad (i = 1,....,n;\ l = 1,...,t) \qquad [2.13]$$

$$\sum_{i=0}^{n} x_{ihkl} - \sum_{j=0}^{n} x_{hjkl} = 0 \qquad (h = 0,.....,n;\ k = 1,.....,\ m;\ l = 1,....,t) \qquad [2.14]$$

$$\sum_{j=1}^{n} x_{0\,jkl} \le 1 \qquad (k = 1,.....,\ m;\ l = 1,....,t) \qquad [2.15]$$

$$\sum_{i=0}^{n} \sum_{j=0}^{n} q_i . x_{ijkl} \le Q_k \qquad (k = 1,.....,\ m;\ l = 1,....,t) \qquad [2.16]$$

$$\sum_{i=0}^{n} \sum_{j=0}^{n} (d_{ijkl} + s_i) x_{ijkl} \le rs_k \qquad (k = 1,.....,\ m;\ l = 1,....,t) \qquad [2.17]$$

$$\sum_{ci \in S} \sum_{cj \in S} x_{ijkl} \le |S| - 1 \qquad (k = 1,.....,\ m;\ l = 1,....,t;\ S \subseteq N \setminus \{0\};|S| \ge 2) \qquad [2.18]$$

$$x_{ijkl} \in \{0,1\} \qquad (i = 0,....., n;\ j = 0,....., n;\ k = 1,....., m;\ l = 1,...., t) \qquad [2.19]$$

$$y_{ir} \in \{0,1\} \qquad (i = 1,....., n;\ r \in C_i) \qquad [2.20]$$

Eş.2.13 kısıtı müşterilerin atanmış kombinasyonlarla ilgili günlerde ziyaret edilmesini garanti ederken Eş.2.12 kısıtı, uygun bir ziyaret kombinasyonunun tüm müşterilere atanmasının gerektiğini vurgular. Eş.2.14 kısıtı araçların müşterileri ziyaret ettiği günlerde, o müşteriden ayrılması sağlar. Eş.2.15 her aracın bir günde sadece bir defa kullanılmasını, Eş.2.16 ve Eş.2.17 kısıtları araçların kapasite ve rota

limitlerini gösterir. Eş.2.18 ise alt tur eleme kısıtıdır. (Daha ayrıntılı bilgi için Cordeau (1997) (9)).

3.2.3. Çok depolu araç rotalama problemi

Çok Depolu Araç Rotalama Problemi (ÇDARP), bir günlük bir planlama ufku içerisinde, araçların tek bir depo yerine birden fazla depodan dağıtım yaptığı araç rotalama problemi türüdür. Bu problemde her araç, atanmış olduğu depodan turuna başlamak ve tur bitiminde aynı depoya geri dönmek zorundadır. ÇDARP, PARP probleminin özel bir durumudur. PARP modelindeki gün terimleri, depoları ifade edecek şekilde değiştirilerek ÇDARP formüle edilebilir. Bunun için t adet depo olduğunu varsayar ve i müşterisine servis yapacak kombinasyonlar kümesi olan $C_i = \{\{1\},.....,\{t\}\}$ $(i = 1,....,n)$ şeklinde yazılırsak, d_{0ikl} ve d_{i0kl} terimleri l deposu ile i müşterisi arasında k aracını kullanarak seyahat etmenin maliyeti olur (9).

3.2.4. Karışık Filolu Araç Rotalama Problemi

Karışık Filolu Araç Rotalama Problemi (KFARP), değişik kapasiteli heterojen araç filolarının bulunduğu taşıma sistemleri için geliştirilmiş bir ARP tipidir. $N = \{0,....,n\}$ düğümler kümesini, $A = \{(i,j), i, j \in N\}$ ise ark kümesini belirttiği $G = (N, A)$ yönsüz grafında c_0 düğümü araç filosunun bulunduğu depoyu geri kalan düğümler ise müşterileri temsil eder. Her c_i müşterisinin kendine ait negatif olmayan bir q_i talebi vardır. k tipinde her aracın kapasitesi Q_k, sabit maliyeti F_k ve uzaklık başına değişken maliyeti ise α_k' dir. Her i ve j müşteri çiftinin arasındaki uzaklık d_{ij} ile ifade edildiğinde k tipindeki bir aracın seyahat maliyeti $c_{ij} = d_{ij} \times \alpha_k$'ye eşit olur. Her rota depodan başlamalı ve depodan bitmelidir. Bir rotanın kapasitesi, o rotaya atanan k aracının Q_k kapasitesini aşmamalıdır. Her müşteri sadece bir defa ziyaret edilmelidir. KFARP' nin amacı, heterojen araçların ve eşlendiği rota kümelerinin optimal karışımını, toplam değişken ve sabit maliyeti minimum yapacak şekilde bulmaktır (10).

3.2.5. Yer bağımlı araç rotalama problemi

Yer Bağımlı Araç Rotalama Probleminde (YBARP), heterojen araç filosu müşterilere servis yapar. Tipik olarak birkaç araç tipi bulunmaktadır. Bu problemde araç tipi ile müşteriler arasında uyumluluk vardır. YBARP' de bir ana depo, N tane müşteri ve her müşteriye uyumlu araç tiplerinin kümesi vardır. Bu problem aşağıdaki kısıtlar altında araçların seyahat ettiği toplam mesafeyi minimize etmektir.

- Bir müşteri için sadece bir tip araç seçilebilir ve o müşteriye sadece o araç ile servis yapılabilir.

- Bir müşteriden diğer müşteriye aynı araç tipine atanmadıkça seyahat edilmez.

- Bir araç turu depoda başlamalı ve depoda bitmelidir ve araç bir defa kullanılmalıdır.

- Araç ziyaret ettiği müşteriyi terk etmelidir.

- Araç kapasiteleri ve kullanılabilir araç sayıları geçilmemelidir.

YBARP çok aşamalı rotalama problemi olarak karakterize edilebilir. Birinci aşamada her müşteri için uygun araç tipleri belirlenir. İkinci aşamada kapasitelendirilmiş ARP her araç tipi için çözülür. Üçüncü aşamada ise her rota için GSP çözülür (11).

4. ZAMAN PENCERELİ ARAÇ ROTALAMA PROBLEMİ

Zaman Pencereli Araç Rotalama Problemi (ZPARP), depodan hareket eden araçların müşterileri belli zaman aralığı içinde ziyaret etme zorunluluğu olan (zaman penceresi kısıtı) özel bir araç rotalama problemi türüdür. Yapısı nedeniyle okul otobüsü rotalama, posta, gazete dağıtımı, akaryakıt dağıtımı, tam zamanlı üretim için satıcı dağıtımı, güvenlik devriyesi kontrolleri, kentsel atık toplama ve zincir mağaza dağıtım lojistiği gibi gerçek hayat problemlerine daha uyumludur. Bu nedenlerden dolayı diğer araç rotalama problemlerine göre hakkında daha çok araştırma yapılmıştır (11-12).

ZPARP' de, merkez depoya yerleştirilmiş kapasiteleri (Q_k) bilinen V araç kümesine ait araçlar, coğrafi olarak dağılmış müşterilere servis yapmaktadır. Buradaki her i müşterisinin q_i miktarı talebi vardır ve her müşteri taleplerinin karşılanması sırasında s_i kadar servis zamanına ihtiyaç duyarlar. Ayrıca tüm müşterilere belirtilen bir zaman penceresi kısıtı $[e_i, l_i]$ içinde servis yapılması gerekmektedir. ZPARP' de i ve j müşterileri arasındaki seyahat zamanı (t_{ij}) ve uzaklığa bağlı (d_{ij}) olarak değişen C_{ij} maliyeti söz konusudur. Her rota merkez depodan başlar ve yine merkez depoda son bulur. ZPARP' de diğer ARP türlerinde olduğu gibi Q_k araç kapasitesi aşılmamalıdır. Klasik ARP' den farklı olarak, araçlar her i müşterisine e_i zamanından önce, l_i zamanından sonra servis yapmamalıdır (12).

4.1. Zaman Pencereli Araç Rotalama Problemi Matematiksel Formulasyonu

ZPARP, benzer tip araçlardan oluşmuş bir araç kümesi, bir merkez depo düğümü, müşteri düğümleri kümesi ve bu düğümleri bağlayan bir şebekeden oluşur.

Şebekede, c_0 düğümü merkez düğümü ve $c_i \in \{1,2,....,N\}$ müşteri düğümleri olmak üzere toplam $N+1$ düğüm ve K adet araç ($V \in \{1,2,...,K\}$) vardır. Her rota depodan başlar, müşteri düğümlerini ziyaret eder ve tekrar depoya geri döner. Şebekedeki rota

sayısı aynı zamanda kullanılan araç sayısını da gösterir. Bir araç sadece bir rotaya hizmet eder. c_{ij} maliyeti ve t_{ij} seyahat süreleri şebekedeki her ark ile ilişkilendirilmiştir. Şebekedeki bir düğüm sadece bir araç tarafından ziyaret edilebilir. Her aracın Q_k yükleme kapasitesi ve her müşterinin ise q_i kadar taşınmasını istediği talebi mevcuttur. Q_k kapasitesi, k aracı tarafından ziyaret edilen düğümlerdeki taleplerin toplamından büyük veya eşit olmalıdır, araçlar aşırı yüklenmemelidir. Zaman penceresi kısıtı bir i müşterisi için önceden tanımlanmış en erken ulaşma (e_i) ve en geç ulaşma (l_i) zaman aralığıdır. Yükleme/boşaltma faaliyetlerinin gerçekleştirilebilmesi için her müşteriye ait bir s_i servis zamanı bulunmaktadır. Araçların, kendi rotalarını izin verilen toplam rota zamanı içinde tamamlaması gerekmektedir. Bu şart deponun zaman penceresi için gereklidir. ZPARP' de üç tip karar değişkeni vardır. $x_{ijk}=1$ ise k aracı i müşterisinden j müşterisine seyahat eder; $x_{ijk}=0$ ise seyahat etmez $(i, j \in \{0,1,2,.....,N\}; k \in \{1,2,.....,K\}; i \neq j)$. t_i karar değişkeni, bir aracın i müşterisine ulaştığındaki zamanı verir. w_i ise i müşterisinde aracın bekleme durumunda bulunduğu süredir. ZPARP' nin genel amacı bütün kısıtları sağlayarak, toplam seyahat maliyetini minimize etmektir.

Parametreler

K : toplam araç sayısı

N : toplam müşteri sayısı

c_i : i. müşteri düğümü; $i \in \{1,2,...,N\}$

c_0 : merkez depo düğümü

y_i : rastgele bir sayı

d_{ij} : i ve j düğümleri arasındaki öklid uzaklığı

C_{ij} : i ve j düğümleri arasındaki seyahatten doğan maliyet

t_{ij} : i ve j düğümleri arasındaki seyahat süresi

q_i : i düğümünün talebi

Q_k : k aracının kapasitesi

e_i : i düğümüne en erken varış zamanı

l_i : i düğümüne en geç varış zamanı

s_i : i düğümündeki servis zamanı

rs_k : k aracı için izin verilen maksimum rota zamanı

Karar değişkenleri

t_i : i düğümüne varış zamanı,

w_i : i düğümünde bekleme zamanı,

x_{ijk} : k aracı, i ' den j ' ye gidiyorsa 1, yoksa 0; $i \neq j \quad i, j \in \{0,1,2,.....,N\}$ olmak üzere;

Amaç fonksiyonu

$$MIN \sum_{k=1}^{K} \sum_{i=0}^{N} \sum_{j=0}^{N} c_{ij} x_{ijk} \qquad [4.1]$$

s.t.

$$\sum_{k=1}^{K}\sum_{j=1}^{N} x_{ijk} \le K \qquad i = 0 \text{ için} \qquad [4.2]$$

$$\sum_{j=1}^{N} x_{ijk} = \sum_{j=1}^{N} x_{jik} \le 1 \qquad i = 0 \text{ ve } k \in \{1,...,K\} \text{ için} \qquad [4.3]$$

$$\sum_{k=1}^{K}\sum_{j=0}^{N} x_{ijk} = 1 \qquad i \in \{1,...,N\} \text{ için} \qquad [4.4]$$

$$\sum_{k=1}^{K}\sum_{i=0}^{N} x_{ijk} = 1 \qquad j \in \{1,...,N\} \text{ için} \qquad [4.5]$$

$$\sum_{i=0}^{N} q_i \sum_{j=0}^{N} x_{ijk} \le Q_k \qquad k \in \{1,...,K\} \text{ için} \qquad [4.6]$$

$$\sum_{i=0}^{N}\sum_{j=0}^{N} x_{ijk}(t_{ij} + s_i + w_i) \le rs_k \qquad k \in \{1,...,K\} \text{ için} \qquad [4.7]$$

$$t_0 = w_0 = s_0 = 0 \qquad [4.8]$$

$$\sum_{k=1}^{K}\sum_{i=0}^{N} x_{ijk}(t_i + t_{ij} + s_i + w_i) = t_j \qquad j \in \{1,...,N\} \text{ için} \qquad [4.9]$$

$$e_i \le (t_i + w_i) \le l_i \qquad i \in \{0,...,N\} \text{ için} \qquad [4.10]$$

Genel ZPARP modelinde Eş.4.2 kısıtı depodan çıkacak olan rotaların maksimum K tane olmasını, Eş.4.3 kısıtı her rotanın depoda başlayıp depoda bitmesini, Eş.4.4 ve Eş.4.5 kısıtları her müşterinin sadece bir araç tarafından bir defa ziyaret edilmesini, Eş.4.6 kapasite kısıtını, Eş.4.7 kısıtı maksimum seyahat zamanı kısıtını, Eş.4.8-4.10 kısıtları ise zaman penceresi kısıtlarını tanımlamaktadır (13).

4.2. ZPARP Literatürü

ZPARP deterministik–tam çözüm algoritmaları ile çözülebilmekle beraber, NP-zor türünde bir problem olması nedeniyle bu çözümler uzun işlem zamanı gerekmektedir (13). Bu nedenle yerel arama algoritmaları ve/veya meta-sezgiselleri kullanarak yaklaşık çözüm arayan bir çok çalışma da literatürde mevcuttur. Bu üç türde yapılan araştırmalar izleyen bölümlerde tanıtılmaktadır.

4.2.1. Deterministik – tam çözüm algoritmaları

Deterministik-tam çözüm algoritmaları ZPARP için optimum çözüm bulabilen yöntemlerdir. Ancak çözüm süresi ve bilgisayar kapasiteleri göz önüne alındığında büyük boyutlu problemlerde yetersiz kalmaktadır.

Bu konulardaki ilk çalışmalardan birini Desrosiers v.d. (1986) yapmıştır. m-GSP probleminin zaman pencere kısıtlı durumu için çalışma yapmışlardır. Çözümlerinde simpleks ve dal-sınır algoritması kullanmışlardır. Küçük boyutlu problemlerde optimal sonuca ulaşmışlardır (14). Kolen v.d. (1987)'de, ZPARP çözümü için dal-sınır algoritması geliştirmişler. 6-15 müşteri arasında değişen 9 problemin optimal olarak çözümünü bulmuşlardır (15). Min (1991) çalışmasında karışık tamsayılı amaç programlama modeli kullanılarak, gecikmelerden kaynaklanan memnuniyetsizlikler en aza indirilmeye çalışılmıştır (16). Desrochers v.d. (1992), Küme kaplama formulasyonu için bir sütun yaratma yaklaşımı kullanan bir optimizasyon algoritması geliştirmişler. Bu algoritma literatürde belirtilmiş optimal algoritmalara göre 6 kat daha büyük problemleri optimal olarak çözme yeteneğine sahiptir (17). Kohl ve Madsen (1997)' in geliştirdikleri metot, kısıt kümesine yapılan langranj gevşetmesine dayanmaktadır. Ana problemde amaç, optimal langranj çarpanlarını bulmak, alt problemde ise zaman pencereli ve kapasite kısıtlı en kısa yol problemini çözmektir. Optimal çarpanlar yığın metodu kadar sub-gradiyentlerin faydaları metodu kullanılarak bulunmuştur. Metot, 100 müşteriye kadar olan Solomon (1987) problemleri üzerinde çözülmüş, çözümü olmayan birkaç problemin çözümüne ulaşılmıştır (18). Kallehauge v.d. (2001)' de, ZPARP ile bağlantılı

diferansiyellenemeyen optimizasyon yönteminin uygulama sonuçları anlatılmaktadır. ZPARP' nin en kısa yol ayrışması, lagranj çarpanlarının optimal bulunmasını gerektirmektedir. Bu problem konveks ve diferansiyellenemez bir problemdir. Optimal çarpanları bulmak için, bir güvenilir-bölge dengeleyici araçlı kesme-düzlemi algoritmasının geliştirilmesi amaçlanmıştır. Bu algoritma dal-sınır şemada tam sayılı çözümü bulmak için kullanılan Dantzig-Wolfe algoritması ile birleştirilmiştir. Bu yaklaşım Solomon (1987)' un örnek problemleriyle test edilmiş. 14 adet optimal çözülmemiş büyük boyutlu problemlerde sonuca ulaşılmıştır. Çözüm zamanları dikkate değer şekilde azalmıştır (19).

4.2.2. Yerel arama sezgiselleri

Yerel arama sezgiselleri, GSP için geliştirilen algoritmaların ZPARP' ye uyarlanması ile gerçekleştirilmiştir. Bu çalışmalardan, Baker ve Schaffer (1986)' da daha önce GSP çözümünde kullanılan 2-opt ve 3-opt prosedürleri zaman penceresi kısıtı altında incelenmişlerdir. Başlangıç çözümleri, ekleme (insertion1) veya yakın komşuluk (nearest neighborhood) algoritmaları sezgisel ile kurulmuştur. Daha sonra bu çözümler 2-opt ve 3-opt prosedürleri ile iyileştirilmiştir (20). Solomon (1987) çalışmasında, literatürdeki 5 adet sezgisel yaklaşımı ZPARP' ne uygulamışlardır. Bu algoritmalar daha önce GSP için geliştirilmiş tasarruf sezgiseli (savings heuristics), zaman yönelimli en yakın komşuluk sezgiseli (time-oriented, nearest-neighbor heuristic), ekleme sezgiseli (insertion heuristics), zaman yönelimli tarama sezgiseli (time-oriented sweep heuristic)dir. Çalışmanın sonunda ekleme tipi sezgisellerin daha iyi sonuç verdiği anlaşılmıştır (21). Van Landeghem (1988)' in çalışmasında, ZPARP için geliştirilen bir sezgisel anlatılmıştır. Sezgisel Clarke ve Wright Tasarruf metodu üzerine ek kriterler eklenerek bina edilmiştir. Deneyler sonucu bu sezgiselin, saf rotalama sezgisellerinden daha iyi sonuç verdiği görülmüştür (22). Koskosidis ve Powell (1990) ZPARP için bir matematiksel model ve optimizasyon tabanlı sezgisel algoritma geliştirmişlerdir. Model ana üreticiden dağıtım faaliyetlerini değerlendirmek ve optimize etmek için kullanılmıştır. Yöntem 20 müşterili problemlere kadar sonucu rahat bulabilmektedir (23). Ahn ve Shin (1991), ZPARP'de zaman-değişkenli sıkışıklık durumunu incelemişler ve problem için

sezgisel bir algoritma geliştirmişlerdir (24). Salvelsbergh (1992), kenar-değiştirme iyileştirme metodunu (edge-exchange) amaç fonksiyonu rota süresinin minimizasyonu olan ZPARP' ne uygulamışlar, farklı amaç fonksiyonlu 10 rassal ZPARP örneği çözmüşler, ZPARP' de gerçekçi bir amaç fonksiyonu kullanmanın önemini belirtmişlerdir (25). Koskosidis v.d. (1992) ise yumuşak zaman pencereli kısıtlı araç rotalama ve çizelgeleme problemini, karışık tamsayılı ve optimizasyon-tabanlı olarak formüle etmişlerdir (26). Potvin ve Rousseau (1993) çalışmalarında, ZPARP için bir ekleme algoritması tanımlanmıştır. Algoritma, rotaları paralel olarak oluşturmaktadır ve gelecek adayı seçmek için rotalanmamış müşterilere genelleştirilmiş pişmanlık ölçütü kullanmaktadır. Algoritma Solomon (1987) (21) standart örnekleri ile denenmiş ve bazı örnek problemlerde optimuma rastlanmıştır (27). Foisy ve Potvin (1993) araştırmalarında, ekleme (insertion) sezgiselini paralel işlemcilere uygulayarak işlem zamanını kısaltmaya çalışmışlardır. İşlem sürelerinin kullanılan işlemci sayısıyla paralel olduğunu vurgulamışlardır (28). Balakrishnan (1993) ise ZPARP' de, zaman penceresinin uygun ceza fonksiyonu ile aşılabildiği üç basit sezgisel algoritma geliştirmiştir. Burada ceza fonksiyonu, zaman penceresi aşımı sayısının lineer bir fonksiyonudur (29). Bramel ve SimchiLevi (1993), ZPARP' nin genel versiyonunda olasılıklı analiz gerçekleştirmişler. En kötü-durum analizinin aksine, sezgisel algoritmaların ortalama performansları dikkate alınmıştır. Modellerinde zaman pencereli araç rotalama probleminin basitleştirilmiş versiyonu kullanılmıştır. Depo ve müşteriler öklid düzleminin elemanı olarak ele alınmıştır (30). Derigs ve Grabenbauer (1993), k-iyi-ekleme adında bir sezgisel geliştirmişler. Geliştirilen bu sezgisel, düğüm değişimlerinin sonucu iyileştirdiği fikrine dayanmaktadır (31). Russell (1995) çalışmasında ise, tur inşaa ve tur iyileştirme için yerel arama sezgiselleri geliştirilmiştir. Bu çalışmanın öncü yanı küresel tur iyileştirme prosedürünü tur inşaa prosedürüne yerleştirmiş olmasıdır. Çözüm algoritması literatürdeki test problemlerine uygulanmış ve araç filo büyüklüğünü düşürmede daha verimli olduğu görülmüştür (32). Johns (1995), talebin belirsiz olduğu ZPARP çözümü için bir takım sezgiseller geliştirilmiştir. Geliştirilen sezgiseller ziyaretleri, varolan ziyaretlere yakınlık ölçüsünde sıralayarak iyi turlar üretmektedir (33). Shieh ve May (1998) çalışmalarında, ZPARP'nin online versiyonunu çözmek için "on-line", "her zaman" ve yerel arama sezgiseli

kavramlarını bağlayan online (bağlı) optimizasyon - tabanlı sezgisel geliştirmişlerdir (34). Shaw (1998), kısıt programlama teknikleri kullanılarak seçilen müşterin ziyaretlerini yeniden çizelgeleyen geniş komşuluk arama (large neighborhood search – LNS) yöntemi tanımlamışlardır. LNS atölye tipi çizelgelemede kullanılan shufling tekniği ile benzerdir. Bu teknik Adam v.d. (1988)' nin dar boğaz işlerin ilerletilmesi prosedüründen esinlenilmiştir. Dal-sınır algoritmasının müşterilerin yeniden eklenmesi prosedüründe kısıt-tabanlı sınırlı uyumsuzluk arama (limited discreapancy search – LDS) yöntemi kullanılmaktadır (35). Caseau ve Laburthe (1999a) geniş kapsamlı rotalama problemleri için özel bir kurma-iyileştirme algoritması geliştirmişlerdir. Yazarlar, paralel en ucuz ekleme sezgiseli için bir LDS dönüşümü geliştirmişlerdir. Çözüm kuruluş aşamasında, her ekleme (insertion) işleminden sonra 2-opt*, reinsertion ve basit değişim hareketleri çalıştırılmaktadır. Ayrıca bir müşteri için uygun bir ekleme yeri bulunamadığı takdirde ise swap, relocate ve flush-relocate hareketleri kullanılarak, o müşteri için uygun bir yer açılmaya çalışılmaktadır (35). Liu ve Shen (1999), çok araçlı ZPARP çözümü için rota kurma metodu geliştirmişlerdir. Ekleme tabanlı-tasarruf algoritması sunulmuştur. Problem 100 müşteri 24 problem üzerinde test edilmiş. Yaklaşımlarının iyi sonuçlar verdiği görülmüştür (36). Cordone ve Calvo (2001)' nin, ZPARP için geliştirdikleri model, üç basit prosedürün bir kombinasyonudur. Klasik k-opt değişimleri çözümü iyileştiriyor, ad hoc (özel amaç) prosedürü araç sayısını azaltmaya ve ikinci bir amaç fonksiyonu da aramayı yerel optimumdan uzaklaştırmayı amaçlamaktadır (37). Braysy (2003a) tarafından ZPARP çözümü için deterministik yerel arama yöntemleri sunulmuştur. Yeni algoritma üç aşamalı olup, birinci aşamada iki tür kurma algoritmasından biri ile başlangıç çözümü elde edilmekte, ikinci aşamada zincir atma (ejection chain) tabanlı yerel algoritma ile rota sayıları azaltılmakta, son aşamada ise or-opt değişimleri ile rotalar kısaltılmaya çalışılmaktadır. deney sonuçları bu algoritmanın diğer çalışmalara göre hem rekabetçi hem de daha hızlı olduğunu göstermiştir (38).

4.2.3. Meta-sezgiseller

Yerel arama algoritmalarının lokal minimumda takılmasını engellemek ve global optimuma daha yakın sonuç bulmak amacıyla meta-sezgiseller ZPARP' de de çok sık kullanılmıştır. Kullanılan algoritmalara tabu arama, tavlama benzetimi, genetik ve karınca kolonisi algoritmaları örnek verilebilir. Zaman pencereli araç rotalama problemlerine meta-sezgisellerin uygulandığı ulaşılabilen ilk algoritma Thangiah v.d. (1991)' nindir. Thangiah v.d. (1991), ZPARP için bir genetik algoritma geliştirmiştir. GIDEON adlı bu genetik algoritmayı Solomon (1987) örnek problemlerine uygulayarak önceki çözümlerden daha iyi sonuçlar bulmuşlardır (39). Garcia v.d. (1994) çalışmalarında tabu arama algoritmasının paralel uygulamasını anlatmaktadır. Paralel algoritma senkronizedir ve çoklu-talimatlı, çok-verili bilgisayar mimarisi üzerinde çalışmaktadır. Paralelizm çözümün çok farklı komşulukları göz önüne alarak ve hazır çözüm üzerine birçok değişikliği bir defada uygulayarak kullanılmaktadır. Yöntem standart test problemleri kullanılarak denenmiş ve Solomon (1987)' un (21) bulduğu sonuçlara göre iyileşmeler olduğu görülmüştür (40). Rochat ve Taillard (1995) ZPARP için tabu arama algoritması geliştirmişler. Bu algoritmada arama işlemini yönlendiren ve "uyarlanabilir hafıza" adı verilen özelleştirilmiş farklılaştırma ve yoğunlaşma yöntemi kullanmıştır. Uyarlanabilir hafıza arama sırasında elde edilen en iyi çözümlerden oluşan bir rota havuzudur. Bu havuzun amacı yeni arama adımları için bir başlangıç çözümü sağlamaktır. Havuzdan rota seçimi rassal olarak yapılmakta ve iyileştirilen rotalar tekrar bu havuza kaydedilmektedir (41). Kontoravdis ve Bard (1995) ZPARP için iki aşamalı cimri rassallaştırılmış uyarlanabilir arama prosedürü (greedy randomized adaptive search procedure – GRASP) üzerine çalışmışlardır. Burada başlangıç prosedürü en fazla dağılmış veya en fazla zaman kısıtlı kaynak müşteriler ile başlamakta ve iyileştirme prosedüründe ise algoritma rotalanmamış müşteriler için en iyi eklenecek (insert) yerleri bulmakta ve Solomon (1987)' deki maliyet fonksiyonunu kullanarak bir ceza fonksiyonu hesaplamaktadır. Elde edilen bu rotalar daha sonra 2-opt kullanılarak iyileştirilmektedir (41). Potvin v.d. (1996a), ZPARP' ye tabu arama ile bir çözüm getirmeye çalışmışlar. Tabu arama algoritması her zaman sonucun uygunluğunu sağlayan özelleştirilmiş yerel arama metotlarına dayanmaktadır.

Yöntem Solomon (1987)' nin literatürdeki örneklerine uygulanmış, C tipi problemlerde optimal sonucu bulmuştur. Ayrıca literatürdeki diğer sezgisellerle de karşılaştırılmış ve bunlar ile rekabet edebilecek kapasitede olduğu söylenmiştir (42). Potvin ve Bengio (1996) ise, ZPARP'ne evrim paradigmasına dayanan bir çözüm getirmişler ve çalışmalarını Solomon (1987) örnek problemleri üzerinde denemişlerdir. Solomon (1987) örnek problemlerinden 5' i için (C tipi) optimumu bulabilmişler ancak R tipi problemler için çözümün geliştirilmesi gerektiğini vurgulamışlardır (43). Chiang ve Russell (1996) çalışmalarında benzetim tavlaması meta-sezgiseli geliştirmişlerdir. İki farklı komşuluk mekanizması, λ ve k-düğüm değişim süreçleri uygulanmıştır. Kısa-dönemli hafıza fonksiyonu kullanılmış. Literatür ve büyük kapsamlı gerçek hayat problemlerine uygulanan meta-sezgisel, daha önceki çalışmalardan daha olumlu sonuçlar vermiştir (44). Potvin v.d. (1996b), Sinir ağları ve genetik algoritma kullanarak, ZPARP için rota kurmayı iyileştiren hibrit bir algoritma geliştirmişlerdir. Algoritma, Potvin ve Rousseau (1993) de geliştirilen paralel rota kurma algoritmasının geliştirilmişidir (45). Taillard v.d. (1997), Yumuşak zaman pencereli araç rotalama problemi çözümü için tabu-arama sezgiseli geliştirmişler. Ayrıca komşuluk yapısı olarak CROSS isimli bir değişim yöntemi geliştirmişlerdir (46). Chiang ve Russell (1997) arama durumuna göre parametreleri dinamik olarak ayarlayan bir reaktif tabu arama algoritması geliştirmişlerdir. Burada tabu listesi büyüklüğü belirli bir çözüm sık sık bulunduğunda artmakta, uygun çözüm bulunamadığında ise azalmaktadır (41). Berger v.d. (1998) ZPARP çözümü için iki populasyonlu genetik algoritma kullanmıştır. Populasyonlardan biri toplam uzaklığı azaltmakta diğeri ise zaman penceresi ihlallerini azaltmaktadır. Solomon' un (1987) en yakın komşuluk algoritması kurucu algoritma olarak kullanılmıştır (45). Homberger ve Gehring (1998) ise çözümü için iki adet evrim strateji geliştirilmiştir. Evrim stratejileri 100 den 417 arasında değişen müşteriler ve 2 den 54 e kadar değişen araçların bulunduğu 58 probleme uygulanmıştır. Kendi çözümlerini literatürde bilinen en iyi çözümler ile karşılaştırmışlardır (47). Kilby v.d. (1999) ZPARP için yönlendirmeli yerel arama (guided local search – GLS) yöntemi geliştirmişlerdir. GLS yöntemi arama esnasında daha önceden bulunan çözüme yaklaşıldıkça maliyet fonksiyonunu bir ceza maliyeti ile artırarak bir önceki yerel minimumdan farklı alanları da araştırmaya zorluyor.

Burada yerel arama yöntemi olarak; 2-opt, relocate, exchange ve en iyi - kabul stratejili 2-opt* kullanılmaktadır (41). Liu ve Shen (1999), Yeni komşuluk yapısına dayanan iki aşamalı meta-sezgisel geliştirilmişler. Komşuluk inşaası rotalar ve düğümler arasındaki ilişkiye dayanmıştır. Paralel rota inşaası için kullanılan konvansiyonel metotların aksine, rotalar yüksek çözüm kalitesi için iç içe girmiş paralel yapıda inşaa edilmiştir. 60 test problemi üzerinde yapılan denemelerde yaklaşımın çok rekabetçi olduğu görülmüştür (48). Schulze ve Fahle (1999), çalışmalarında paralel tabu arama algoritması geliştirmişler. Komşuluk yapısı basit müşteri değişimlerine dayanmaktadır ve mümkün olmayan çözümlere de izin verilmektedir. Kesin algoritmalarda kullanılan sütun üretme metodu gibi, tabu arama tarafından türetilen rotalar bir havuzda toplanmaktadır. Tabu arama sezgiseline, başlangıç çözüm sağlamak için, hızlı bir küme kaplama sezgiseli havuzdaki rotalara uygulanmaktadır. Solomon (1987) (21) problemleri üzerine yapılan denemelerde yüksek kaliteli çözümler elde edilmiştir (49). Gambardella v.d. (1999) araç sayısı ve seyahat uzaklığı fonksiyonlarını en küçüklemeye çalışan çoklu karınca kolonisi sistemi geliştirmişlerdir. Burada birinci öncelikli olarak filo büyüklüğü, daha sonrada mesafe en küçüklenmeye çalışılır. Bu fonksiyonları yerine getirebilmek için iki farklı karınca kolonisi eş zamanlı çalışmaktadır (50). Caseau v.d. (1999)) birkaç tekniği hibritleştirmiştir. Bunlar sınırlı uyumsuzluk araması (limited discrepancy search – LDS), geniş komşuluk arama (large neighborhood search – LNS), zincir atma (ejection chains) ve ağaç atmadır (ejection trees). Başlangıç çözümü Caseau ve Laburthe (1999)' deki (35) yöntem ile kurulmaktadır. Daha sonra ağaç ve zincir atma yöntemleri, maliyetli müşterileri diğer rotalara eklemektedir. Ağaç ve zincir atma yöntemlerinin temel farkı bir müşteri eklenen rotadan, birden fazla sayıda müşteri çıkarılmasıdır. LNS prosedürü Shaw (1998)' deki (35) yöntem ile aynı şekilde çalışmaktadır (41). Tan v.d. (2001), çalışmalarında 3 adet sezgisel geliştirmişlerdir. Bunlar farklılaştırmalı yerel arama metodu, tavlama benzetimi-tabu arama hibrit algoritması ve hibrit genetik algoritmadır. Bu üç algoritma Solomon (1987) örnek problemleri ile denenmiş ve literatürdeki diğer sezgiseller ile karşılaştırılmıştır. Ek olarak problemlerin literatürde bilinen en iyi ve optimal sonuçlar ile de karşılaştırılmıştır. Örnek problem grupları ile yapılan denemeler sonucu önerilen algoritmalardan tavlama benzetimi ve tabu arama algoritmasının hibritlendiği çözüm

tekniğinin diğer önerdiklerine göre daha iyi sonuçlar verdiği görülmüştür. Ayrıca zaman olarak da farklılaştırılmış yerel arama algoritmasında sonra hibrit genetik algoritmasına göre çok daha kısa zamanda sonuca ulaşmaktadır (13). Braysy (2001a), ise ZPARP çözümü için saf genetik algoritma kullanmışlardır. Metodun performansı test problemleri ile denenmiş ancak en literatürdeki en iyi sonuçlardan daha iyi sonuç verememiştir. Ancak fark çok baskın değildir (51). Cordeau v.d. (2001), Periyodik ve çok-depolu zaman pencereli araç rotalama problemleri için bütünleştirilmiş bir tabu arama algoritması geliştirmişlerdir. Bu yöntemin başlıca yararları hızı, basitliği ve esnekliğidir. Birçok alternatif metodun aksine, bu metot çok basit bir taslağa dayanmaktadır ve sınırlandırılmış sayıda parametre ile çalışmaktadır (52). Ioannou v.d. (2001), çalışmalarında merkez fabrikadan coğrafi olarak dağılmış müşterilere mal dağıtımını planlamak istemektedirler. Mallar müşterilere en erken ve en geç zaman aralığında dağıtılmak zorundadır. ZPARP yapısına uyan bu probleme araştırmacılar sezgisel bir çözüm yöntemi geliştirmişlerdir. Atkinson' un greedy look-ahead sezgiseline dayanan çözüm metot geleneksel çözüm metotlarını geliştirmektedirler ve standart test problemlerinde çok iyi sonuçlar vermektedir (53). Czech ve Czarnas (2002), paralel benzetim tavlaması algoritmasını ZPARP' ye uygulamışlardır. Solomon (1987) örnek problemleriyle yapılan denemelerde bilinen çözümlere göre çok iyi sonuçlar alınmıştır (54). Gehring ve Homberger (2002), çalışmalarında iki aşamalı meta-sezgiselin paralelleştirilmesi anlatılmaktadır. Araç sayısının azaltılması ilk aşamada, seyahat mesafesinin azaltılması ise ikinci aşamada kullanılmaktadır. Metot 358 test problemi ile denenmiş, elde edilen sonuçlar paralelleştirme konseptinin doğruluğunu kanıtladığı öne sürülmüştür (55). Mester (2002), ZPARP için evrimsel stratejiler üzerinde çalışma yapmıştır. Burada ilk önce bütün müşteriler ayrı rotalarda servis edilmekte, daha sonra altı başlangıç çözümlü küme en-ucuz ekleme yöntemi ile üretilmektedir. Buradan elde edilen en-iyi çözüm genetik algoritmaya başlangıç çözümü olarak kullanılmaktadır. Burada atalar rassal olarak seçilmekte, arama ise, geleneksel arama algoritmalara (2-opt*, or-opt, ve λ-interchanges) dayanan mutasyonlar ile gerçekleşmektedir (41). Berger v.d. (2003), araştırmasında rassal üretilen başlangıç çözümü λ-değişim ve Liu ve Shen (1999) ekleme prosedürüne dayanan tekrar başlatma algoritmaları ile iyileştirilmektedir. Bu yaklaşımda da,

araştırmacılar Berger v.d. (1998)' de olduğu gibi iki adet populasyon kullanılmaktadır. Bu populasyonlardan biri toplam mesafeyi azaltmakta, diğeri ise zaman penceresi ihlallerini azaltmaktadır (41). Braysy (2001b, 2003b)' de ise Mladenovic ve Hansen (1997)' nın değişken komşuluk arama yönteminde değişikliğe dayanan yeni bir meta-sezgisel ortaya koyulmuştur. Bulunan sonuçlar, literatürde elde edilen diğer sonuçlardan daha iyidir. Amaçlanan sistem dört aşamalıdır. 1. aşamada ekleme tabanlı kurma sezgiseli ile rotalar oluşturulur, 2. aşamada araç sayılarını azaltmak üzere rota-eleme prosedürü çalıştırılır. 3. aşamada dört adet yeni arama prosedürü ile rota uzunlukları kısaltılır, son aşamada ise amaç fonksiyonu iyileştirilmesi yapılarak süreç tamamlanır (56-57). Chaovalitwongse v.d. (2003) çalışmasında, ZPARP' de filo büyüklüğü ve seyahat zamanını minimize etmek için GRASP yöntemi tartışılmıştır. GRASP yönteminde iki aşama vardır; inşaa ve yerel arama aşamaları. İnşaa aşamasında başlangıç çözümü uyarlanabilir rassallaştırılmış cimri fonksiyonun uyarlanması ile hesaplanmış. Yerel arama aşamasında ise birinci aşamada elde edilen sonuçların iyileştirilmesine çalışılmıştır. Bu çalışmada iyileştirilmiş çözüm, yeni yerel arama teknikleri ve alt sınır prosedürleri ile bulunmaya çalışılmıştır (58). Li ve Lim (2003), tavlamaya benzetimine dayanan ve yerel aramaları farklılaştıran ve yoğunlaştıran bir meta-sezgisel geliştirilmiş, Solomon (1987) (21) örnek problemleriyle yapılan denemelerde o zamana kadar bilinen 7 sonuçtan daha iyi, 19 sonuca da eşit sonuçlar elde edilmiştir (59). Lee v.d. (2003), Singapur' da faaliyet gösteren bir lojistik firmasının planlamasını yapan, araç planlama sistemini anlatmaktadır. Problem ZPARP modeli olarak modellenmiş ve tabu arama metodu ile çözüme ulaştırılmıştır. Kullanılan yöntem ile eski ölçütlere göre %8 iyileştirme sağlanmıştır (60). Lau v.d. (2003) çalışmasında ZPARP'nin kısıtlı araç sayısı altındaki türevi anlatılmış. Bu senaryo altında çözüm servis yapılmamış müşterileri ve/veya gevşetilmiş zaman penceresini içerebilir. Bu problemi çözmek için tabu arama algoritması geliştirilmiş. Ayrıca zaman pencerelerinin, gecikme için ceza notasyonu ile gevşetilmesine izin verilmiştir. Bu yaklaşımda müşteri işleri bütün çoklu amaçları örten bir hiyerarşik amaç fonksiyonuna dayalı eklenmiştir. Örnek problem çözümleri, bu yaklaşımın rekabetçi olduğunu göstermiştir (61). Sa'adah v.d. (2004), çalışmasında çoklu karınca koloni sistemleri ile yapılan deneyleri anlatmaktadır. Bu üçlü karınca kolonisi sisteminde;

bir koloni araç sayılarının minimize edilmesine, bir diğeri toplam uzaklığın minimize edilmesine üçüncüsü ise müşteri bekleme zamanının maksimize edilmesine çalışmaktadır. Bu üç koloni sisteminin eklenmesi çözüm sonuçlarını fark edilir şekilde değiştirmiştir. Solomon (1987)'nin 56 örnek problemine uygulanmış ve diğer yöntemlerle karşılaştırılabilir olduğu görülmüştür (62). Cordeau v.d. (2004), Yazarlar daha önce (2001) geliştirdiği tabu arama algoritmasının özelleştirmiş tipini sunmuşlardır (63). Ioannou ve Kritikos (2004) ise çalışmalarında, üç aşamalı metot geliştirmişlerdir. Çözüm metodu Macar metodunu kullanarak probleme başlar, atamalar basit bir ayırma sezgiseli ile ZPARP için uygun rotalara çevrilmekte, seyahat ve araç bekleme zamanları göz önüne alınarak seçilen en iyi rotalar sonucun bir parçasını oluşturmaktadır. Buradan sonra kalan müşterilere sezgisel bir yaklaşım uygulanarak çözüme ulaşılmaktadır. Metot literatür problemlerine uygulanmış ve toplam seyahat zamanları göze alındığında diğer sezgisellere göre iyileşmeler olmuştur (12). Bent ve Van Hentenryck (2004), müşteri sayısının stokastik değiştiği ZPARP için servis verilen müşteri sayısını maksimize etmeye yarayan bir metot geliştirmişlerdir. Ayrıca sürekli rotalama planları üreten bir çoklu-senaryo yaklaşımı da sunulmuştur. Yaklaşım Solomon (1987)' un (21) örnek problemlerinde dinamik değişiklikler yapılarak denenmiştir (64). Braysy v.d. (2004) çalışmaları, çok-başlangıçlı yerel arama sezgiseline ve yeni iyileştirme algoritmalarına dayanmaktadır. İnşaa sezgiseli için, hızlandırılmış bir teknik tanıtılmış ve sonuçlar eşik kabul işlemcisi ile optimize edilmiştir. Solomon (1987) örnek problemlerine uygulanan yöntemin verimli ve rekabetçi olduğu görülmüştür (65). Homberger ve Gehring (2005) çalışmalarında ZPARP birincil amaç kullanılan araç sayısının minimizasyonu, ikincil amaç da toplam seyahat uzunluğunun minimizasyonudur. Burada birincil amacı evrim strateji ile ikinci amacıda tabu arama algoritmasıyla çözen bir yaklaşım geliştirilmiştir. Yaklaşım literatürdeki problemlerle denenmiş ve sonuçların diğer yöntemler ile rekabet edeceği görülmüştür (66). Le Bouthiller ve Cranic (2005), içinde uygun çözümler havuzu yardımıyla birbirleri ile iletişim kuran temsilcilerin bulunduğu, paralel bir metodoloji sunmuşlardır. Bu temsilciler kümesi, basit kurma algoritmaları, yerel arama algoritmaları, genetik algoritmalar ve Gendreau v.d. (1994)' nun taburoute algoritmasından oluşmaktadır. Genetik algoritmada, bir kromozomdaki genler arasında öncelik ilişkisine dayanan geleneksel

sıra-tabanlı operatörler kullanılmaktadır (41). Ibaraki v.d. (2005) Yumuşak zaman pencereli araç rotalama problemi için yerel arama algoritmaları üzerinde durmuşlardır. Zaman penceresi kısıtı ceza fonksiyonu olarak ele alınmıştır. Geliştirilen çözüm yönteminde, müşterileri araçlara atamak ve araçlar tarafından ziyaret edilecek müşteri sıralarını bulmak için yerel arama yöntemleri kullanılmaktadır. Komşuluk algoritması olarak gelişmiş bir yöntem olan çevrimsel-değişim (cyclic-exchange neighborhood) kullanılmaktadır. Algoritmaya, toplam cezayı minimize edecek, servise başlama zamanlarını optimal bulmak için dinamik programlama (DP) yaklaşımı eklenmiştir (67).

Yukarıda anlatılan çalışmaların kurma ve iyileştirme algoritmaları ve Solomon (1987) (21) örnek problemlerindeki çözüm performansları Çizelge 4.1' de gösterilmektedir.

Çizelge 4.1. Literatürde meta-sezgisellerle çözülmüş zaman pencereli araç rotalama problemleri (41)

Yazar	Yıl	Yöntem	Kurma Algoritmaları	Geliştirme Algoritmaları	CNV	CTD	Notlar
Thangiah v.d.	1991	GENETİK	cheapest insertion / GIDEON	lambda-interchange	418	58.905	cheapest insertion: en ucuz ekleme sezgiseli / ilk-eniyi veya küresel-en iyiyi kabul stratejisi
Garcia v.d.	1994	TABU	insertion-1	2-opt / Or-opt	436	65.977	insertion-1: Solomon (1987)' nin basit ekleme sezgiseli / komşulukta uzaklık olarak yakın müşteriler bulunuyor (21)
Rochat ve Taillard	1995	TABU	düzeltilmiş insertion-1	2-opt / relocate	415	57.231	uyarlanabilir hafıza
Kontoravdis ve Bard	1995	META-HEU		2-opt	427	64.196	
Potvin v.d.	1996	TABU	insertion-1	2-opt* / Op-opt	426	63.530	komşulukta uzaklık olarak yakın müşteriler bulunuyor
Potvin ve Bengio	1996	GENETİK	insertion-1	relocate / Or-opt	422	62.634	mutasyon ve crossover operatörleri iyileştirme algoritması olarak ele alınmıştır.
Chiang ve Russell	1996	SA	Russell (1995)' nin paralel versiyonu (32)	lambda / k-düğüm interchange	422	64.996	ilk kabul stratejisi
Taillard v.d.	1997	TABU	insertion-1	CROSS	410	57.523	yumuşak zaman penceresi / uyarlanabilir hafıza
Chiang ve Russell	1997	TABU	düzeltilmiş Russell (1995) (32)	lambda-interchange	411	58.502	reaktif tabu arama
Berger v.d.	1998	GENETİK	en yakın komşuluk	relocate / LNS (35)	424	60.539	LNS: large neighborhood search (Shaw 1998) / mutasyon ve crossover operatörleri iyileştirme algoritması olarak ele alınmıştır.
Schulze ve Fahle	1999	TABU	paralel insertion-1 / savings sezgiseli	ejection chains / Or-opt	414	60.346	savings heuristics: tasarruf sezgiseli / üretilen rotalar bir havuzda depolanmakta
Homberger ve Gehring	1999	GENETİK	olasılıklı savings sezgiseli	Or-opt / 2-opt / lambda-interchange	406	57.876	mutasyon ve crossover operatörleri iyileştirme algoritması olarak ele alınmıştır.
Kilby v.d.	1999	GLS		2-opt / relocate / exchange / 2-opt*	423	57.423	GLS: guided local search / en iyi kabul stratejisi
Liu ve Shen	1999	META-HEU	-	lambda-interchange / basit reinsertion	412	59.318	

Çizelge 4.1. Literatürde meta-sezgisellerle çözülmüş zaman pencereli araç rotalama problemleri (41) (devamı)

Yazar	Yıl	Yöntem	Kurma Algoritmaları	Geliştirme Algoritmaları	CNV	CTD	Notlar
Gambardella v.d.	1999	ACO	karınca kolonisi	CROSS	407	57 525	CROSS: Taillard v.d. (1997) çalışması
Caseau v.d.	1999	LDS	Caseau ve Laburthe (1999a) sezgiseli	ejection chains / ejection trees / LNS	-	-	
Cordeau v.d.	2001	TABU	düzeltilmiş sweep sezgiseli	relocate / GENI	407	57 556	GENI: relocate operatörü devamı / sweep Heuristics: tarama sezgiseli
Tan v.d.	2001	GENETİK +SA +TABU	insertion-1	lambda-interchange	470	57 799	ilk-kabul stratejisi
Gehring ve Homberger	2002	GENETİK	olasılıklı savings sezgiseli	Or-opt / 2-opt / lambda-interchange	406	57 641	crossover operatörü yok
Mester	2002	GENETİK	cheapest insertion	Or-opt / 2-opt / GENIUS / LNS / lambda-interchange	406	57 219	crossover operatörü yok
Berger v.d.	2003	GENETİK	rassal insertion sezg.	düzeltilmiş LNS / lamda-interchange / relocate	405	57 952	
Li ve Lim	2003	SA ve TABU	insertion-1 / sweep	rassal shifts / rassal exchange	411	57 467	
Braysy	2003b	VND (VNS)	cheapest insertion	ICROSS / IOPT / IRP / O-opt / ejection chains	405	57 710	
Bent ve Van Hentenryck	2004	SA-LNS		Or-opt / 2-opt / 2-opt* / exchange, relocate	405	57 273	başlangıç çözümü belirtilmemiş
Braysy v.d.	2004	VND (VNS)	cheapest insertion	ICROSS / ejection chains	406	57 422	Braysy (2003b) devamı
Le Bouthillier ve Cranic	2005	GENETİK	2-opt, 3-opt - Or-opt ile birleştirilmiş algoritma	Or-opt / 2-opt / 3-opt / taburoute	409	57 574	
Homberger ve Gehring	2005	GENETİK	olasılıklı savings sezgiseli	Or-opt / 2-opt / lambda-interchange	405	57 309	crossover operatörü yok
Ibaraki v.d.	2005	MLS-ILS-AMLS	MLS	CROSS / 2-opt* / cyclic exchange /Or-opt	405	57 444	MLS: multistart local search / ILS: iterated local search / AMLS: adaptive multistart local search

5. KARINCA KOLONİSİ OPTİMİZASYON ALGORİTMASI

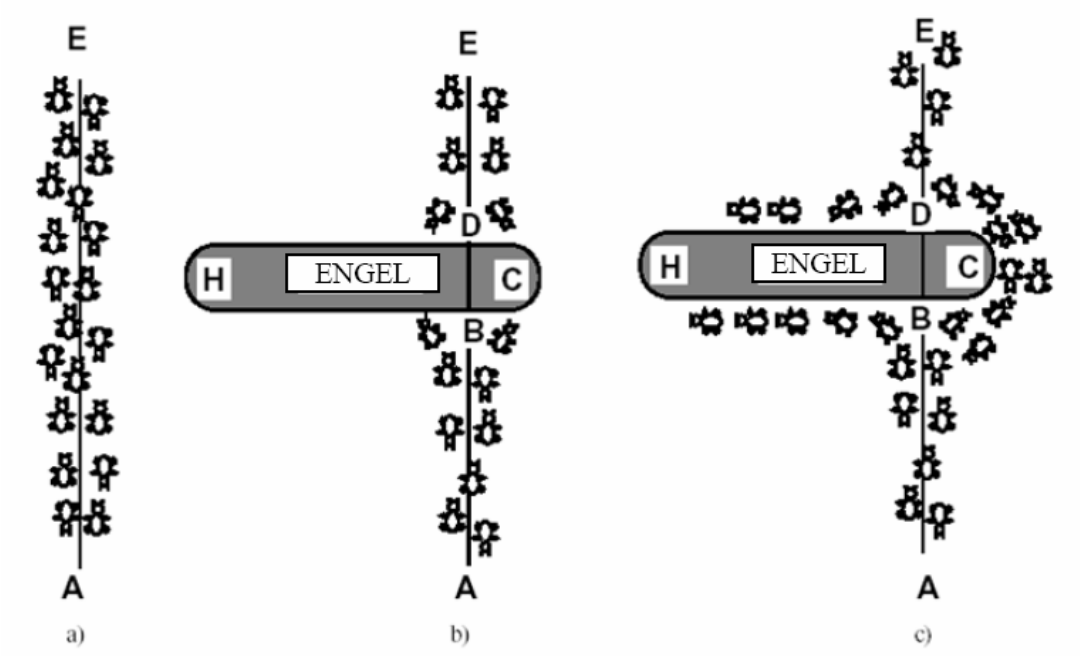
Bu tez çalışmasında geliştirilen zaman pencereli araç rotalama problemi karar destek sisteminin çözüm modülünde faydalanılması nedeniyle bu bölümde karınca kolonisi optimizasyon (KKO) algoritması tanıtılacaktır. KKO algoritması son yıllarda bir çok NP-zor problemin çözümüne başarıyla uyarlanmış çok popüler bir meta-sezgiseldir. Diğer meta-sezgisellerle karşılaştırıldığında, KKO algoritması, araç rotalama problemlerinde daha az sayıda araştırmada kullanılmıştır.

5.1. Gerçek Karınca Kolonisinin Davranışı

Karıncalar, koloni halinde yaşayan ve problem çözmede örnek alınabilecek davranış özellikleri olan hayvanlardır. Karıncalar tam bir görme yeteneğine sahip olmamasına rağmen yuvalarından yiyecek kaynağına ve yiyecek kaynağından yuvaya uzanan en kısa yolu bulabilmektedirler. Karıncalar, yuva-yiyecek arasındaki yolun bir şekilde bozulmasından dolayı ortaya çıkacak şartlara uyum sağlayarak, yeni bir en kısa yol kurma doğal yeteneğine sahiptir.

Yuva ile yiyecek arasındaki bu en kısa yolu bulmak için karıncalar tarafından kullanılan ve karıncalar arasında haberleşmeyi sağlayan maddeye feremon denilmektedir. Karıncalar, hareket halinde bulundukları yola belirli miktarda feremon maddesi bırakırlar. Gitmek için seçecekleri yönün belirlenmesinde feremon miktarı önem arz etmektedir. Karıncaların feremon miktarı fazla olan yönleri tercih etme olasılığı, feremon miktarı az olan yönleri tercih etme olasılığında daha fazladır. Ancak feremon maddesi az olan yönlerin tercih edilme olasılığı az da olsa vardır. Bundan dolayı hareket halindeki karıncalar en uygun yolu kısa süre içerisinde bulabilmektedirler. Şekil 5.1.a ’ daki yuva-yiyecek (A-E) arasını birbirine bağlayan en kısa hat üzerinde hareket eden karıncaların durumun incelendiğinde, Şekil 5.1.b’ deki gibi yol üzerine bir engel konularak, mevcut en kısa yol bozulursa, engelin hemen önündeki ve arkasındaki karıncalar, tercih edilmesi gereken feremonsuz yönler ile karşı karşıya gelirler. Bu durumda Şekil 5.1.c’ deki gibi karıncalardan yaklaşık yarıya yakını H yönünü kalan yarısının ise, C yönünü tercih etme durumu

söz konusudur. Engelin kısa C tarafında kalan karıncalar, uzun H tarafını tercih eden karıncalarla kıyaslandığında, kesilmiş olan feremon maddesi bağlantı yolunu çok daha hızla oluştururlar. Bunun nedeni karınca hızlarının aynı olduğu varsayımı altında, birim zamanda kısa C yolundan geçen karınca sayısının, uzun H yolundan geçen karınca sayısından fazla olmasıdır. Kolonideki karıncaların yaklaşık olarak geçtikleri yola eşit miktarda feremon maddesi bıraktıkları kabul edilirse, kısa C yoluna bırakılan feremon miktarı daha fazla olacaktır (68).



Şekil 5.1. Karınca davranışları
a) Karıncalar A-E arasındaki yolu izlemektedirler b) Yolun bir yerine bir engel konulmuştur ve karıncalar hangi yönü seçeceklerine rasgele karar verirler c) Kısa olan yolda daha fazla feremon birikir (69)

Yön tercihinin feremon miktarına bağlı olması nedeniyle, arkadan gelen karıncalarda yüksek ihtimalle kısa C yolunu tercih edeceklerdir ve bu yolda daha fazla feremon kokusu birikecektir. Böylece, kolonideki çoğu karınca yeni kısa yolu seçerek değişmiş olan çevreye ayak uyduracaklardır. Bu olayda bir pozitif geri besleme vardır. Bir yoldan geçen karınca sayısı arttıkça yola yapıştırılan feremon miktarı artmakta ve feremon miktarı arttıkça da yolu tercih eden karınca sayısı artmaktadır.

Bu olay oto-katalitik işlem olarak adlandırılır. Pozitif geri besleme nedeniyle, karıncaların çoğu kısa süre içerisinde daha kısa yolu seçmesi geçekleşmektedir.

5.2. Karınca Kolonisi Optimizasyon Algoritması (Ant Colony Algorithm-ACA)

KKO algoritması, gerçek karınca kolonisi davranışlarının matematiksel modeller ile anlatılmasıdır. İlk çalışma Dorigo v.d. (1991) tarafından yapılmış, oluşturdukları sistemi "karınca sistemi" ve ortaya çıkan algoritmayı ise "karınca algoritması" olarak tanımlamışlardır. Araştırmacıların geliştirdikleri yapay karınca kolonileri, bir optimizasyon aracı olarak kullanıldığından, gerçek karınca davranışlarından farklı yapıdadır. Örneğin, yapay karıncalar belirli bir hafızaya sahiptirler ve gerçek karıncalar gibi tamamen kör değildirler. Dorigo v.d. (1991) yaptıkları ilk çalışmada üç karınca algoritması tanımlamışlar ve bu algoritmaları GSP probleminin çözümünde kullanmışlardır (68).

Daha önceki bölümlerde de değinildiği gibi GSP, n tane şehir verildiğinde, gezgin bir satıcının her bir şehri bir kez ziyaret etmek şartıyla minimum uzunluklu kapalı turu oluşturma problemidir. $i.$ ve $j.$ şehirler arasındaki yolun öklid uzunluğu $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ dir. GSP' nin herhangi bir durumu ağırlıklandırılmış $G(N, A)$ grafıdır. m, kolonideki toplam karınca sayısını göstermek üzere ve t anındaki $i.$ şehirde bulunan karıncaların sayısı $b_i(t)(i = 1,2,...,n)$ ile verilirse, iki büyüklük arasındaki ilişki $m = \sum_{i=1}^{n} b_i(t)$' dir.

Ayrıca her karıncanın aşağıdaki özelliklere sahip bir bireyi temsil ettiği kabul edilirse karınca;

- i şehrinden j şehrine giderken (i, j) hattına bir miktar feremon maddesi bıraksın,

- Gideceği şehri, o hatta mevcut olan feremon maddesi miktarı ile iki şehir arasındaki mesafenin fonksiyonu olan bir olasılıkla seçsin,

- Karıncalar, şartları sağlayan turu gerçekleştirebilmeleri için zorlansın. Ziyaret edilmiş şehirlerin tekrar ziyaret edilmesi yasaklanarak tur tamamlansın.

$\tau_{ij}(t)$, t anında (i,j) hattına bırakılan feremon miktarı, ρ, buharlaşma katsayısı (feremon miktarının sınırsız büyümesini engelleyen 0-1 arası bir sayı) olmak üzere $t+1$ anında (i,j) hattındaki toplam feremon miktarı Eş. 5.1' ile hesaplanır.

$$\tau_{ij}(t+1) = \rho.\tau_{ij}(t) + \Delta\tau_{ij}(t,t+1) \qquad [5.1]$$

Eş.5.2' de ise birim zamanda, (i,j) hattına tüm karıncalarca bırakılan feremon maddesi miktarı gösterilmektedir.

$$\Delta\tau_{ij}(t,t+1) = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}(t,t+1) \qquad [5.2]$$

Burada $\Delta\tau_{ij}^{k}(t,t+1)$, t ve $t+1$ zaman aralığında k. karınca tarafından (i,j) hattına bırakılan feremon maddesinin birim uzunluk başına miktarıdır.

Bir karıncanın n faklı şehri ziyaret etmesini sağlamak için her karıncaya bir tabu listesi atanır. Bu tabu listesi yardımıyla karıncanın t anına kadar ziyaret etmiş olduğu şehirler hafızaya alınır ve tur tamamlanıncaya kadar bu şehirlerin tekrar ziyaret edilmesi önlenir. Tur tamamlandığında tabu listesi boşaltılır.

$\eta_{ij} = 1/d_{ij}$ olarak tanımlanan seçilebilirlik parametresi kullanılarak k. karıncanın i. şehirden j. şehre geçebilme olasılığı Eş. 5.3 ile tanımlanır.

$$p_{ij}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{j\in Ni}[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}, j\in N_i \\ 0 \qquad\qquad\qquad j\notin N_i\end{cases} \qquad [5.3]$$

Bu formüldeki N_i, k. karıncanın tabu listesinde olmayan şehirleri ifade eder. α ve β parametreleri $p_{ij}(t)$ olasılık değerleri hesaplanırken feremon maddesi ve seçilebilirlik kriteri arasında nispi önemi belirleme imkanı vermektedir. Burada seçilebilirlik, yakın şehirlerin daha yüksek ihtimalle seçilmesini teşvik eden aç gözlü (greedy) sezgisel kuralı tanımlar, feremon maddesi ise trafiği daha fazla olan hattın seçilmesine katkıda bulunarak oto-katalitik bir işleme sebep olur.

Bu açıklamalara göre karınca kolonisi algoritmasının GSP için kullanım adımları aşağıdaki gibidir.

Adım 1: Tüm şehirlere belirli miktarlarda $b_i(t)$ karınca yerleştir. Buna göre karıncaların tabu listesini yenile. Her hattın koku miktarını sıfırla. Sayacı sıfırlar.

Adım 2: Tabu listeleri dolana kadar bu adımdaki işlemleri tekrarla. Her şehirdeki tüm karıncalar için Eş.5.3 yardımıyla hesaplanan $p_{ij}(t)$ değerine bağlı olarak hareket etmek amacıyla j. şehri seç. Karınca k'yı j. şehre hareket ettir ve j. şehri k. karıncanın tabu listesine dahil et. $\Delta\tau_{ij}(t,t+1)=\Delta\tau_{ij}(t,t+1)+Q/d_{ij}$ bağıntısı aracılıyla feremon miktarını yinele. Her kenar (i,j) için $\tau_{ij}(t+1)$' yi Eş.5.1 aracılığıyla hesapla

Adım 3: Şu ana kadar bulunan en kısa turu hafızaya al. Durdurma kriteri sağlanıyorsa Adım 4' e git. Yoksa, tüm tabu listesini boşalt. Tüm şehirlere belirli miktarda karınca yerleştir ve Adım 2' ye git.

Adım 4: En kısa turu yaz ve dur.

Bu algoritmada başlangıçta karıncalar farklı şehirlerde konumlanırlar ve hatlara feremon miktarlarının başlangıç değerleri atanır. Her karıncanın tabu listesinin ilk elemanı başlangıç şehrine set edilir. Sonra her karınca *i*. şehirden *j*. şehre α ve β'nın fonksiyonu olarak hesaplanan olasılık değerine göre hareket eder. İlk parametre olan feremon maddesi τ_{ij}, aynı hattı (i, j) geçmişte ne kadar çok karıncanın seçtiği hakkında bilgi verirken, ikinci parametre seçilebilirlik η_{ij}, daha yakın şehrin daha çok tercih edilmesi gerektiğini vurgulamaktadır. $\alpha = 0$ yapılarak çoklu başlama noktasına sahip stokastik bir aç gözlü algoritma üretir. Bir karınca, hareketi gerçekleştirdiğinde (i, j) hattına bırakılmış olduğu feremon miktarı, aynı hatta daha önceden var olan koku miktarı ile toplanır. Her karınca hareketinden sonra geçiş olasılıkları yeni feremon miktarları kullanılarak hesaplanır.

Karıncaların tabu listeleri dolunca, kolonideki m karınca tarafından bulunan en kısa tur hesaplanır ve hafızaya alınır, tabu listesi boşaltılır. Buraya kadar olan işlemler ardışık olarak tur sayıcısı maksimum istenilen tur sayısına eşit olana kadar tekrarlanır veya tüm karıncalar aynı turu oluşturana kadar işlem devam eder. Bu algoritmada durdurma kriteri olarak önceden belirlenen bir maksimum iterasyon sayısı veya tüm karıncaların aynı turu oluşturması şartı kullanılabilir (68).

5.3. Karınca Kolonisi Optimizasyon Algoritmasının Rotalama Problemlerindeki Uygulamaları

KKO algoritması, birçok NP türü problemin çözümünde kullanılmıştır. Bu problemler rotalama, atama, çizelgeleme, küme kaplama, sırt çantası, şebeke rotalama v.b. olarak sayılabilir. Bu çalışmada yalnız rotalama problemleri ile ilgili referanslar Çizelge 5.1' de verilmektedir (70).

Çizelge 5.1. Karınca kolonisi optimizasyonunun bazı uygulamaları (70)

Problem Tipi	**Problem Adı**	**Referans**
Rotalama	Gezgin Satıcı	Dorigo, Maniezzo ve Colorni (1991a, b, 1996) Dorigo (1992) Gambardella ve Dorigo (1995) Dorigo ve Gambardella (1997a,b) Stützle ve Hoos (1997, 2000) Bullnheimer, Hartl ve Strauss (1999c) Cordón, de Viana, Herrera ve Morena (2000)
	Araç Rotalama	Bullnheimer, Hartl ve Strauss (1999a, b) Gambardella, Taillard ve Agazzi (1999) Reimann, Stummer ve Doerner (2002) Sa'adah Ross ve Paechter (2004) (62)

6. ZAMAN PENCERELİ ARAÇ ROTALAMA PROBLEMİ KARAR DESTEK SİSTEMİ (ZPARP-KDS) UYGULAMASI

6.1. Amaç

Zaman pencereli araç rotalama probleminin gerçek hayat problemlerine (okul otobüsü rotalama, akaryakıt dağıtımı, gazete dağıtımı, posta dağıtımı, tam zamanlı üretim için satıcı dağıtımı, güvenlik devriyesi kontrolleri, kentsel atık toplama ve zincir mağaza dağıtım lojistiği gibi...) daha uygun olduğunu ve hakkında bir çok araştırma yapıldığı önceki bölümlerde belirtilmiştir. Ancak bu çalışmaların çoğu, az ve orta seviyede yöneylem araştırması bilgisi olan veya hiç olmayan lojistik ve dağıtım planlamacıları için kullanıcı-dostu bir ortam sağlayamazlar. Çünkü bu çalışmalar daha çok teorik ağırlıklıdır. Bunları, gerçek problemlerin çözümünde kullanma durumu, bazı uyarlamalar (problem verileri girilmesi, bazı varsayımların eklenmesi gibi...) gerektirir. Bu çözüm algoritmalarının kullanabilirliğini artırma, her türlü ortam için çözüm geliştirebilecek bir şekle dönüştürme gereği ZPARP için bir KDS tasarlanmasını zorunlu kılmıştır. Aşağıda özetlenen çalışmalar klasik ARP için tasarlanmış KDS' lerin varlığını göstermekle beraber, bu çalışmalar içinde ZPARP' leri modelleyen bir KDS tasarımı bulunmamaktadır.

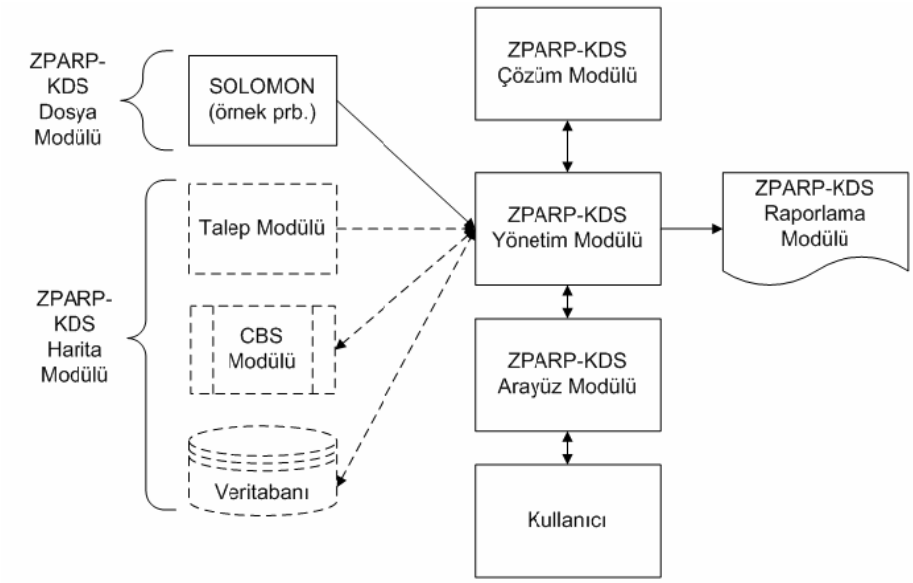
ARP için KDS mantığını anlamak açısından literatürdeki çalışmalar özetlenmek gerekirse; Keenan (1998) coğrafi bilgi sistemi (CBS) tekniklerinin bir çok araç rotalama probleminin çözümüne katkıda bulunabileceğini ileri sürmüş ancak genel amaçlı CBS' nin, ARP çözümü için ihtiyaç duyulan algoritmalar ile kullanıcıları etkileştirmediğini belirtmiştir. Bu nedenle tasarlanacak bir KDS' de, rota algoritmaları ile CBS' nin birleştirilmesini önermişler ve bu konuda araştırılması gereken bir çok yönün varlığını ortaya koymuşlardır (71). Bu öngörü ile tasarlanmış bir KDS Tarantilis v.d. (2004)' indir. Tarantilis v.d. (2004) araçların depoya dönmediği açık araç rotalama problemi için BoneRoute adlı bir metasezgisel kullanan bir KDS tasarlamışlardır. Tasarlanan KDS bir coğrafi bilgi sistemi ile birlikte çalışmaktadır. Model olarak kullanılan BoneRoute ise genetik çözüm prosedürlü bir algoritmadır ve uyarlanabilir hafıza konseptine dayanmaktadır (72).

Diğer bir çalışmada ise Ruiz v.d. (2004) gerçek hayat araç rotalama problemi için tam sayılı programlama yöntemi kullanan bir KDS geliştirmişlerdir. Bu KDS' de bir CBS ile entegre çalışmaktadır. Problem çözümünün ilk aşamasında tüm rotalar birerleme yöntemi ile üretilmekte, ikinci aşamada ise tam sayılı programlama algoritması ile optimum rotalar seçilmektedir (8). Özetle bu araştırmalarda, klasik ARP için tasarlanmış KDS' ler sunulsa da, bu araştırmalarda diğer tür ARP' ler için de KDS' leri tasarlanmasının gereği vurgulanmaktadırlar. Bu gereklilik, rotalama algoritmalarının kullanıcı ve CBS' ler ile uyumlu çalışmaması nedeniyle aralarında bazı uyarlamalar yapılması ihtiyacından kaynaklanmaktadır. Bu uyarlamalar ise KDS mantığı ile gerçekleşebilmektedir.

Tüm bu sebeplerden ötürü, bu yüksek lisans tez çalışmasında, gerçek hayat problemlerinde geniş uygulama alanları bulunan ZPARP' nin çözüm algoritmalarını kullanıcı ve CBS ile uyumlu bir hale getiren bir KDS (ZPARP-KDS) tasarlanması gerekli görülmüştür. Bu tasarımla, birçok lojistik ve dağıtım şirketinin rota planları yaparken harcadığı zamanı azaltmak, bu planlar uygulanırken gereken araç yatırım ve işletme maliyetlerini en aza düşürmek amacıyla verecek kararlar için kullanıcıya bazı işaret ve öngörüler sunulması amaçlanmıştır. Tasarlanan ZPARP-KDS' nin sistem mimarisi ve çalışma mantığı ilerleyen bölümlerde ayrıntılı olarak anlatılmaktadır.

6.2. ZPARP-KDS Mimarisi

Bu bölümde tasarlanan ZPARP-KDS' nin genel işleyişi açıklanacaktır. Sistemde bulunan bileşenler; dosya modülü, harita modülü, çözüm modülü, yönetim modülü, arayüz modülü, raporlama modülü ve kullanıcıdır. Bu bileşenlerin birbirleri ile etkileşimi Şekil 6.1' de gösterildiği gibidir.



Şekil 6.1. ZPARP-KDS sistemi yapısı

6.2.1. ZPARP-KDS dosya modülü

Bu modül, hali hazırda sistemde mevcut çözüm algoritmalarını veya daha sonradan ZPARP-KDS Çözüm Modülüne eklenebilecek çözüm algoritmalarının çalışabilirliğini denetlemek amacıyla tasarlanmıştır. Bu modül ile Solomon (1987)' un (21) geliştirdiği C, R ve RC tipindeki problem setleri sisteme yüklenir ve çözüm algoritmalarının sonuçları, kalitesi ve davranışları test edilebilir. Ayrıca farklı problem verileri Solomon (1987) formatına uygun düzenlenip sistemden sonuçlar elde edilebilir. Bu modülün çalıştırılmış hali, sistem sonuçlarını ve ortaya çıkan raporları tanıtmak amacıyla ZPARP-KDS Arayüz Modülünde gösterilmektedir.

6.2.2. ZPARP-KDS harita modülü

ZPARP-KDS' yi gerçek problem verileri ile beslemek bu modülün görevidir. Harita Modülü üç alt modülden oluşur. Bu alt modüler; Talep Modülü, CBS Modülü ve veritabanı modülüdür. Bu üç alt modülün kendi aralarında iletişimi yoktur. İletişimin ZPARP-KDS Yönetim Modülü üzerinden kontrollü gerçekleştirilmesi amaçlanmıştır. Buna göre alt modüllerin çalışma prensipleri aşağıda anlatılmaktadır;

Talep Modülü:

Bu modülün görevi müşteri taleplerini online veya manuel yöntemler ile toplayarak, değerlendirmektir. Değerlendirme sonucu bu veriler veritabanına kaydedilip çözüm modelinin çalıştırılmaya ihtiyaç duyulacağı anlarda tekrar sisteme geri yüklenecektir. Talep modülünün bilgi elde etme yolları online yöntemde web üzerinden müşterilerin taleplerini girmesi ile, manuel yöntemde ise ZPARP-KDS kullanıcısının telefon, faks, e-mail gibi araçlarla gelen talepleri arayüz aracılığıyla sisteme girmesi ile olacaktır.

CBS Modülü:

CBS modülü, aşağıda anlatılan rotalama işlemlerini harita yardımıyla yapılabilmek için ZPARP-KDS tasarımına eklenmiştir. Bu işlemlerin bazıları şunlardır;

- Daha önceden kayıtlı bulunan müşteri verileri ile talep modülü tarafından elde edilip veritabanına saklanan verilerin veritabanından geri çekilerek sayısal haritalar üzerinde gösterilmesi,

- Haritaya aktarılan bu veriler üzerinde düzeltme yapılabilmesi,

- Yollar, araçlar ve diğer veriler için kısıt atama işlemlerinin yapılabilmesi,

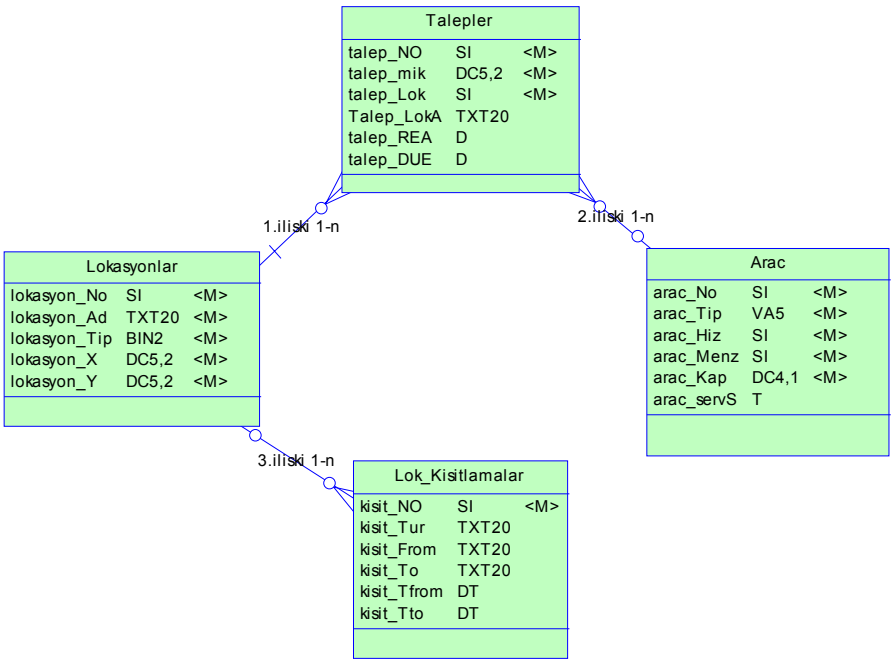
- Üzerinde değişiklik yapılmış-yapılmamış ve/veya kısıtlanmış-kısıtlanmamış tüm problem verilerinin son hali ile çözüm modeline aktarılabilmesi,

- Algoritma çözüm rotalarının harita üzerinde bağlantılar ile gösterilebilmesi,

- Son aşamada, algoritma ile elde edilmiş çözümde manuel olarak değişiklik yapılabilmesi gibidir.

Veritabanı Modülü:

Veritabanı modülü, talepler modülünden gelen müşteri talebi ile ilgili bilgilerin (talep miktarı, talep eden müşteri lokasyonu, talebin istendiği en erken zaman, talebin istendiği en geç zaman gibi...), sisteme daha önceden girilen müşteri lokasyon bilgilerinin (lokasyon koordinatları ve tipi gibi...), araç bilgilerinin (kapasite, hız, menzil, servis süresi gibi...) ve kısıtlama bilgilerinin (kısıtın nereden nereye, hangi zaman aralığında olduğu gibi...) tutulduğu tablolardan oluşmaktadır. Veri tabanının kavramsal veri yapısı Şekil 6.2' deki gibidir.



Şekil 6.2. ZPARP-KDS kavramsal veri modeli

Bu modülün tabloları SQL Server 2000 üzerinde oluşturulmuş olup, veritabanına yazma silme benzeri işlemlerin ZPARP-KDS yönetim modülü üzerinden yapılması öngörülmüştür. Üretilen tablolar Ek-1, Ek-2, Ek-3 ve Ek-4' de gösterilmektedir.

ZPARP-KDS tasarımındaki harita modülü için CBS araçları gerektiğinden burada sunduğumuz çalışmayı sadece dosya modülü ile örneklendiriyoruz. Ancak taleplerin alınıp sayısal haritalar üzerinde gösterilmesi, talep verilerinin harita üzerinde

değiştirilmesi, çözüm rotaların haritalar üzerinde gösterilmesi gibi harita modülü ve CBS alt modülünden istenen özellikler gelecek çalışmalara tasarım fikri vermesi açısından önemlidir.

6.2.3. ZPARP-KDS çözüm modülü

Bu modül, ZPARP' leri çözebilecek algoritmaların içinde bulunduğu ve gerektiğinde başkalarının da eklenebileceği modüldür. Bu modülde algoritmalar, istenilen çözüm kalitesine göre, deterministik–tam çözüm algoritmalar veya yerel, meta-sezgisel arama algoritmaları arasından seçilebilir. Ayrıca ileride sisteme eklenebilecek diğer ARP çözüm algoritmalarının bulunacağı modüldür.

Bu modülde literatürde bulunan tüm deterministik ve meta-sezgisel algoritmalar aynı mantıkla sisteme uyarlanarak eklenebilir. Fakat bu çalışmada literatürdeki farklı çalışmaların aynı işlevi gören bölümleri birleştirilerek diğerlerinden farklı bir yöntem ortaya konulmuştur. Ortaya konulan çözüm modelinin farklı yanları, rota sayılarını azaltmak için Braysy (2001b,2003b)' deki (56-57) ***"route elimination"*** yöntemini, rota uzaklıklarını azaltmak içinse Gambardella v.d. (1999)' deki (50) "***ACS_TIME()***" yöntemini kullanmasıdır.

ZPARP-KDS ana prosedürü (ANA_ ZPARP)

ZPARP için tasarlanan çözüm algoritmasını, kurma ve geliştirme algoritmaları olarak ikiye ayıracak olursak; kurma aşamasında Solomon (1987)' deki INSERT1 kullanılmakta (21), daha sonra elde edilen bu başlangıç çözümü, Ψ_{iyi} literatürdeki bir çok araştırmada kullanılan 2-opt ve 3-opt (20) yöntemleri kullanılarak iyileştirilmektedir. Bu yerel iyileştirme yöntemleri kullanıldıktan sonra, Braysy (2001b, 2003b) araştırmalarında kullanılmış ve zincir atma yöntemi (ejection chains) tabanlı rota-eleme prosedürü kullanılarak, başlangıç çözümündeki rota sayıları azaltılmaya çalışılmaktadır (56-57). Rota-eleme prosedüründen sonra elde edilen çözüm ile Gambardella v.d. (1999)' deki global feremon güncelleme formülü

kullanılarak, karınca kolonisi algoritmasında kullanılacak başlangıç feremon miktarları belirlenir. Algoritmanın son aşamasında ise elde edilen en iyi araç sayısına (v) göre yapay karınca kolonileri yeni çözümler kurmaya başlamaktadır. Buradaki yapay karıncalar, Gambardella v.d. (1999)' deki *ACS_TIME(v)* karınca kolonisi gibi çalışarak, rotaların toplam uzaklığını en azlamaya çalışmaktadırlar (50). Ana prosedür aşağıdaki mantıkta çalışmaktadır.

procedure ANA_ ZPARP-KDS
{
$\Psi_{iyi} \leftarrow$ *INSERT1;*
$\Psi_{iyi} \leftarrow$ *2-OPT*(Ψ_{iyi}) *ve 3-OPT*(Ψ_{iyi})
$\Psi_{iyi} \leftarrow$ *ROTA_ELEME(* Ψ_{iyi}*);*
feremon[,] $\leftarrow$ *Feremon_Hesapla*(Ψ_{iyi} *);*
v $\leftarrow$ *aktif_araçlar_sayısı* (Ψ_{iyi}*);*
repeat
{ $\Psi_{iyi} \leftarrow$ *KKO_ZMN (v);* }
until *(iyileşme olmayana kadar)*
}

<u>*INSERT1 alt prosedürü*</u>

Solomon (1987)' ye dayanan bu algoritma, her rota kurma aşamasında depoya (0 düğümü) en uzak müşteriden işleme başlar. Daha sonra, rotalanmamış her müşteri iki aşamada incelenerek rotalar kurulur. Birinci aşamada, rotalanmamış müşteri için optimal ekleme (insertion) noktası belirlenir. Daha sonra ekleme karar kriteri hesaplanır. Algoritmada uzaklık ölçütüne dayanan karar kriteri kullanmaktadır. Bu kriterlerden ilki, $C1$, i ve j müşterileri arasına u müşterisini eklemenin maliyete getirdiği yükü, ikinci kriter, $C2$ ise eklemeyi yapmakla elde edilen tasarrufları (savings) hesaplamaktadır. Bu kriterler matematiksel olarak aşağıdaki gibidir;

$$C1 = d_{iu} + d_{uj} - A1 * d_{ij} \qquad [6.1]$$

$$C2 = A2 * d_{0u} - C1 \qquad [6.2]$$

Burada $A1 = 1$ ve $A2 = 2$ değerleri seçildiği takdirde, INSERT1 algoritması, genelleştirilmiş tasarruf algoritması gibi çalışmaktadır. INSERT1 algoritmasının adımları aşağıdaki şekildedir (20).

Adım1: Değişkenleri tanımla!

Adım2: Veri dosyasını oku!

Adım3: Müşteriler arasındaki mesafeleri hesapla!

Adım4: Değişkenlerin ilk değerlerini ata! P[]; sonra eklenecek müşteriler kümesinin değerlerini ata!

Adım5: P[] kümesi tükenene kadar

Adım5.1: k. rotaya yerleştirilecek 1. müşteriyi seç

Adım5.2: k. rotaya eklenecek U müşterisini seç

Adım5.3: U müşterisi rota kapasitesini artırıyorsa k←k+1 yap ve Adım5.1' e dön, yoksa Adım5.3.1-Adım5.3.... arasını tekrarla

Adım5.3.1: U müşterisini yerleştirecek en iyi yeri seç!

Adım5.3.2: Eklemenin uygunluğunu denetle!

Adım5.3.3: İleri itme (push-forward) uygunluğunu denetle!

Adım5.4: U' yu ekledikten sonra rotadaki diğer müşterileri ileri ittir ve Adım5.2' ye dön!

Adım6: Rota sayısı, toplam zaman ve toplam bekleme değerlerini hesapla!

2-OPT(Ψ_{iyi}) ve 3-OPT(Ψ_{iyi}) alt prosedürleri

Bu algoritmada kullanılan iyileştirme algoritmaları 2-opt ve 3-opt, ilk olarak GSP çözümü için kullanılan yöntemlerdir. Bu yöntemlerin açıklaması daha önceki bölümlerde yapılmıştır. Bu yöntemler ZPARP' ye uygulanırken, araç kapasitesi ve zaman penceresi kısıtlarına uyacak şekilde eklemeler yapılmıştır. Bu algoritmalar, bir sonraki *ROTA_ELEME()* aşamasına daha uygun çözümler göndermek için kullanılmaktadır (56-57).

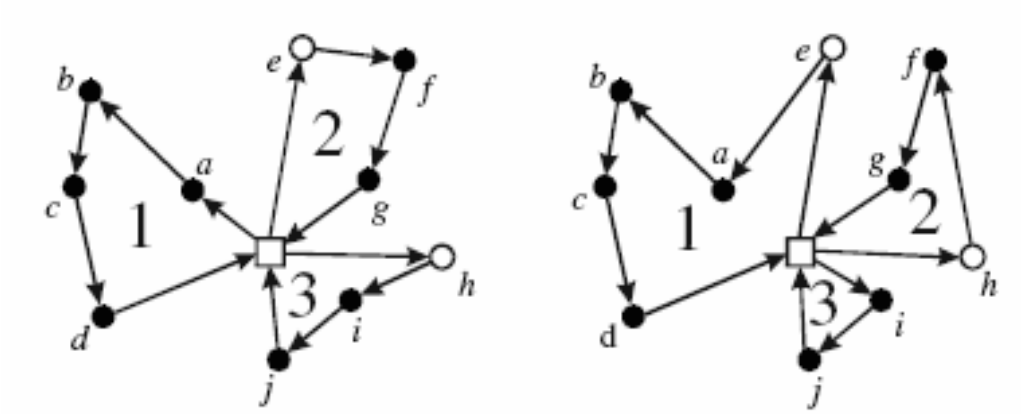
ROTA_ELEME( ) alt prosedürü (56-57)

Bu prosedür sadece rota sayısının azaltılması için kullanılır. Temel fikir, Glover (1991,1992) tarafından sunulan zincir atma (ejection chains –EC)' ye dayalı temel bir komşuluk yapısı kullanmaktır.

ZPARP kavramında EC' nin amacı, ilk önce c_i gibi bazı müşterileri rt_k rotasından çıkarmak ve daha sonra rt_l rotası tarafından servis yapılan bir c_j müşterisini rt_k rotasına eklemektir. Eğer c_j' yi eklemek uygun ise, c_i müşterisini de başka bir $rt_m \neq rt_k$ rotasına eklemek için girişim yapılır. Uygun bir ekleme pozisyonu bulunabilirse, zincir tamamlanır ve başka bir c_{j+1} müşterisi aynı şekilde bir zincir başlatmak için seçilir. Zincir atma ve ekleme prosedürleri, bir rotadan herhangi bir müşteriyi çıkarmadan o rotaya bir müşteri ekleyebilene dek devam eder (56-57).

EC algoritması mantığı şöyle Şekil 6.3' de örneklendirilmiştir. Burada 3 nolu rotanın elenmeye çalışıldığını ve h müşterisinin de başka rotalara eklenmeye çalışılan bir müşteri olduğunu varsayarsak; h müşterisinin 2 nolu rotaya eklenmeye çalışıldığını

görülmektedir. Ancak bu eklemenin doğrudan yapılması, araç ve zaman penceresi kısıtları nedeniyle uygun olmadığından 2 nolu rotadan e müşterisi çıkarılarak, h müşterisinin 2 nolu rotaya eklenmesi sağlanmıştır. Çıkarılan e müşterisi ise 1 nolu rotaya eklenmiştir. Ancak çıkarılan e müşterisinin başka rotalara eklenmesinin uygun olmaması durumunda yapılan bu değişiklikler geri alınır (56-57).



Şekil 6.3. Zincir atma (EC) algoritması mantığı

Elenecek rotaların seçimi için içinde bulundurduğu müşterilerin zaman penceresi genişliklerine bakılır. Zaman penceresi genişliği, deponun zaman penceresi genişliğinin %50' sinden daha dar olan müşteriler, zaman penceresi kısıtlı müşteri olarak adlandırılır ve elenecek rotaların seçiminde bu özelliğe sahip müşterileri daha az olan rotalar eleme işlemine tabii tutulur. Ancak araştırma uzayını sınırlı tutmak için tasarlanan algoritmada 4 ve daha az zaman kısıtlı müşteriye sahip rotaların elenmesi amaçlanmaktadır. Buna ek olarak müşteri sayısı çok olan rotaları (30 müşteri) elemek, zaman pencere kısıtlı müşteri sayısı az olsa bile zordur. Bu nedenlerden ötürü, elenecek rotaların makul sayıda müşteriye sahip olan ve zaman pencereli müşteri sayısı 4 ve daha az rotalardan seçilmesi uygun olmaktadır.

Eliminasyon için ele alınan rt_e rotasındaki tüm müşteriler, bu rotaya komşu rotalara $rt_n \in N(rt_e)$ eklenmeye çalışılmıştır. Komşu rotaların belirlenmesi, elenecek rotaya yakın müşterileri olup olmadığının kontrolü ile gerçekleşmektedir. Matematiksel olarak Eş 6.3' deki ifadeyle belirlenir.

$$N(rt_e) = \{ rt_j \in I^{rt} \mid \min_{\substack{ck \in rt_j \\ cp \in rt_e}} \{ d_{ckcp} \} \leq \bar{d} \} \qquad [6.3]$$

Bu eşitlikteki I^{rt}, tüm rotaların kümesini, $\bar{d}$ ise komşu rota yakınlık ölçütünü gösterir. Buradaki komşu rotalar kümesi $N(rt_i)$, EC algoritması başlamadan önce hazır çözümdeki tüm rotalar için bulunmaktadır. Bunun nedeni, sadece elenecek rotalardaki müşterilerin değil, diğer rotalardaki müşterinde gerektiği takdirde komşu rotalara eklenebilmesi gerekliliğindendir. Son olarak algoritmaya başlamadan önce, rotalar için n farklı artmayan sıra, bütün rotalardaki her müşterinin $C1$ ekleme maliyetine uygun oluşturulmaktadır.

Tüm ön hazırlık hesaplamaları bittikten sonra, rotalar zaman penceresi kısıtlı müşteri sayısına göre artan sırada eleme işlemine tutulur. Burada ilk önce elenecek rotadaki her müşteri basit ekleme yöntemiyle komşu rotalardan birine eklenmeye çalışılır. Ancak bu gerçekleşmediği takdirde, yukarıda değinilen EC algoritması adımları uygulanır.

rt_e rotası basit ekleme ve EC algoritmaları uygulanmasına rağmen elenemiyorsa, bu rotanın korunması gerekmektedir. Bu nedenle rt_e rotasının kaynaklarını mümkün olduğunca kullanmak için rt_e rotasına $rt_n \in N(rt_e)$ rotalarından mümkün olduğu kadar fazla müşteriyi eklemek için son bir algoritma uygulanır.

Araştırılacak zincirlerin sayısını kısıtlamak için, ll uzaklık kısıtı kullanılır. Hedef rotada uzaklığı bu kısıttan daha fazla artarsa ilgili ekleme ve EC değişiklikleri iptal edilir. Buna ek olarak EC prosedürünün maksimum uzunluğu $\bar{c}$ ile sınırlanmaktadır. Rota_Eleme algoritmasının çalışma prensibi aşağıdaki şekilde özetlenmiştir;

Adım1: $\bar{d}$: komşu rota belirleme uzaklığı, ll : uzaklık artış maksimum değeri, $\bar{c}$: zincir uzunluğu maksimum değeri, $\bar{z}$: elenecek rotaları sınırlamak için içinde

bulundurduğu zaman pencere kısıtlı müşteri sayısının limit değeri parametrelerini gir!

Adım2: Zaman penceresi genişliklerini hesapla, zaman penceresi genişlikleri deponun zaman penceresi genişliğinin %50' sinden az olan müşterileri "zaman pencere kısıtlı" olarak ata!

Adım3: Her rotadaki "zaman pencere kısıtlı" müşteri sayısının hesapla, rotaları bu değere göre artan sırada sırala!

Adım4: Sıralanmış rotaları ilk elemandan itibaren, "zaman penceresi kısıtlı müşteri sayısı" $\bar{z}$ sayısına ulaşana kadar elenecek rotalar kümesine ekle!

Adım5: Tüm rotalar için kendine en yakın diğer rotalar kümesini bul!

Adım6: Tüm müşterilerin rotalara ekleme maliyeti $C1$'i hesapla! Bu maliyetlere göre rotalara eklenme öncelik sırasını hesapla!

Adım7: Elenecek rotayı seç! Elenecek rotalar kümesi boşsa Adım 13' e git!

Adım8: Elenecek rt_e rotasındaki müşteriyi c_i seç ve komşu rotalardan $C1$ değeri en az olan birine $rt_n \in N(rt_e)$' ye basit ekleme ile ekle! Ekleme başarılı ise Adım 11' e git. Yoksa devam et!

Adım9: $rt_n \in N(rt_e)$ rotasından bir adet c_d müşterisini çıkar ve c_i müşterisini rt_n rotasına, c_d müşterisini $rt'_n \in N(rt_n)$ rotasına eklemeye çalış. Her iki eklemede başarılı olursa Adım 11' a git. Aksi takdirde, bu aşamanın başındaki duruma dön ve $d \leftarrow d+1$ yaparak Adım 9' u tekrar et (c_d, rt_n rotasının son elemanı ise rt_{n+1} rotasının ilk elemanı c_d olarak ata).

Adım10: Eğer tüm $rt_n \in N(rt_e)$ rotaları denenmiş ve c_i ler uygun pozisyonlara eklenememiş ise, art arda müşteri çıkarılacak rota zincir sayısını 1 arttır ($\bar{c}$ sayısına kadar) ve Adım 9' a benzer olarak çıkarma ve ekleme işlemlerini gerçekleştir.

Adım11: rt_e' deki tüm müşteriler araştırıldıysa Adım 12' ye git. Aksi takdirde $i \leftarrow i+1$ yap ve Adım 8' e git!

Adım12: rt_e rotasındaki tüm müşteriler dağıtılmış ise elenecek diğer rotayı seç ve Adım 8' e git! Aksi takdirde bu rotaya en yakın $rt_n \in N(rt_e)$ rotasından maksimum sayıdaki müşteriyi, elenemeyen rt_e rotasına ekle, rt_n rotasını zaman pencere kısıtlı müşteri sayısı uygunsa, elenecek rotalar listesine dahil et ve Adım 7' ye git!

Adım13: Elenecek rotalar kümesi boşalmışsa, yeni rota kümesini döndür ve algoritmayı bitir (56-57).

KKO_ZMN (v) alt prosedürü (50)

KKO_ZMN(v) algoritması Gambardella v.d., (1999)' da geliştirilmiş *ACS_TIME(v)*' a benzer çalışan yapay bir karınca kolonisi sistemidir. *KKO_ZMN()*'da *ACS_TIME(v)* gibi, önceki aşamadan gelen araç/rota sayısı *v*' ye göre çalışarak eldeki mevcut rota uzaklığını en azlamaya çalışır.

Algoritmanın bu bölümüne başlanılmadan önce *Feremon_Hesapla*(ψ^{iyi}) fonksiyonu kullanılarak, *KKO_ZMN()* kolonisine başlangıç feremon miktarları atanır. Feremon miktarları Eş.6.4' de görülen fonksiyon yardımıyla hesaplanır.

$$\tau_{ij} = (1-\rho).\tau_{ij} + \rho / J\psi^{iyi} \qquad \forall (i,j) \in \psi^{iyi} \qquad [6.4]$$

Bu eşitlikte, τ_{ij}; bir önceki çözümde arklara bırakılan feremon miktarını, ρ; feremonun buharlaşma katsayısı, $J\psi^{iyi}$; bir önce bulunan ψ^{iyi} çözümün uzunluğu

olmak üzere $(1-\rho).\tau_{ij}$ değerinde $\tau_{ij}=0$ alınarak başlangıç feremon miktarları hesaplanır. Ayrıca *aktif_araçlar_sayısı* (ψ^{iyi}) fonksiyonu ile de ψ^{iyi} çözümündeki araç/rota sayısı belirlenir.

KKO_ZMN() kolonisi en kısa zamanda bir tur oluşturmayı amaçlayan bir karınca kolonisi sistemidir. Burada m yapay karınca, $\psi^1,.....,\psi^m$ problem çözümlerini bulmak için aktive edilirler. Her çözüm *new_active_ant()* algoritmasının çağırılmasıyla bulunur. $\psi^1,.....,\psi^m$ çözümleri bulunduğunda bunlar tek tek ψ^{iyi} çözümüyle karşılaştırılır. Çözümlerden birinin ψ^{iyi}' den iyi olması durumunda bu çözüm ψ^{iyi} olarak güncellenir.

subprocedure KKO_ZMN (v);

//{sıfırla}

initialize v' yi kullanarak feremon ve veri yapılarını başlangıç haline getir!

{ foreach bütün k karıncaları için $(k=1,.....,m)$ *//{döngü}*

{ $\psi^k \leftarrow$ *new_active_ant(k); // {* ψ^k *çözümünü inşaa et}}*

for(int $k=1;\ k\le m;\ k++$)

{ if ($J\psi^k < J\psi^{iyi}$)

{ $\psi^{iyi} \leftarrow \psi^k$; }

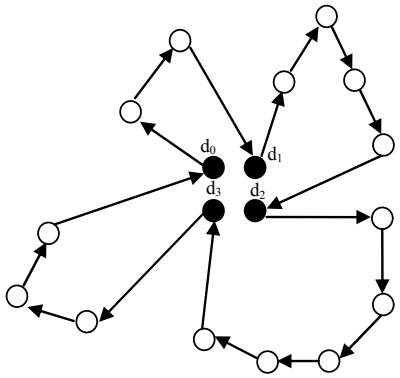
else continue; }

$\tau_{ij}=(1-\rho).\tau_{ij}+\rho / J\psi^{iyi}$ $\forall(i,j)\in\psi^{iyi}$ *{ küresel güncellemeyi gerçekleştir}*

return ψ^{iyi} ; }

KKO_ZMN (v) çözüm modeli (50)

KKO_ZMN (v) çözüm modelinde her karınca Şekil 6.4' deki gibi bir tur oluşturmaktadır. Bunun için, depo tüm müşterilere olan bağlantılarıyla birlikte *v* araç sayısı kadar çoğaltılır. Depo ve kopyaları arasındaki uzaklıklar sıfıra eşitlenir. Bu yaklaşımla zaman pencereli araç rotalama problemi GSP' ne benzemektedir. Bu modelde uygun çözüm, bütün düğümlerin bir defa ziyaret edildiği bir hamilton çevrimidir. Çoğaltılan tüm depoların koordinatları aynıdır, ancak şekilde *(d_0, ..., d_3)* ayrı gösterilmiştir.



Şekil 6.4. Dört araçlı çiftlenmiş depolu uygun çözüm

Bu gösterimin tek depolu gösterime göre faydası depoya olan yönlerde daha az feremon çekiciliğinin bulunmasıdır. Bu durum çözüm kurucu algoritmanın kalitesini pozitif olarak etkilemektedir.

KKO_ZMN (v) çözüm kurucu algoritması (50)

KKO_ZMN (v)' nun çözüm kurucu algoritmasında, her karınca rassal seçilen bir depo kopyasından çözüm oluşturmaya başlar. Daha sonraki adımlarda daha önceden ziyaret edilmemiş uygun müşterileri, zaman penceresi ve araç kapasitesi kısıtlarını aşmadan ziyaret eder. Uygun müşteriler kümesinde ziyaret edilmemiş olan

çoğaltılmış depo düğümleri de bulunur. Karıncalar bir sonra gideceği düğümü ∂_0 olasılıkla kullanma (exploitation), $(1-\partial_0)$ olasılıkla keşif (exploration) yöntemi ile seçer. Kullanma durumunda kısıtları uygun $j \in N_i^k$ müşterilerinden en büyük $\tau_{ij}.[\eta_{ij}]^\beta$ değerlisi, keşif durumunda ise $j \in N_i^k$ müşterisi Eş.6.5' de gösterilen p_{ij} olasılığı ile seçilir.

$$p_{ij} = \begin{cases} \dfrac{\tau_{ij}.[\eta_{ij}]^\beta}{\sum_{j \in N_i^k} \tau_{ij}.[\eta_{ij}]^\beta}, j \in N_i \\ 0 \qquad\qquad\qquad j \notin N_i \end{cases} \qquad [6.5]$$

n_{ij} çekiciliği, d_{ij} uzaklığı, $[e_i, l_i]$ zaman penceresi miktarı dikkate alınarak hesaplanır. Bir karıncanın bir müşteriden diğerine hareketi sonrası Eş. 6.6' daki yerel feremon güncellemesi gerçekleştirilir.

$$\tau_{ij} = (1-\rho).\tau_{ij} + \rho/(n.J\psi^{I1}) \qquad [6.6]$$

Bu eşitlikte ψ^{I1} başlangıç kurucu algoritması *INSERT1'* in toplam uzunluğunu, n ise toplam müşteri sayısını göstermektedir.

Kurucu algoritma sonunda bazı müşteriler rotalara dahil edilmeyebilir. Bu neden ile bu müşteriler için bir ekleme algoritması çalıştırılır. Ekleme, tüm ziyaret edilmemiş müşterilerin, taleplerinin azalan sırada sıralanması ile başlar. Her müşteri, en az maliyet ile eklenebileceği yerlere eklenir, uygun ekleme yeri bulamayana kadar tekrarlanır. Son olarak yine rotalanmamış müşteriler kalmışsa bu çözüm uygun olmadığından diskalifiye edilir.

Algoritmanın son aşamasında uygun çözümler Braysy (2001b, 2003b)'deki *ICROSS* algoritması ile iyileştirilmeye çalışılmıştır. *new_active_ant(k)* ve *ICROSS* algoritması aşağıdaki gibi çalışır.

subprocedure new_active_ant (k); (50)

Adım1: $\psi_k \leftarrow \langle i \rangle$; $simdiki\_zaman_k \leftarrow 0$; $yük_k \leftarrow 0$; (Sıfırla)

Adım2: k karıncasının ziyaret edebileceği uygun müşteri kümesi $N_i^k = \{\}$ olana dek bu adım tekrarlanır.

Adım2.1: $dagitim\_zamanı_j \leftarrow max\{simdiki\_zaman_k + d_{ij} + s_i, e_j\}$;

Adım2.2: $delta\_zaman_{ij} \leftarrow (dagitim\_zamanı_j - simdiki\_zaman_k)$;

Adım2.3: $uzaklik_{ij} \leftarrow delta\_zaman_{ij} * (l_i - simdiki\_zaman_k)$;

Adım2.4: $\eta_{ij} \leftarrow (1.0 / uzaklik_{ij}).\ \eta_{ij}$;

Adım2.5: ∂_0 olasılıkla kullanma, $(1-\partial_0)$ olasılıkla keşif ile $\langle j \rangle$' yi şeç!

Adım2.6: $\psi_k \leftarrow \psi_k + \langle j \rangle$; $simdiki\_zaman_k \leftarrow dagitim\_zamanı_j$; $yük_k \leftarrow yük_k + q_j$;

Adım2.7: $\langle j \rangle$ düğümü depoya ait ise $simdiki\_zaman_k \leftarrow 0$; $yük_k \leftarrow 0$;

Adım2.8: $\tau_{ij} \leftarrow (1-\rho).\tau_{ij} + \rho / (n.J\psi^{I1})$

Adım2.9: $i \leftarrow j$;

Adım3: $\psi_k \leftarrow ekleme\_prosesi()$; (rotalanmamış müşteri varsa)

Adım4: $\psi_k \leftarrow ICROSS(\psi_k)$; (mümkün çözüm yerel arama yöntemi ile iyileştirilmeye çalışılır)

subprocedure ICROSS (ψ_k); (56-57)

Adım1: r_1 ve r_2 gibi iki uygun rota ile başla ve *segt* ve *inv* değerlerini ata!

Adım2: 3. ve 4. adımlarını 0'dan *segt'e* kadar parça uzunlukları (segment lenghts) ve r_1 rotalarındaki bütün c_i müşterileri ve c_i' den l_s yerleri içinde r_2 rotası içindeki tüm c_j müşterileri için tekrar et! (c_i ve c_j müşterileri parçaların (segment) içerdiği ilk müşterilerdir)

Adım3: $s_1 \subseteq r_1$ ve $s_2 \subseteq r_2$ parçalarını birbirleri ile değiştirmeye çalış. Eğer *inv* true ise, parçalar içindeki müşteri sıralarını da ters çevir ve ters çevrilmiş $s_1' \subseteq r_1$ ve $s_2' \subseteq r_2$ parçalarını da birbirleri ile değiştirmeye çalış.

Adım4: Eğer r_1 ve r_2 değerleri iyileştiyse, r_1 ve r_2 rotalarını Adım3' te yapılan değişikliklere göre güncelle!

Adım5: En çok iyileştirilmiş 2 rota çiftini geri döndür!

6.2.4. ZPARP-KDS yönetim modülü

ZPARP-KDS yönetiminin görevleri şu şekilde özetlenebilir;

- Dosya modülünden örnek problem verilerini okuyarak, çözüm modülüne bu verileri yüklemek,

- Talep modülüne gelen müşteri taşıma taleplerinin, veritabanında daha önceden var olan müşteri lokasyon verileri ile eşleştirerek CBS modülüne, sayısal haritalarda gösterilmek üzere göndermek,

- CBS modülü sayısal haritaları üzerindeki tüm problem verilerinin, kullanıcı tarafından yapılan değişiklerle birlikte çözüm modülüne gönderilerek rotalama sonuçlarının elde edilmesi,

- Rotalama sonuçlarının yine CBS modülü üzerinde görsel olarak gösterilmesi,

- Rotalama sonuçlarının raporlanması için raporlama modülüne aktarılması,

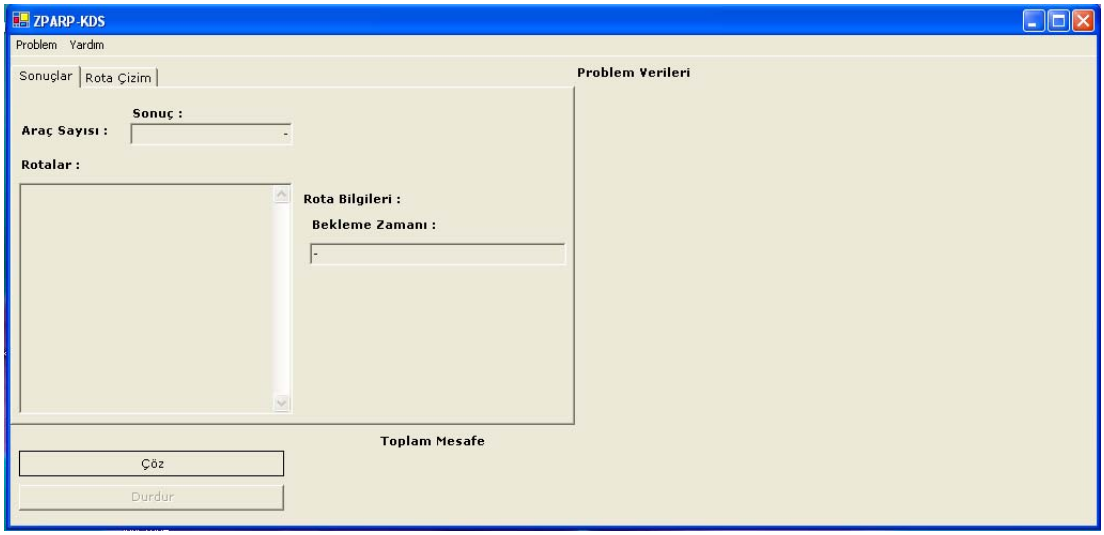
- Rotalara araç tahsisi işlemlerinin yapılması, araç iş emirlerinin araç sürücülerine verilmesi,

- Arayüz modülü yardımıyla kullanıcıdan gelebilecek tüm emirleri işlemek gibi faaliyetlerdir.

6.2.5. ZPARP-KDS arayüz modülü

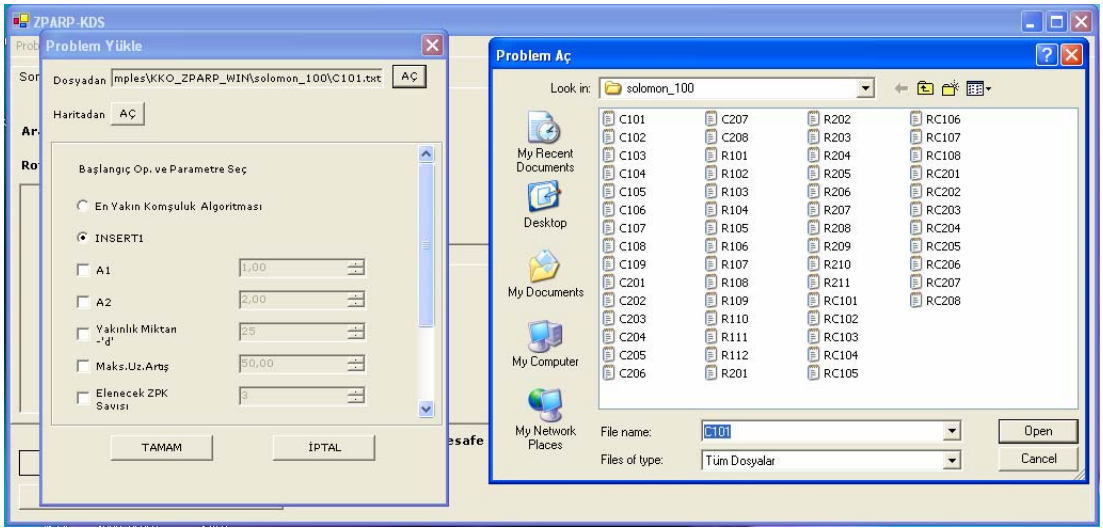
Bu bölümde ZPARP-KDS' nin çözüm modülüne eklenebilecek algoritmaların çalışılabilirliğini test etmek amacıyla tasarlanan dosya modülünün kullanıcı ile bağlantı kuran arayüzleri tanıtılmaktadır. Bu arayüzler C# dili ile programlanmıştır. İlgili kodlar Ek-5' de verilmiştir. Aşağıda, dosya modülünde anlatılan Solomon (1987) (21) problem veri setleri kullanılarak uygulamanın nasıl çalıştığı anlatılmıştır.

İlk olarak ZPARP-KDS programını çalıştırıldığında Şekil 6.5' deki açılış ekranı ortaya çıkar. Buradan, "Problem" menüsüne girilerek, çıkan "Problem Yükle" (Şekil 6.6 ve Şekil 6.7) penceresinde kullanıcı isteğine göre "En yakın komşuluk" veya "INSERT1" başlangıç çözüm operatörü seçilir ve sistemdeki çözüm algoritmasının parametreleri girilir.

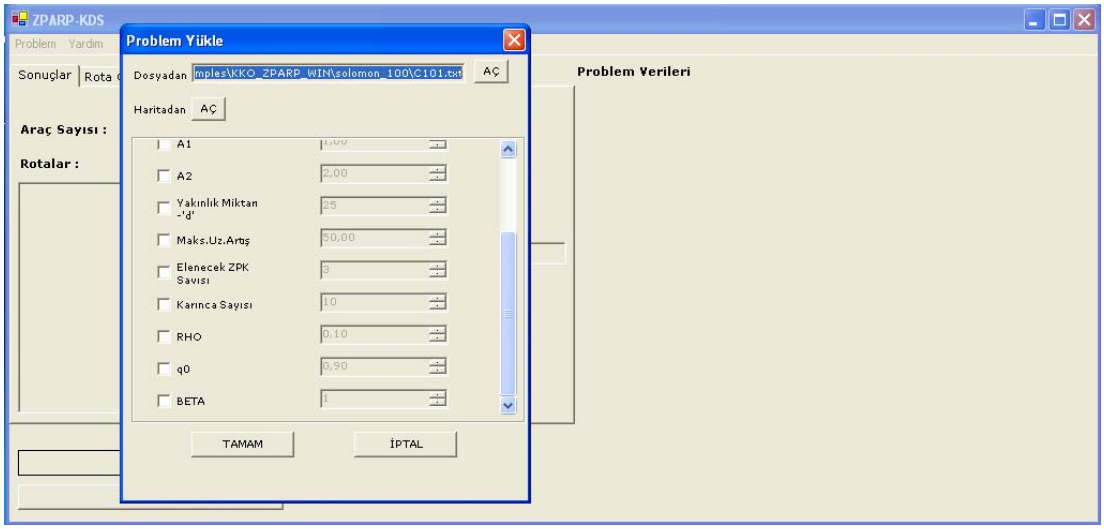


Şekil 6.5. ZPARP-KDS açılış ekranı

Daha sonra dosya modülündeki Solomon (1987) (21) problemleri kullanılacağından "Dosyadan" başlığının karşısındaki "Aç" tuşuna basılarak problemlerin bulunduğu dosya seçilerek sisteme yüklenir. Ayrıca aynı penceredeki "Haritadan" başlığında ise sistemin CBS ile entegre edilen harita modülünün çalışması planlanmıştır.

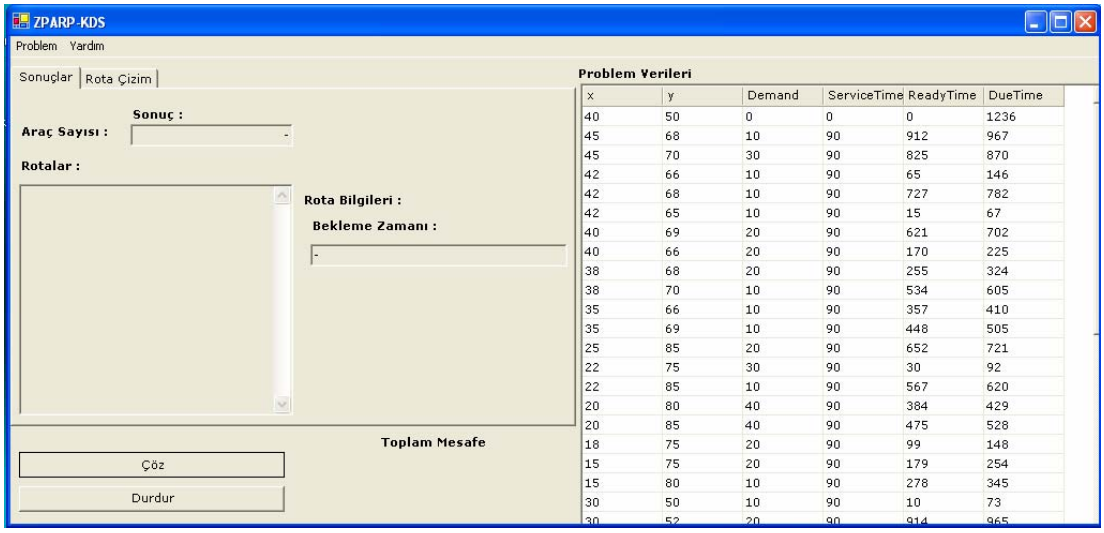


Şekil 6.6. ZPARP-KDS problem yükleme ekranı ve problem parametreleri



Şekil 6.7. ZPARP-KDS problem parametreleri

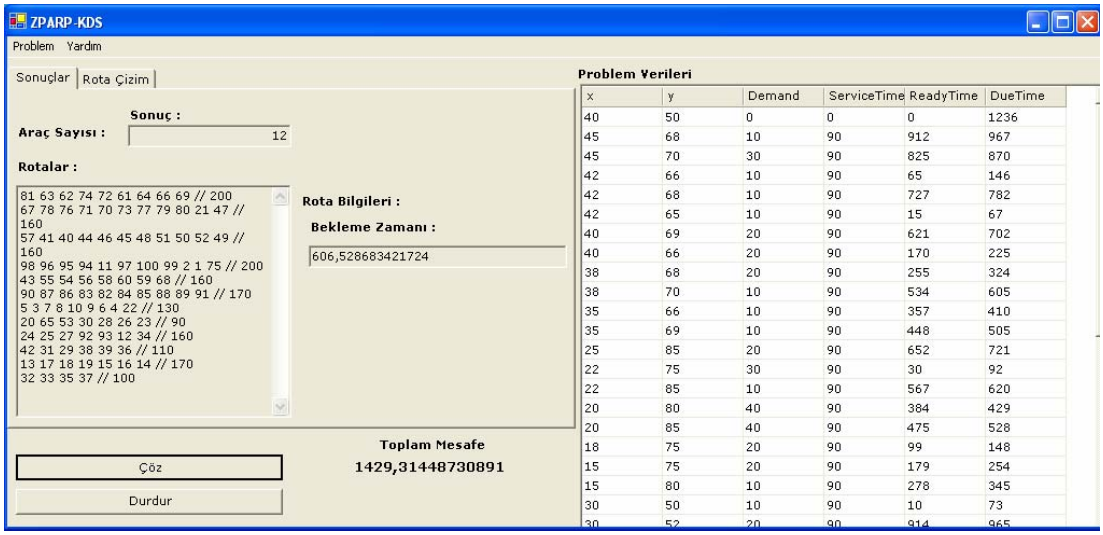
Şekil 6.8' de görülen problem verileri sisteme yüklendikten sonra "Çöz" butona basarak çözüm modülü çalıştırılır.



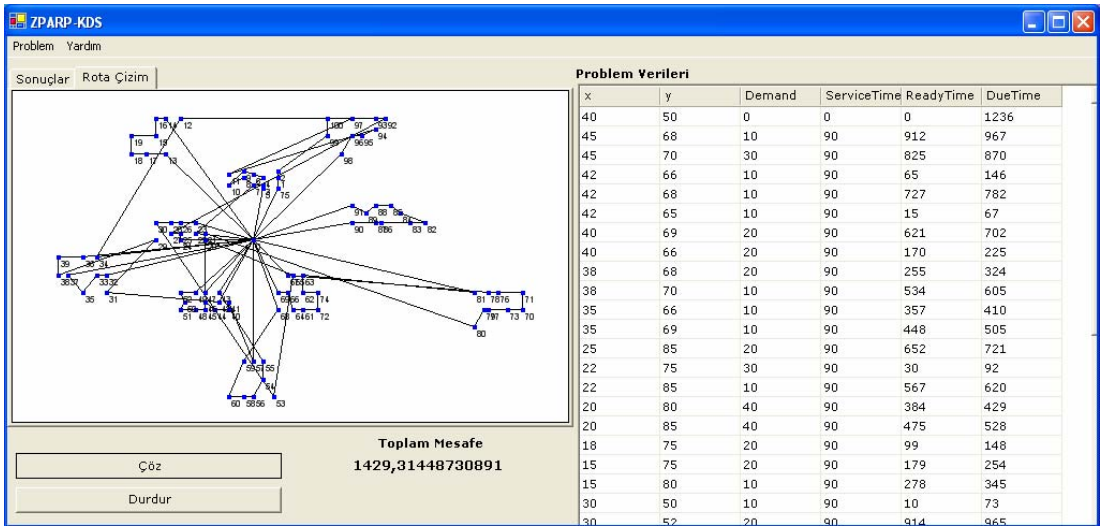
x	y	Demand	ServiceTime	ReadyTime	DueTime
40	50	0	0	0	1236
45	68	10	90	912	967
45	70	30	90	825	870
42	66	10	90	65	146
42	68	10	90	727	782
42	65	10	90	15	67
40	69	20	90	621	702
40	66	20	90	170	225
38	68	20	90	255	324
38	70	10	90	534	605
35	66	10	90	357	410
35	69	10	90	448	505
25	85	20	90	652	721
22	75	30	90	30	92
22	85	10	90	567	620
20	80	40	90	384	429
20	85	40	90	475	528
18	75	20	90	99	148
15	75	20	90	179	254
15	80	10	90	278	345
30	50	10	90	10	73
30	52	20	90	914	965

Şekil 6.8. ZPARP-KDS problem verileri

Elde edilen sonuç rotalar Şekil 6.9' da sayısal olarak, Şekil 6.10' da ise görsel olarak kullanıcıya sunulur.



Şekil 6.9. ZPARP-KDS sonuçlar



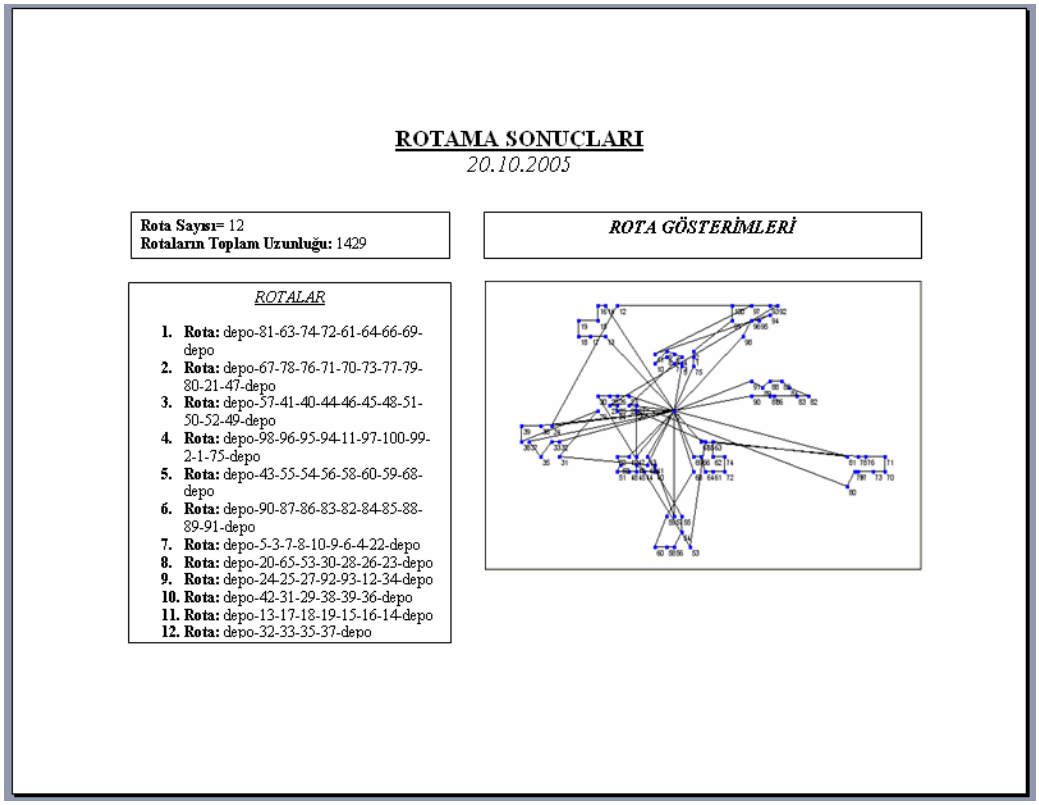
Şekil 6.10. ZPARP-KDS rotalar

6.2.6. ZPARP-KDS kullanıcı

ZPARP-KDS' nin kullanıcısının özel ve kamu alanında zaman pencereli araç rotalama faaliyetlerini planlayan görevlilerin olması tasarlanmıştır.

6.2.7. ZPARP-KDS raporlama modülü

Raporlama modülü çözüm modülü sonuçları ve rotalarının, rotaların araçlara tahsis emirlerinin, dağıtılan yüklerin türü ve miktarının istenildiğinde kağıt ortamına aktarılmasına imkan verecektir. Raporlama modülünden örnek bir rapor taslağı Şekil 6.11' de gösterilmektedir.



ROTAMA SONUÇLARI

20.10.2005

Rota Sayısı= 12
Rotaların Toplam Uzunluğu: 1429

ROTA GÖSTERİMLERİ

ROTALAR

1. **Rota:** depo-81-63-74-72-61-64-66-69-depo
2. **Rota:** depo-67-78-76-71-70-73-77-79-80-21-47-depo
3. **Rota:** depo-57-41-40-44-46-45-48-51-50-52-49-depo
4. **Rota:** depo-98-96-95-94-11-97-100-99-2-1-75-depo
5. **Rota:** depo-43-55-54-56-58-60-59-68-depo
6. **Rota:** depo-90-87-86-83-82-84-85-88-89-91-depo
7. **Rota:** depo-5-3-7-8-10-9-6-4-22-depo
8. **Rota:** depo-20-65-53-30-28-26-23-depo
9. **Rota:** depo-24-25-27-92-93-12-34-depo
10. **Rota:** depo-42-31-29-38-39-36-depo
11. **Rota:** depo-13-17-18-19-15-16-14-depo
12. **Rota:** depo-32-33-35-37-depo

Şekil 6.11. Rotalama modülünden örnek bir rapor taslağı

7. SONUÇ VE ÖNERİLER

Ekonomik faydası olan değerlerin, çok bulundukları yerlerden ihtiyaç duyulan yerlere taşınması faaliyetleri neredeyse insanlık tarihi ile ortaya çıkmıştır. İpek ve baharat yolları bu taşıma faaliyetlerine örnek verilebilir. Ancak o çağlardan 21. yüzyıla değişen, bu taşıma faaliyetlerinin yoğunluğudur. Günümüzde her dakika binlerce araç bir yerden bir yere ekonomik, sosyal ve kültürel değerler taşımak için yollara çıkmaktadırlar.

Taşıma faaliyetlerinin bu yoğunluğu bu işi planlayan kişiler için büyük sorumluluklar getirmektedir. Planlayıcılar, taşınan ekonomik değerlerin en az maliyetle ve tam zamanında yerlerine ulaşmasını sağlamak zorundadır. Çünkü zamanında yerlerine ulaşmayan ekonomik değerler pazar ve prestij kaybı gibi maddi manevi yitiklere neden olacaktır. Böyle bir ortamda planlayıcıların bu sorumluluğunu azaltmak ve daha doğru kararlar (yarı yapısal ve yapısal olmayan) verilmesini sağlamak için lojistik ve dağıtım firmaları bilgi teknolojilerine yatırım yapmak zorundadırlar. Ancak bu şekilde bu firmalar ayakta kalabilir ve gelişebilir.

Lojistik ve dağıtım firmalarının yarı yapısal ve yapısal olmayan kararları kolay alması için bünyesinde bulunan diğer bilgi teknolojileri ile uyumlu çalışabilecek karar destek sistemlerine ihtiyaçları vardır.

Bu nedenle bu çalışmada zaman pencereli araç rotalama problemi ortamında karar alacak firmalara yardımcı olacak bir sistem olan ZPARP-KDS tasarlanmıştır. Bu sistem, zamanlama faktörünün çok önemli olduğu okul ve servis otobüsleri çizelgelemesi, akaryakıt dağıtımları, gazete dağıtımları, posta dağıtımları, güvenlik devriye kontrolleri, evsel ve kentsel atık toplama faaliyetleri ve zincir mağaza dağıtımları gibi ZPARP' ye uyan bir çok gerçek hayat planlama problemini modelleyip, çözmek için tasarlanmıştır. Böyle bir sistemin tasarlamasında diğer bir neden ise ZPARP çözüm algoritmalarını kullanıcı ihtiyaçlarına göre opsiyonlu seçerek kullanıcıya çeşitli çözüm alternatifleri sunmasını sağlamaktır.

Tasarlanan ZPARP-KDS, kullanıcı haricinde dosya, harita, çözüm, yönetim, arayüz ve raporlama olmak üzere toplam 6 modülden oluşmaktadır. Dosya modülü; Solomon (1987) (21) örnek problemleri başta olmak üzere ZPARP biçimdeki problem kümelerini kullanarak, çözüm modülündeki ZPARP algoritmalarının işlerliğini test etmekte, harita modülü; gerçek zamanlı talep verileri toplamakta, verileri sayısal haritalar üzerinde göstermekte, veriler üzerinde işlem yapmakta, verileri çözüm modülüne göndermekte ve çözüm sonuçlarını tekrar sayısal harita üzerinde göstermekte, çözüm modülü; kullanıcının istediği hassasiyetin derecesine göre sistemde mevcut algoritmalar ile ZPARP çözümünü sağlamakta, yönetim modülü; tüm modüller arasındaki veri iletişimini sağlamakta, arayüz modülü; kullanıcı ile program arasında iletişimi sağlamakta, raporlama modülü ise; kullanıcının istediği tüm çıktıları üretmektedir.

Sonuç olarak, ZPARP-KDS' nin lojistik ve dağıtım firmalarında kullanılması, zaman pencereli araç rotalama problemleri ile uğraşan karar vericilerin, karar kalitelerinin artırılmasına, karar verme sürelerinin azaltılmasına ve verilen kararlarda verimlilik artışına neden olacaktır. Bunun yanısıra, verilen kaliteli ve verimli kararlar, şirketlerin kısa dönem araç işletme, orta ve uzun dönem araç yatırım maliyetlerini azaltarak karlılığını artıracak, doğru zamanlar içinde yapılan teslimatlar, müşteri memnuniyetlerinin artmasına ve doğru zamanlarda toplanan atık maddeler, çevre kirliliğini azaltarak toplumdaki bireylerin sağlıklı bir ortamda yaşamasına imkan tanıyacaktır. Ayrıca ZPARP' de doğru planlama kararları, yukarıda sayılanların dışında, firma müşterileri ve toplum bireyleri için, hemen tespit edilmesi zor daha bir çok maddi ve manevi faydalar ortaya çıkaracaktır.

ZPARP-KDS çalışmasından sonra yapılacak çalışmalarda; bir harita modülünün CBS ile ilişkilendirilmesi, sistemin müşteri taleplerini online alabilmesi için internet sunucusu üzerinden çalışması, istenen çözüm kalitesine göre "deterministik" ve farklı "meta-sezgisel" algoritmalar arasında seçim yapılabilmesi, ZPARP dışında farklı tür araç rotalama problemleri çözüm algoritmalarının bu sisteme entegre edilmesi, araç tahsis işlemlerinin sistem yardımıyla yapılabilmesi gibi işlevlerin sisteme eklenmesi önerilmektedir.

KAYNAKLAR

1. Çetinyokuş, T., "Tedarikçi Performansının Değerlendirilmesi için Bir Karar Destek Sistemi", Yüksek Lisans Tezi, ***Gazi Üniversitesi Fen Bilimleri Enstitüsü***, Ankara, 24-43 (2003).

2. Turban, E., "Decision Support Systems and Expert Systems: Management Support Systems 2nd ed.," ***Macmillan Publishing Company***, New York, 6-30, 105-131 (1990).

3. Gökçen, H., "Yönetim Bilgi Sistemleri 2.baskı", ***EPİ Yayınları***, Ankara, 53 - 57, (2005).

4. Laudon, K.C., Laudon, J.P., "Management Information Systems 7th ed.", ***Prentice Hall International***, New Jersey, 43-45, 404-408 (2002).

5. Eiselt, H.A., Sandblom, C.L., "Integer programming and network models", ***Springer-Verlag***, Berlin, 315-341 (2000).

6. Ball, M.O., Magnanti, T.L., Monma, C.L., Nemhauser G.L., "Handbooks in operations research and management science: (Vol 7) Network models", ***Elsevier***, Amsterdam, 225-330 (1995).

7. Drummond L.M.A., Ochi L.S., Vianna D.S., "An asynchronous parallel metaheuristic for the period vechicle routing problem", ***Future Generation Computer Systems,*** 17: 379-386 (2001).

8. Ruiz R., Maroto C., Alcaraz J., "A decision support system for a real vehicle routing problem", ***European Journal of Operational Research***, 153: 593-606 (2004).

9. Cordeau, J.-F., Gendreau M., Laporte G., "A Tabu Search Heuristic for Periodic and Multi-Depot Vechicle Problems" , ***Networks***, 30: 105-119 (1997).

10. Wassan, N.A., Osman, I.H., "Tabu search variants for the mix fleet vehicle routing problem", ***Journal of the Operational Research Society***, 53: 768-782 (2002).

11. Chao, I-M., Golden, B., Wasil, E., " A computational study of a new heuristic for the site-dependent vehicle routing problem", ***INFOR***, 37:3 (1999).

12. Ioannou, G., Kritikos, M., "A synthesis of assignment and heuristic solutions for vehicle routing with time windows", ***Journal of the Operational Research Society***, 55(1): 2-11 (2004).

13. Tan, K.C., Lee, L.H., Qu, K., "Artificial intelligence heuristics in solving vehicle routing problems with time window constraints", ***Engineering Applications of Artificial Intelligence***, 14:825-837 (2001).

14. Desrosiers, J., Soumis F., Desrochers, M., Sauvé, M., "Methods for routing with time windows", ***European Journal of Operational Research***, 23:236-245 (1986).

15. Kolen, A.W.J., Rinnooy Kan, A.H.G., Trienekens, H.W.J.M., "Vehicle routing with time windows", ***Operations Research***, 35(2): 266-273 (1987).

16. Min, H., "A multiobjective vehicle routing problem with soft time windows: the case of a public library distribution system", ***Socio-Economic Planning Sciences***, 25(3): 179-188 (1991).

17. Desrochers, M., Desrosiers, J., Solomon, M., ""A new optimization algorithm for the vehicle routing problem with time windows", ***Operations Research,*** 40(2): 342-354 (1992).

18. Kohl, N., Madsen, O.B.G., "An optimization algorithm for the vehicle routing problem with time windows based on langrangian relaxation", ***Operations Research***, 45(3): 395-406 (1997).

19. İnternet: Kallehauge, B., Larsen, J., Madsen, O.B.G., "Lagrangean duality applied on vehicle routing with time windows Experimental Results", http://www.optimization-online.org/DB_FILE/2001/11/401.pdf (2001).

20. Baker, E.K., Schaffer, J.R., "Solution improvement heuristics for the vehicle routing problem with time window constraints", ***American Journal of Mathematical and Management Sciences***, 6(3-4): 261-300 (1986).

21. Solomon, M.M., "Algorithms for the vehicle routing and scheduling problems with time window constraints", ***Operations Research,*** 35(2): 254-265 (1987).

22. Van Landeghem, H.R.G., "A bi-criteria heuristic for the vehicle routing problem with time windows", ***European Journal of Operational Research***, 36: 217-226 (1988).

23. Koskosidis, Y.A., Powell, W.B., "Application of optimization based models vehicle routing and scheduling problems with time window constraints", ***Journal of Business Logistics,*** 11(2): 101-128 (1990).

24. Ahn, B.H., Shin, J.Y., "Vehicle routing with time windows and time-varying congestion", ***Journal of the Operational Research Society,*** 42(5): 393-400 (1991).

25. Salvelsbergh, M.W.P., "The vehicle routing problem with time windows: minimizing route duration", ***ORSA Journal on Computing***, 4(2): 146-154 (1992).

26. Koskosidis, Y.A., Powell, W.B., Solomon M.M., "An optimization-based heuristic for vehicle routing and scheduling with soft time window constraints", ***Transportation Science***, 26(2): 69-85 (1992).

27. Potvin, J-Y., Rousseau, J-M., "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows", ***European Journal of Operational Research***, 66: 331-340 (1993).

28. Foisy, C., Potvin, J.Y., "Implementing an insertion heuristic for the vehicle routing on parallel hardware", ***Computers and Operations Research***, 20(7): 737-745 (1993).

29. Balakrishnan, N., "Simple heuristics for the vehicle routing problem with soft time windows", ***Journal of the Operational Research Society***, 44(3): 279-287 (1993).

30. Bramel, J., Li, C.L., SimchiLevi, D., "Probabilistic analysis of a vehicle routing problem with time windows", ***American Journal of Mathematical and Management Sciences***, 13(3-4): 267-322 (1993).

31. Derigs, U., Grabenbauer, G., "Intime-a new heuristic approach to the vehicle routing problem with time windows, with a bakery fleet case", ***American Journal of Mathematical and Management Sciences***, 13(3-4): 249-266 (1993).

32. Russell, R.A., "Hybrid heuristics for the vehicle routing problem with time windows", ***Transportation Science***, 29(2): 156-166 (1995).

33. Johns, S., "Heuristics to schedule service engineers within time windows", ***Journal of the Operational Research Society***, 46(3): 339-346 (1995).

34. Shieh, H.M., May, M.D., "On-line vehicle routing with time windows - Optimization-based heuristics approach for freight demands requested in real-time", ***Land Use and Transportation Planning and Programming Applications Transportation Research Record***, 1617: 171-178 (1998).

35. Braysy, O., Gendreau, M., "Vehicle routing problem with time windows, part1: Route construction and local search algorithms", ***Transportation Science***, 39(1): 104-118 (2005a).

36. İnternet: Liu, F.H., Shen, S.Y., "A method for vehicle routing problem with multiple vehicle types and time windows", http:// nr.stic.gov.tw/ ejournal/ ProceedingA/ v23n4/ 526-536.pdf (1999).

37. Cordone, R., Calvo,R.W., “A heuristic for the vehicle routing problem with time windows”, ***Journal of Heuristics***, 7: 107-129 (2001).

38. Braysy, O., “Fast local searches for the vehicle routing problem with time windows”, ***INFOR***, 40(4): 319-330 (2003a).

39. Thangiah, S.R., Nygard, K.E., Juell, P.L., “GIDEON: A genetic algoritm system for vehicle routing with time windows”, ***Proceedings Seventh IEEE Conference on Artificial Intelligence Applications***, IEEE Computer Society Press, Los Alamitos, CA, 322-328 (1991).

40. Garcia, B-L., Potvin, J-Y., Rousseau, J-M., “A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints”, ***Computers and Operations Research***, 21(9): 1025-1033 (1994).

41. Braysy, O., Gendreau, M., “Vehicle routing problem with time windows, part2: Metaheuristics”, ***Transportation Science***, 39(1): 119-139 (2005b).

42. Potvin, J-Y., Kervahut, T., Garcia, B-L., Rousseau, J-M., “The vehicle routing problem with time windows Part I: Tabu search”, ***INFORMS Journal on Computing,*** 8(2): 158-164 (1996a).

43. Potvin, J-Y., Bengio, S., “The vehicle routing problem with time windows Part II: Genetic search”, ***INFORMS Journal on Computing,*** 8(2): 165-172 (1996).

44. Chiang, W.C., Russell, R.A., “Simulated annealing metaheuristics for the vehicle routing problem with time windows”, ***Annals of Operations Research***, 63: 3-27 (1996).

45. Potvin, J.Y., Dubé, D., Robillard, C., “A hybrid approach to vehicle routing using neural networks and genetic algorithms”, ***Applied Intelligence***, 6: 241-252 (1996b).

46. Taillard, E., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.Y., “A tabu search heuristic for the vehicle routing problem with soft time windows”, ***Transportation Science***, 31: 170-186 (1997).

47. Homberger, J., Gehring, H., “Two evolutionary metaheuristics for the vehicle routing problem with time windows”, ***INFOR***, 37(3): 297-317 (1998).

48. Liu, F.H., Shen, S.Y., “A route-neighborhood-based metaheuristic for vehicle routing problem with time windows”, ***European Journal of Operational Research***, 118(3): 485-504 (1999).

49. Schulze, J., Fahle, T., “A parallel algorithm for the vehicle routing problem with time window constraints”, ***Annals of Operations Research***, 86: 585-607 (1999).

50. Gambardella, L.M., Taillard, E., Agazzi, G., "MACS-VRPTW: a multiple ant colony system for vehicle routing problem with time windows", New Ideas in Optimization, Chapter 5, Corne, D., Dorigo, M., Glover, F., ***The McGraw-Hill Companies***, London, 63-76 (1999).

51. İnternet: Braysy, O., "Genetic algorithms for the vehicle routing problem with time windows", http://osiris.tuwien.ac.at/~wgarn/VehicleRouting/Braysy.pdf (2001a).

52. Cordeau, J.F., Laporte, G., Mercier, A., "A unified tabu search metaheuristic for vehicle routing problems with time windows", ***Journal of the Operational Research Society***, 52: 928-936 (2001).

53. Ioannou, G., Kritikos, M., Prastacos, G., "A greedy look-ahead heuristic for the vehicle routing problem with time windows", ***Journal of the Operational Research Society***, 52: 523-537 (2001).

54. İnternet: Czech, Z.J., Czarnas, P., "Parallel simulated annealing for the vehicle routing problem with time windows", http:// csdl.computer.org/ comp/proceedings/ pdp/ 2002/ 1444/00/14440376abs.htm (2002).

55. Gehring, H., Homberger, J., "Parallelization of a two-phase metaheuristic for routing with time windows", ***Journal of Heuristics***, 8(3): 251-276 (2002).

56. Braysy, O., "Local search and variable neighborhood search algorithms for the vehicle routing problem with time windows", Doctoral thesis, ***Department of Mathematics and Statistics, University of Vaasa***, Vaasa, Finland, 84-90, 92-94 (2001b).

57. Braysy, O., "A reactive variable neighborhood search for the vehicle routing problem with time windows", ***INFORMS Journal of Computing,*** 15: 347-368 (2003b).

58. Chaovalitwongse, W., Kim, D., Pardalos, P.M., "GRASP with a new local search scheme for vehicle routing problems with time windows", ***Journal of Combinatoral Optimization,*** 7(2): 179-207 (2003).

59. Li, H., Lim, A., "Local search with annealing-like restarts to solve the vehicle routing problem with time windows", ***European Journal of Operational Research***, 150(1): 115-127 (2003).

60. Lee, L.H., Tan, K.C., Ou,K., Chew, Y.H., "Vehicle capacity planning system: A case study on vehicle routing problem with time windows", ***IEEE Transactions on Systems Man and Cybernetics PartA-Systems and Humans,*** 33(2): 169-178 (2003).

61. Lau, H.C., Sim, M., Teo, K.M., "Vehicle routing problem with time windows and a limited number of vehicles", ***European Journal of Operational Research***, 148(3): 559-569 (2003).

62. Sa'adah, S., Ross, P., Paechter, B., "Improving vehicle routing using a customer waiting time colony", ***Evolutionary Computation in Combinatorial Optimization, Proceedings Lecture Notes in Computer Science***, 3004: 188-198 (2004).

63. Cordeau, J.F., Laporte, G., Mercier, A., "Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows", ***Journal of the Operational Research Society***, 54: 542-546 (2004).

64. Bent, R.W., Van Hentenryck, P., "Scenario-based planning for partially dynamic vehicle routing with stochastic customers", ***Operations Research***, 52(6): 977-987 (2004).

65. Braysy, O., Hasle, G., Dullaert, W., "A multi-start local search algorithm for the vehicle routing problem with time windows", ***European Journal of Operational Research***, 159(3): 586-605 (2004).

66. Homberger, J., Gehring, H., "A two-phase hybrid metaheuristic for the vehicle routing problem with time windows", ***European Journal of Operational Research***, 162(1): 220-238 (2005).

67. Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T., Yagiura, M., "Effective local search algorithms for routing and scheduling problem with general time-window constraints", ***Transportation Science***, 39(2): 206-232 (2005).

68. Karaboğa, D., "Yapay zeka optimizasyon algoritmaları", ***Atlas Yayın Dağıtım***, İstanbul, 113-139 (2004).

69. Alaykıran, K., Engin, O., "Karınca Kolonileri Metasezgiseli ve Gezgin Satıcı Problemleri Üzerinde Bir Uygulaması", ***Gazi Üniv. Müh. Mim. Fak. Der.***, 20: 1, 69-76 (2005).

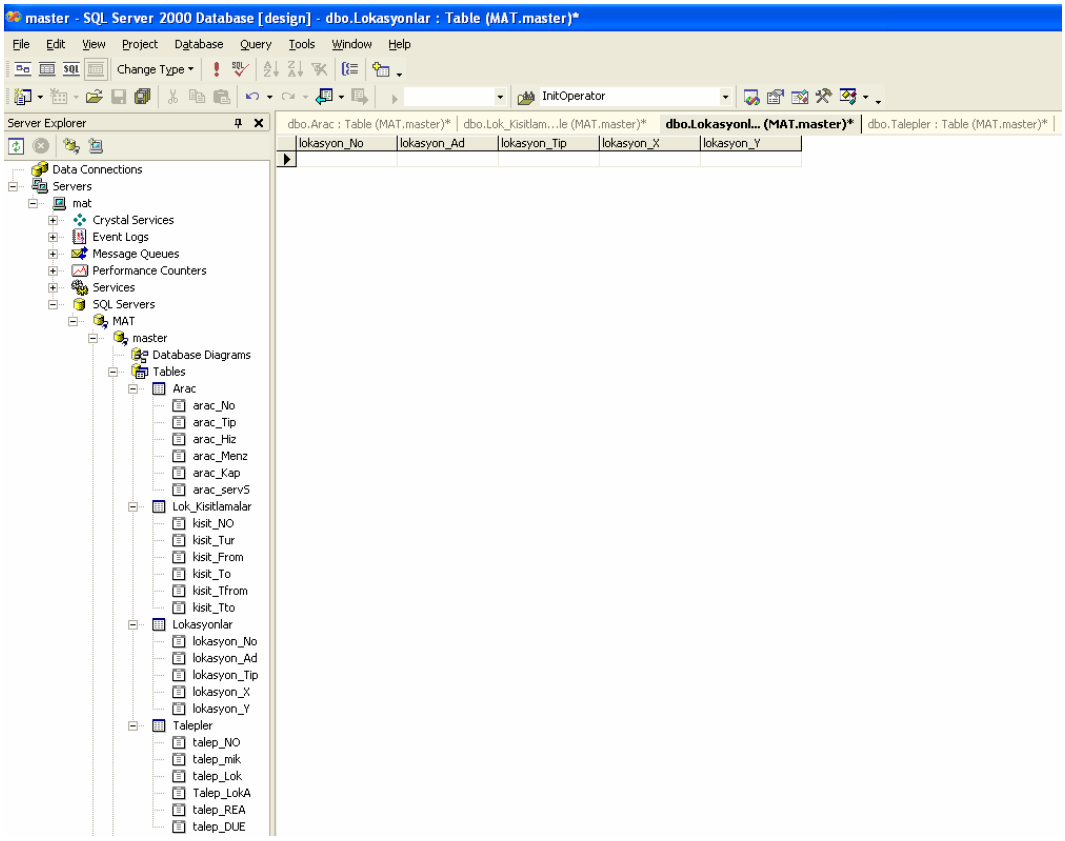
70. Dorigo, M., Stützle, T., "Ant Colony Optimization", ***The MIT Press***, London, England, 39-40 (2004).

71. Keenan, B.K., "Spatial decision support system for vehicle routing", ***Decision Support System***, 22: 65-71 (1998).

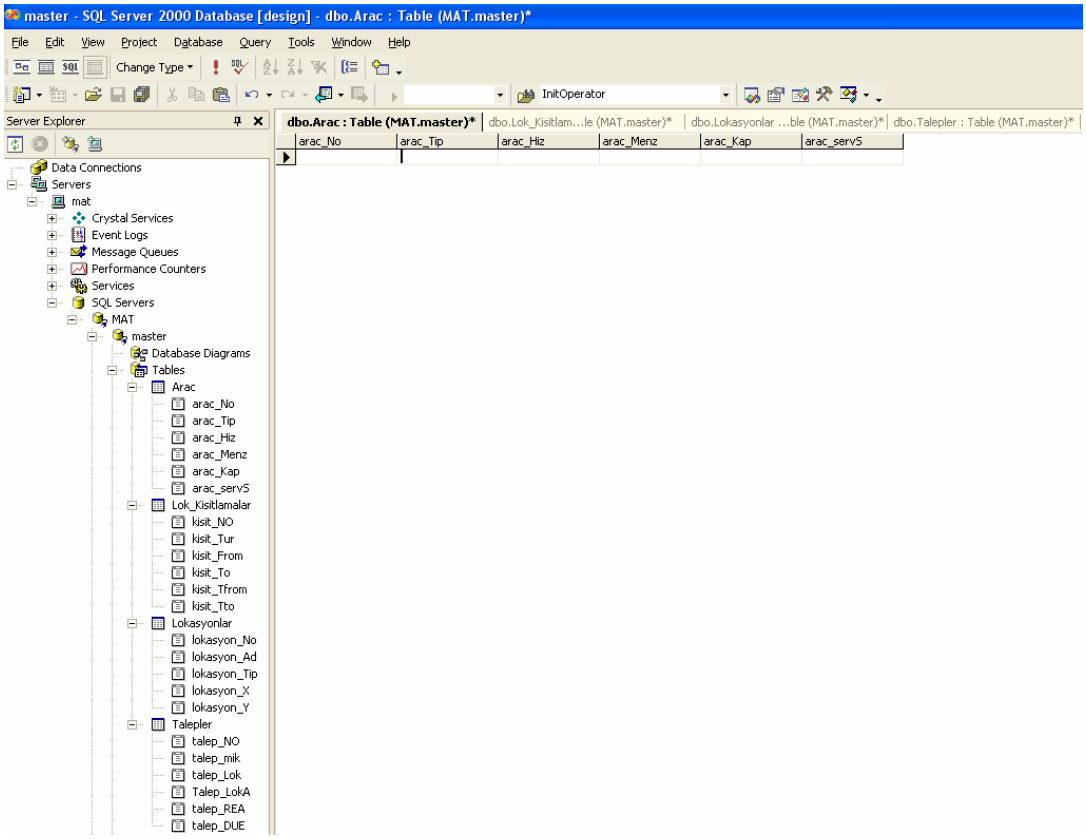
72. Tarantilis, C.D., Diakoulaki, D., Kiranoudis C.T., " Combination of geographical information system and efficient routing algorithms for real life distribution operations" , ***European Journal of Operational Research***, 152: 437–453 (2004).

EKLER

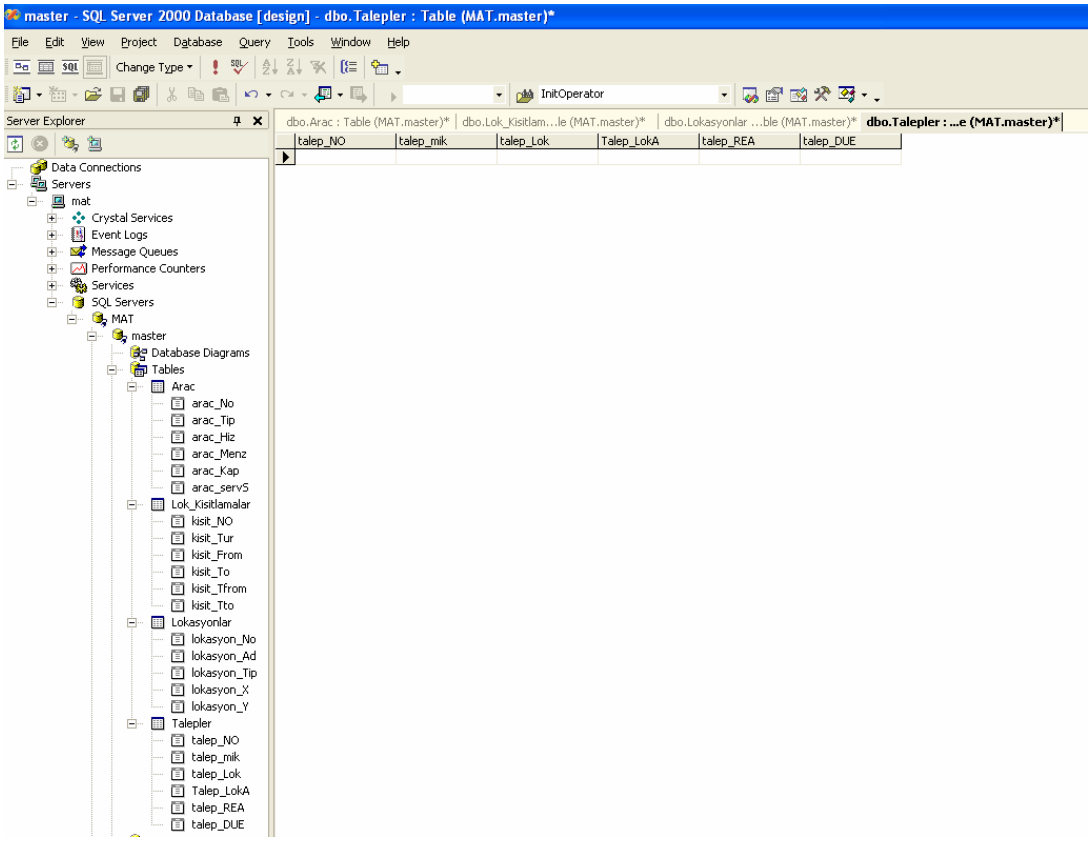
EK-1 LOKASYONLAR VERİ TABANI TABLOSU



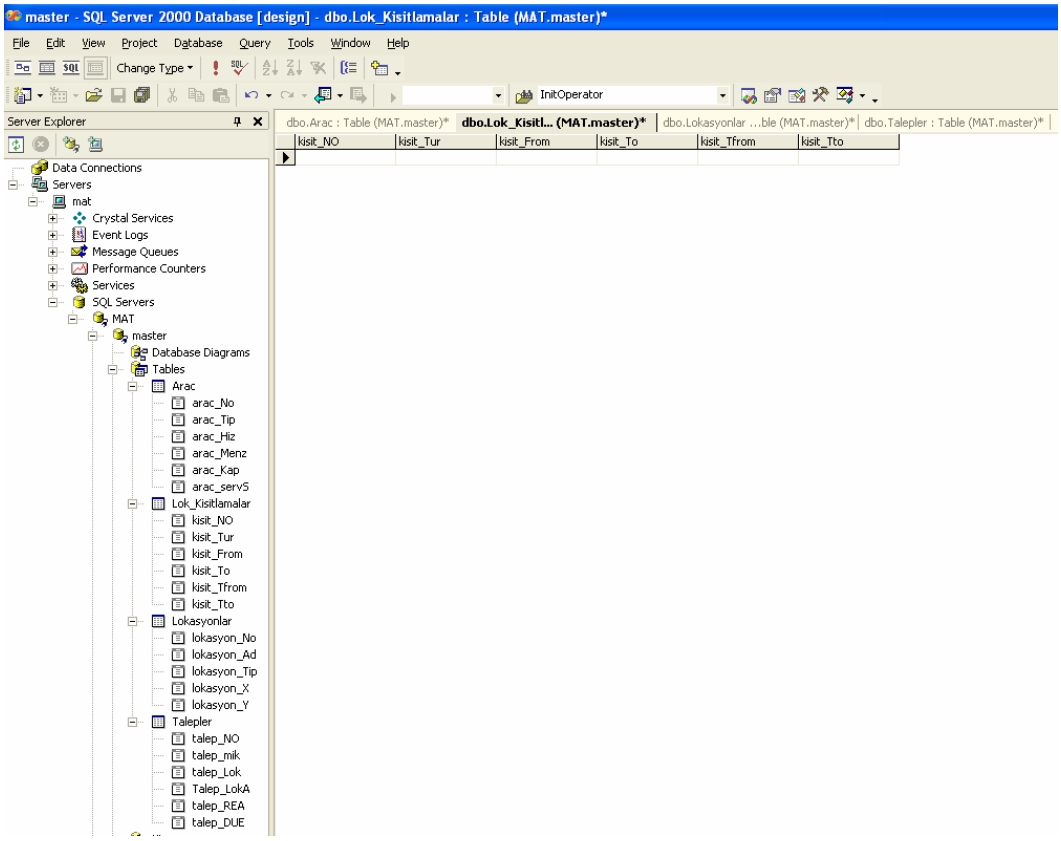
EK-2 ARAÇ VERİ TABANI TABLOSU



EK-3 TALEPLER VERİ TABANI TABLOSU



EK-4 KISITLAMALAR VERİ TABANI TABLOSU



EK-5 C# PROGRAM KODLARI

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
namespace VRP
{public class Form1 : System.Windows.Forms.Form
{private HeuristicLab.VRP.SolomonImporter si;
private HeuristicLab.VRP.VRPSolutionDisplay vrpSolutionDisplay1;
private System.Windows.Forms.Timer timer1;
private System.Windows.Forms.Button btnDur;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.PictureBox pictureBox1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.MainMenu mainMenu1;
private System.Windows.Forms.MenuItem menuItem1;
private System.Windows.Forms.MenuItem menuItem2;
private System.Windows.Forms.MenuItem menuItem3;
private System.Windows.Forms.MenuItem menuItem4;
private System.Windows.Forms.MenuItem menuItem5;
private System.Windows.Forms.MenuItem menuItem6;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.MenuItem menuItem7;
private System.Windows.Forms.MenuItem menuItem8;
private System.Windows.Forms.Button button1;
private System.ComponentModel.IContainer components;
public Form1()
{InitializeComponent();}
private void ProblemAc()
{btnDur.Enabled = false;
si = new HeuristicLab.VRP.SolomonImporter();
si.Text = "Problem Yükle";
si.ShowDialog();
if(si.Problem != null)
{HeuristicLab.VRP.VRPForm vfrm = new HeuristicLab.VRP.VRPForm(si.Problem,
"problem");
for(int i = 0; i< vfrm.Controls.Count; i++)
{if(vfrm.Controls[i].GetType().ToString() == "System.Windows.Forms.DataGrid")
{vfrm.Controls[i].Location = new Point(532, 24);
vfrm.Controls[i].Size = new Size(498, 660);
this.Controls.Add(vfrm.Controls[i]);}}
btnDur.Enabled = true;}}
HeuristicLab.VRP.VRPSolution vrpsol;
private void ProblemCoz()
```

```
{vrpsol = null;
Random rnd = new Random();
vrpsol = new HeuristicLab.VRP.VRPSolution(si.Problem, rnd);
vrpsol.Evaluate();
vrpSolutionDisplay1.Solution = vrpsol;
HeuristicLab.VRP.VRPSolutionDisplay.GraphicsTab gt = new
HeuristicLab.VRP.VRPSolutionDisplay.GraphicsTab();}
protected override void Dispose( bool disposing )
{if( disposing )
{if (components != null)
{components.Dispose();}}
base.Dispose( disposing );}
private void InitializeComponent()
{this.components = new System.ComponentModel.Container();
System.Resources.ResourceManager resources = new
System.Resources.ResourceManager(typeof(Form1));
this.vrpSolutionDisplay1 = new HeuristicLab.VRP.VRPSolutionDisplay();
this.timer1 = new System.Windows.Forms.Timer(this.components);
this.btnDur = new System.Windows.Forms.Button();
this.label1 = new System.Windows.Forms.Label();
this.pictureBox1 = new System.Windows.Forms.PictureBox();
this.label2 = new System.Windows.Forms.Label();
this.mainMenu1 = new System.Windows.Forms.MainMenu();
this.menuItem1 = new System.Windows.Forms.MenuItem();
this.menuItem2 = new System.Windows.Forms.MenuItem();
this.menuItem3 = new System.Windows.Forms.MenuItem();
this.menuItem4 = new System.Windows.Forms.MenuItem();
this.menuItem5 = new System.Windows.Forms.MenuItem();
this.menuItem6 = new System.Windows.Forms.MenuItem();
this.menuItem7 = new System.Windows.Forms.MenuItem();
this.menuItem8 = new System.Windows.Forms.MenuItem();
this.label3 = new System.Windows.Forms.Label();
this.button1 = new System.Windows.Forms.Button();
this.SuspendLayout();
this.vrpSolutionDisplay1.CoarseDisplay = false;
this.vrpSolutionDisplay1.Data = null;
this.vrpSolutionDisplay1.Location = new System.Drawing.Point(0, 8);
this.vrpSolutionDisplay1.Message = "Ready";
this.vrpSolutionDisplay1.Name = "vrpSolutionDisplay1";
this.vrpSolutionDisplay1.Size = new System.Drawing.Size(528, 336);
this.vrpSolutionDisplay1.Solution = null;
this.vrpSolutionDisplay1.TabIndex = 3;
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
this.btnDur.Enabled = false;
this.btnDur.Location = new System.Drawing.Point(8, 400);
this.btnDur.Name = "btnDur";
this.btnDur.Size = new System.Drawing.Size(248, 24);
```

```
this.btnDur.TabIndex = 2;
this.btnDur.Text = "Durdur";
this.btnDur.Click += new System.EventHandler(this.btnDur_Click);
this.label1.Font = new System.Drawing.Font("Verdana", 10F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label1.Location = new System.Drawing.Point(264, 368);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(256, 24);
this.label1.TabIndex = 4;
this.label1.TextAlign = System.Drawing.ContentAlignment.MiddleCenter;
this.label2.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label2.Location = new System.Drawing.Point(344, 352);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(100, 16);
this.label2.TabIndex = 6;
this.label2.Text = "Toplam Mesafe";
this.mainMenu1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[]
{this.menuItem1,
this.menuItem7});
this.menuItem1.Index = 0;
this.menuItem1.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
this.menuItem2, this.menuItem3,  this.menuItem4, this.menuItem5,
this.menuItem6});
this.menuItem1.Text = "Problem";
this.menuItem2.Index = 0;
this.menuItem2.Text = "&Aç";
his.menuItem2.Click += new System.EventHandler(this.menuItem2_Click);
this.menuItem3.Index = 1;
this.menuItem3.Text = "-";
this.menuItem4.Index = 2;
this.menuItem4.Text = "&Çöz";
this.menuItem4.Click += new System.EventHandler(this.menuItem4_Click);
this.menuItem5.Index = 3;
this.menuItem5.Text = "-";
this.menuItem6.Index = 4;
this.menuItem6.Text = "&Çıkış";
this.menuItem6.Click += new System.EventHandler(this.menuItem6_Click);
this.menuItem7.Index = 1;
this.menuItem7.MenuItems.AddRange(new System.Windows.Forms.MenuItem[] {
this.menuItem8});
this.menuItem7.Text = "&Yardım";
this.menuItem8.Index = 0;
this.menuItem8.Text = "Hakkında";
this.menuItem8.Click += new System.EventHandler(this.menuItem8_Click);
```

```
this.label3.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label3.Location = new System.Drawing.Point(528, 8);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(264, 16);
this.label3.TabIndex = 7;
this.label3.Text = "Problem Verileri";
this.button1.FlatStyle = System.Windows.Forms.FlatStyle.Flat;
this.button1.Location = new System.Drawing.Point(8, 368);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(248, 24);
this.button1.TabIndex = 8;
this.button1.Text = "Çöz";
this.button1.Click += new System.EventHandler(this.button1_Click);
this.AutoScaleBaseSize = new System.Drawing.Size(6, 14);
this.ClientSize = new System.Drawing.Size(1016, 705);
this.Controls.Add(this.button1);
this.Controls.Add(this.label3);
this.Controls.Add(this.label2);
this.Controls.Add(this.pictureBox1);
this.Controls.Add(this.label1);
this.Controls.Add(this.vrpSolutionDisplay1);
this.Controls.Add(this.btnDur);
this.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.Menu = this.mainMenu1;
this.Name = "Form1";
this.Text = "ZPARP-KDS";
this.WindowState = System.Windows.Forms.FormWindowState.Maximized;
this.Load += new System.EventHandler(this.Form1_Load);
this.ResumeLayout(false);}
[STAThread]
static void Main()
{Application.Run(new Form1());}
private void timer1_Tick(object sender, System.EventArgs e)
{pictureBox1.Visible = true;
ProblemCoz();
if(vrpSolutionDisplay1.velid() != "valid")
{label1.Text = vrpsol.Route.TravelDistance().ToString();
return;}
else
{timer1.Enabled = false;
pictureBox1.Visible = false;
label1.Text = vrpsol.Route.TravelDistance().ToString();}}
private void btnDur_Click(object sender, System.EventArgs e)
```

```
{timer1.Enabled = false;
pictureBox1.Visible = false;}
private void menuItem6_Click(object sender, System.EventArgs e)
{System.Environment.Exit(0);
this.Close();}
private void menuItem2_Click(object sender, System.EventArgs e)
{if(vrpsol != null)
vrpsol = null;
for(int i = 0; i < this.Controls.Count; i++)
{if(this.Controls[i].GetType().ToString() == "System.Windows.Forms.DataGrid")
{this.Controls.RemoveAt(i);}
this.Refresh();}
ProblemAc();}
private void menuItem4_Click(object sender, System.EventArgs e)
{timer1.Enabled = true;}
private void menuItem8_Click(object sender, System.EventArgs e){
MessageBox.Show(" Bu Bir \n ZAMAN PENCERELİ \n ARAÇ ROTALAMA
PROBLEMİ \n KARAR DESTEK SİSTEMİDİR \n Aydın \t 2005",
"HAKKINDA");}
private void button1_Click(object sender, System.EventArgs e)
{timer1.Enabled = true;}}}

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;
using System.Threading;
using HeuristicLab.Framework;
using HeuristicLab.Framework.GUI;
using HeuristicLab.Framework.Exceptions;
namespace HeuristicLab.VRP
{public class VRPSolutionDisplay : HeuristicLab.Framework.GUI.SolutionDisplay
{public class GraphicsTab : System.Windows.Forms.TabPage {
private Point[] cities;
public bool display = false;
public bool upToDate = false;
public Image image = new Bitmap(10, 10);
public Mutex imageMutex = new Mutex();
private double xFactor, yFactor, xMin, yMin, xMax, yMax;
public int[] currentRoute;
private VRPSolution solution;
public VRPSolution Solution {
get { return solution; }
set { if (value != null) { solution = value; CalculateMinimum();}}}
public GraphicsTab() : base () {}
```

```
protected override void OnPaint(PaintEventArgs e) {
base.OnPaint(e);
if (display && solution != null) {
if (! upToDate) {UpdateFactor();GenerateImage();}
imageMutex.WaitOne(); Graphics graphics = e.Graphics;
graphics.DrawImage(image, 0, 0);
graphics.Dispose(); imageMutex.ReleaseMutex();}}
protected override void OnResize(EventArgs e) {
base.OnResize(e); if ((Width > 0) && (Height > 0)) {
imageMutex.WaitOne();
image.Dispose();image = new Bitmap(Width, Height);
imageMutex.ReleaseMutex();UpdateFactor();UpdateImage();}}
public void UpdateImage() {
upToDate = false;
if ((display) && (this.Visible))
{GenerateImage();
if (this.Handle != IntPtr.Zero)
{this.Invoke(new RepaintDelegate(this.Repaint)); thread}}}
private delegate void RepaintDelegate();
private void Repaint() {
imageMutex.WaitOne();
Graphics graphics = this.CreateGraphics();
graphics.DrawImage(image, 0, 0);
graphics.Dispose();
imageMutex.ReleaseMutex();}
public void CalculateMinimum() { VRP problem = (VRP)Solution.Problem;
if (problem == null) return;
double xMargin, yMargin;
xMin = problem.CityLocations[0, 0];
xMax = problem.CityLocations[0, 0];
yMin = problem.CityLocations[0, 1];
yMax = problem.CityLocations[0, 1];
for (int i = 0; i < problem.CityCount; i++) {
xMin = (problem.CityLocations[i, 0] < xMin ? problem.CityLocations[i, 0] : xMin);
xMax = (problem.CityLocations[i, 0] > xMax ? problem.CityLocations[i, 0] : xMax);
yMin = (problem.CityLocations[i, 1] < yMin ? problem.CityLocations[i, 1] : yMin);
yMax = (problem.CityLocations[i, 1] > yMax ? problem.CityLocations[i, 1] :
yMax);}
xMargin = (xMax - xMin) * 0.1;
yMargin = (yMax - yMin) * 0.1;
xMin -= xMargin;
yMin -= yMargin;
xMax += xMargin;
yMax += yMargin;}
public void UpdateFactor()
{xFactor = this.Width / (xMax - xMin);
yFactor = this.Height / (yMax - yMin);
```

```
if (display && Solution != null && Solution.Problem != null) {
VRP problem = (VRP)Solution.Problem;
cities = new Point[problem.CityCount];
for (int i=0; i<cities.Length; i++) {cities[i] = new
Point((int)((problem.CityLocations[i,0]-xMin) * xFactor),Height-
((int)((problem.CityLocations[i,1]-yMin) * yFactor)));}}}
public void GenerateImage() {
if (solution == null) return;
Point[] points = new Point[solution.Route.Length];
for (int i = 0; i < solution.Route.Length; i++) {points[i] = cities[solution.Route[i]];}
imageMutex.WaitOne();
Graphics graphics = Graphics.FromImage(image);
graphics.Clear(Color.White);
graphics.DrawPolygon(Pens.Black, points);
for (int i = 0; i < cities.Length; i++) {
graphics.FillRectangle(Brushes.Blue, cities[i].X - 1, cities[i].Y - 1, 4, 4);
graphics.DrawString(i.ToString(),  new Font("Arial", 6), new
SolidBrush(Color.Black), cities[i].X+2, cities[i].Y+2, null);}
graphics.DrawRectangle(Pens.Black, 0, 0, Width - 1, Height - 1);
graphics.Dispose();
upToDate = true;
imageMutex.ReleaseMutex();}}
private System.Windows.Forms.TabControl tabControl1;
private GraphicsTab graphic;
private System.Windows.Forms.TabPage text;
private System.Windows.Forms.ContextMenu contextMenu;
private System.Windows.Forms.MenuItem copyMenuItem;
private System.Windows.Forms.MenuItem saveAsMenuItem;
private System.Windows.Forms.SaveFileDialog saveImageDialog;
private System.ComponentModel.IContainer components = null;
private bool display = false;
private bool upToDate = false;
private VRPSolution solution;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox lengthOfRoute;
private System.Windows.Forms.TextBox nrOfVehicles;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.TextBox routesTextBox;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.TextBox optimalLength;
private System.Windows.Forms.TextBox optimalVehicles;
private System.Windows.Forms.Label label6;
private System.Windows.Forms.Label label7;
private System.Windows.Forms.TextBox validRoute;
private System.Windows.Forms.TextBox waitingTime;
```

```
private System.Windows.Forms.Label label8;
private System.Windows.Forms.Label label9;
private System.Windows.Forms.TextBox routeTime;
private int[] currentRoute;
public VRPSolutionDisplay() : base (false)  {
InitializeComponent();
graphic = new GraphicsTab();
tabControl1.Controls.Add (graphic);
graphic.Text = "Rota Çizim";
Message = "Ready";
Settings settings;
try {settings = HeuristicLab.Framework.Settings.Read();} catch (Exception ex)
{GUIUtilities.ShowErrorMessage (this, ex); settings = new Settings();}
saveImageDialog.InitialDirectory = settings.DefaultDocumentsPath;}
protected override void Dispose( bool disposing )  {
if( disposing ){if (components != null) {components.Dispose();}}
base.Dispose( disposing );}
private void InitializeComponent()
{this.tabControl1 = new System.Windows.Forms.TabControl();
this.text = new System.Windows.Forms.TabPage();
this.routeTime = new System.Windows.Forms.TextBox();
this.label9 = new System.Windows.Forms.Label();
this.waitingTime = new System.Windows.Forms.TextBox();
this.label8 = new System.Windows.Forms.Label();
this.validRoute = new System.Windows.Forms.TextBox();
this.label7 = new System.Windows.Forms.Label();
this.label6 = new System.Windows.Forms.Label();
this.optimalVehicles = new System.Windows.Forms.TextBox();
this.optimalLength = new System.Windows.Forms.TextBox();
this.label5 = new System.Windows.Forms.Label();
this.label4 = new System.Windows.Forms.Label();
this.routesTextBox = new System.Windows.Forms.TextBox();
this.label3 = new System.Windows.Forms.Label();
this.nrOfVehicles = new System.Windows.Forms.TextBox();
this.label2 = new System.Windows.Forms.Label();
this.lengthOfRoute = new System.Windows.Forms.TextBox();
this.label1 = new System.Windows.Forms.Label();
this.contextMenu = new System.Windows.Forms.ContextMenu();
this.copyMenuItem = new System.Windows.Forms.MenuItem();
this.saveAsMenuItem = new System.Windows.Forms.MenuItem();
this.saveImageDialog = new System.Windows.Forms.SaveFileDialog();
this.tabControl1.SuspendLayout();
this.text.SuspendLayout();
this.SuspendLayout();
this.tabControl1.Anchor =
((System.Windows.Forms.AnchorStyles)((((System.Windows.Forms.AnchorStyles.T
op |
```

```
System.Windows.Forms.AnchorStyles.Bottom)|System.Windows.Forms.AnchorStyl
es.Left) | System.Windows.Forms.AnchorStyles.Right)));
this.tabControl1.Controls.Add(this.text);
this.tabControl1.Location = new System.Drawing.Point(0, 0);
this.tabControl1.Name = "tabControl1";
this.tabControl1.SelectedIndex = 0;
this.tabControl1.Size = new System.Drawing.Size(536, 352);
this.tabControl1.TabIndex = 1;
this.tabControl1.SelectedIndexChanged += new
System.EventHandler(this.tabControl1_SelectedIndexChanged);
this.text.Controls.Add(this.routeTime);
this.text.Controls.Add(this.label9);
this.text.Controls.Add(this.waitingTime);
this.text.Controls.Add(this.label8);
this.text.Controls.Add(this.validRoute);
this.text.Controls.Add(this.label7);
this.text.Controls.Add(this.label6);
this.text.Controls.Add(this.optimalVehicles);
this.text.Controls.Add(this.optimalLength);
this.text.Controls.Add(this.label5);
this.text.Controls.Add(this.label4);
this.text.Controls.Add(this.routesTextBox);
this.text.Controls.Add(this.label3);
this.text.Controls.Add(this.nrOfVehicles);
this.text.Controls.Add(this.label2);
this.text.Controls.Add(this.lengthOfRoute);
this.text.Controls.Add(this.label1);
this.text.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.text.Location = new System.Drawing.Point(4, 22);
this.text.Name = "text";
this.text.Size = new System.Drawing.Size(528, 326);
this.text.TabIndex = 1;
this.text.Text = "Sonuçlar";
this.text.Click += new System.EventHandler(this.text_Click);
this.routeTime.Anchor = System.Windows.Forms.AnchorStyles.None;
this.routeTime.Location = new System.Drawing.Point(280, 248);
this.routeTime.Name = "routeTime";
this.routeTime.ReadOnly = true;
this.routeTime.Size = new System.Drawing.Size(240, 21);
this.routeTime.TabIndex = 16;
this.routeTime.Text = "-";
this.routeTime.Visible = false;
this.routeTime.TextChanged += new
System.EventHandler(this.routeTime_TextChanged);
this.label9.Anchor = System.Windows.Forms.AnchorStyles.None;
```

```
this.label9.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label9.Location = new System.Drawing.Point(280, 232);
this.label9.Name = "label9";
this.label9.Size = new System.Drawing.Size(176, 16);
this.label9.TabIndex = 15;
this.label9.Text = "Toplam Seyahat Zaman : ";
this.label9.Visible = false;
this.label9.Click += new System.EventHandler(this.label9_Click);
this.waitingTime.Anchor = System.Windows.Forms.AnchorStyles.None;
this.waitingTime.Location = new System.Drawing.Point(280, 152);
this.waitingTime.Name = "waitingTime";
this.waitingTime.ReadOnly = true;
this.waitingTime.Size = new System.Drawing.Size(240, 21);
this.waitingTime.TabIndex = 14;
this.waitingTime.Text = "-";
this.waitingTime.TextChanged += new
System.EventHandler(this.waitingTime_TextChanged);
this.label8.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label8.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label8.Location = new System.Drawing.Point(280, 128);
this.label8.Name = "label8";
this.label8.Size = new System.Drawing.Size(128, 16);
this.label8.TabIndex = 13;
this.label8.Text = "Bekleme Zamanı : ";
this.label8.Click += new System.EventHandler(this.label8_Click);
this.validRoute.Anchor = System.Windows.Forms.AnchorStyles.None;
this.validRoute.Location = new System.Drawing.Point(408, 128);
this.validRoute.Name = "validRoute";
this.validRoute.ReadOnly = true;
this.validRoute.Size = new System.Drawing.Size(112, 21);
this.validRoute.TabIndex = 12;
this.validRoute.Text = "-";
this.validRoute.Visible = false;
this.validRoute.WordWrap = false;
this.validRoute.TextChanged += new
System.EventHandler(this.validRoute_TextChanged);
this.label7.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label7.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label7.Location = new System.Drawing.Point(280, 132);
this.label7.Name = "label7";
this.label7.Size = new System.Drawing.Size(96, 16);
```

```
this.label7.TabIndex = 11;
this.label7.Text = "Geçerlilik : ";
this.label7.Visible = false;
this.label7.Click += new System.EventHandler(this.label7_Click);
this.label6.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label6.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label6.Location = new System.Drawing.Point(272, 104);
this.label6.Name = "label6";
this.label6.Size = new System.Drawing.Size(128, 16);
this.label6.TabIndex = 10;
this.label6.Text = "Rota Bilgileri : ";
this.label6.Click += new System.EventHandler(this.label6_Click);
this.optimalVehicles.Anchor = System.Windows.Forms.AnchorStyles.None;
this.optimalVehicles.Location = new System.Drawing.Point(304, 72);
this.optimalVehicles.Name = "optimalVehicles";
this.optimalVehicles.ReadOnly = true;
this.optimalVehicles.Size = new System.Drawing.Size(192, 21);
this.optimalVehicles.TabIndex = 9;
this.optimalVehicles.Text = "-";
this.optimalVehicles.TextAlign =
System.Windows.Forms.HorizontalAlignment.Right;
this.optimalVehicles.Visible = false;
this.optimalVehicles.TextChanged += new
System.EventHandler(this.optimalVehicles_TextChanged);
this.optimalLength.Anchor = System.Windows.Forms.AnchorStyles.None;
this.optimalLength.Location = new System.Drawing.Point(304, 40);
this.optimalLength.Name = "optimalLength";
this.optimalLength.ReadOnly = true;
this.optimalLength.Size = new System.Drawing.Size(192, 21);
this.optimalLength.TabIndex = 8;
this.optimalLength.Text = "-";
this.optimalLength.TextAlign =
System.Windows.Forms.HorizontalAlignment.Right;
this.optimalLength.Visible = false;
this.optimalLength.TextChanged += new
System.EventHandler(this.optimalLength_TextChanged);
this.label5.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label5.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label5.Location = new System.Drawing.Point(304, 24);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(160, 16);
this.label5.TabIndex = 7;
this.label5.Text = "Bilinen En İyi Sonuç : ";
```

```
this.label5.Visible = false;
this.label5.Click += new System.EventHandler(this.label5_Click);
this.label4.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label4.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label4.Location = new System.Drawing.Point(112, 24);
this.label4.Name = "label4";
this.label4.Size = new System.Drawing.Size(100, 16);
this.label4.TabIndex = 6;
this.label4.Text = "Sonuç : ";
this.label4.Click += new System.EventHandler(this.label4_Click);
this.routesTextBox.Anchor = System.Windows.Forms.AnchorStyles.None;
this.routesTextBox.Location = new System.Drawing.Point(8, 96);
this.routesTextBox.Multiline = true;
this.routesTextBox.Name = "routesTextBox";
this.routesTextBox.ReadOnly = true;
this.routesTextBox.ScrollBars = System.Windows.Forms.ScrollBars.Both;
this.routesTextBox.Size = new System.Drawing.Size(256, 216);
this.routesTextBox.TabIndex = 5;
this.routesTextBox.Text = "";
this.routesTextBox.TextChanged += new
System.EventHandler(this.routesTextBox_TextChanged);
this.label3.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label3.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label3.Location = new System.Drawing.Point(8, 72);
this.label3.Name = "label3";
this.label3.Size = new System.Drawing.Size(104, 16);
this.label3.TabIndex = 4;
this.label3.Text = "Rotalar : ";
this.label3.Click += new System.EventHandler(this.label3_Click);
this.nrOfVehicles.Anchor = System.Windows.Forms.AnchorStyles.None;
this.nrOfVehicles.Location = new System.Drawing.Point(112, 40);
this.nrOfVehicles.Name = "nrOfVehicles";
this.nrOfVehicles.ReadOnly = true;
this.nrOfVehicles.Size = new System.Drawing.Size(152, 21);
this.nrOfVehicles.TabIndex = 3;
this.nrOfVehicles.Text = "-";
this.nrOfVehicles.TextAlign = System.Windows.Forms.HorizontalAlignment.Right;
this.nrOfVehicles.TextChanged += new
System.EventHandler(this.nrOfVehicles_TextChanged);
this.label2.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label2.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
```

```
this.label2.Location = new System.Drawing.Point(8, 40);
this.label2.Name = "label2";
this.label2.Size = new System.Drawing.Size(104, 16);
this.label2.TabIndex = 2;
this.label2.Text = "Araç Sayısı : ";
this.label2.Click += new System.EventHandler(this.label2_Click);
this.lengthOfRoute.Anchor = System.Windows.Forms.AnchorStyles.None;
this.lengthOfRoute.Location = new System.Drawing.Point(112, 40);
this.lengthOfRoute.Name = "lengthOfRoute";
this.lengthOfRoute.ReadOnly = true;
this.lengthOfRoute.Size = new System.Drawing.Size(152, 21);
this.lengthOfRoute.TabIndex = 1;
this.lengthOfRoute.Text = "-";
this.lengthOfRoute.TextAlign =
System.Windows.Forms.HorizontalAlignment.Right;
this.lengthOfRoute.Visible = false;
this.lengthOfRoute.TextChanged += new
System.EventHandler(this.lengthOfRoute_TextChanged);
this.label1.Anchor = System.Windows.Forms.AnchorStyles.None;
this.label1.Font = new System.Drawing.Font("Verdana", 8.25F,
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,
((System.Byte)(162)));
this.label1.Location = new System.Drawing.Point(8, 45);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(104, 16);
this.label1.TabIndex = 0;
this.label1.Text = "Rota Uzunluğu : ";
this.label1.Visible = false;
this.label1.Click += new System.EventHandler(this.label1_Click);
this.contextMenu.MenuItems.AddRange(new System.Windows.Forms.MenuItem[]
{this.copyMenuItem,this.saveAsMenuItem});
this.contextMenu.Popup += new System.EventHandler(this.contextMenu_Popup);
this.copyMenuItem.Index = 0;
this.copyMenuItem.Text = "&Copy";
this.copyMenuItem.Click += new System.EventHandler(this.copyMenuItem_Click);
this.saveAsMenuItem.Enabled = false;
this.saveAsMenuItem.Index = 1;
this.saveAsMenuItem.Text = "&Save as...";
this.saveAsMenuItem.Click += new
System.EventHandler(this.saveAsMenuItem_Click);
this.saveImageDialog.DefaultExt = "gif";
this.saveImageDialog.FileName = "Image";
this.saveImageDialog.Filter = "All image files (*.bmp; *.jpg; *.gif; *.gif;
*.tif)|*.bmp;*.jpg;*.gif;*.tif|Bitma" + "p files (*.bmp)|*.bmp|JPEG files
(*.jpg)|*.jpg|GIF files (*.gif)|*.gif|Tiff file" +"s (*.tif)|*.tif";
this.saveImageDialog.FilterIndex = 4;
this.saveImageDialog.InitialDirectory = ".\\\\";
```

```
this.saveImageDialog.RestoreDirectory = true;
this.saveImageDialog.Title = "Save Image As";
this.saveImageDialog.FileOk += new
System.ComponentModel.CancelEventHandler(this.saveImageDialog_FileOk);
this.Controls.Add(this.tabControl1);
this.Name = "VRPSolutionDisplay";
this.Size = new System.Drawing.Size(536, 352);
this.Controls.SetChildIndex(this.tabControl1, 0);
this.tabControl1.ResumeLayout(false);
this.text.ResumeLayout(false);
this.ResumeLayout(false);}
private void copyMenuItem_Click(object sender, System.EventArgs e)  {
if (display) {if (! upToDate) {graphic.GenerateImage();}
graphic.imageMutex.WaitOne();
Clipboard.SetDataObject(graphic.image, true);
graphic.imageMutex.ReleaseMutex();
}else {Clipboard.SetDataObject(Message);}}
public override void UpdateDisplay() {
sonuc = "invalid";
if ((Solution != null) && (Solution is VRPSolution)) {
if (! Solution.Equals(solution)) { solution = (VRPSolution)Solution;
VRP problem = (VRP)solution.Problem;
graphic.Solution = solution;
currentRoute = solution.Route.ToArray();
graphic.currentRoute = solution.Route.ToArray();
optimalLength.Text = (solution.Problem as
VRP).BestKnownSolutionQuality.ToString();;
optimalVehicles.Text = (solution.Problem as VRP).OptimalNrOfTrucks.ToString();
validRoute.Text = "invalid";
if (! solution.Problem.Equals(problem)) {problem = (VRP)Solution.Problem;
graphic.Solution = (VRPSolution)Solution;
if ((problem.CityLocations != null) && (problem.CityLocations.Length > 0)) {
Message = "";
currentRoute = solution.Route.ToArray();
graphic.currentRoute = solution.Route.ToArray();
lengthOfRoute.Text = solution.Length.ToString();
nrOfVehicles.Text = solution.Vehicles.ToString();
routesTextBox.Text = solution.ToString();
if (solution.Route.Overload() == 0 && solution.Route.Tardiness() == 0) {
validRoute.Text = "valid";
}else {validRoute.Text = "invalid";}
waitingTime.Text = solution.Route.getWaitingTime().ToString();
routeTime.Text = solution.Route.getRouteTime().ToString();
display = true;
saveAsMenuItem.Enabled = true;
graphic.display = true;
graphic.CalculateMinimum();
```

```
graphic.UpdateFactor();
graphic.UpdateImage();}else {
display = false;
graphic.display = false;
saveAsMenuItem.Enabled = false;
Message = "Problem doesn't provide valid city locations!";}
}else {  if (! solution.Route.Equals(currentRoute)) {currentRoute =
solution.Route.ToArray();
graphic.currentRoute = solution.Route.ToArray();
lengthOfRoute.Text = solution.Length.ToString();
nrOfVehicles.Text = solution.Vehicles.ToString();
routesTextBox.Text = solution.ToString();
graphic.Solution = solution;
currentRoute = solution.Route.ToArray();
if (solution.Route.Overload() == 0 && solution.Route.Tardiness() == 0) {
validRoute.Text = "valid";}else {validRoute.Text = "invalid";}
waitingTime.Text = solution.Route.getWaitingTime().ToString();
routeTime.Text = solution.Route.getRouteTime().ToString();
graphic.display = true;
graphic.upToDate = false;
graphic.UpdateImage();}}}}else {if (display && (Solution == null)) {
throw(new InvalidSolutionException("ERROR: Given solution is null"));}
if (Solution != null) { throw(new InvalidSolutionException("ERROR: Given solution
isn't a VRPSolution"));}}
sonuc = validRoute.Text;}
string sonuc = string.Empty;
private void saveAsMenuItem_Click(object sender, System.EventArgs e) {
if (display) {if (saveImageDialog.ShowDialog(this) == DialogResult.OK) {
graphic.imageMutex.WaitOne();
try {if (! upToDate) {graphic.GenerateImage();}
if (saveImageDialog.FileName.EndsWith(".bmp")) {
graphic.image.Save(saveImageDialog.FileName, ImageFormat.Bmp);}
else if (saveImageDialog.FileName.EndsWith(".jpg"))
graphic.image.Save(saveImageDialog.FileName, ImageFormat.Jpeg);
}else if (saveImageDialog.FileName.EndsWith(".gif")) {
graphic.image.Save(saveImageDialog.FileName, ImageFormat.Gif);
}else if (saveImageDialog.FileName.EndsWith(".tif")) {
graphic.image.Save(saveImageDialog.FileName, ImageFormat.Tiff);
}else {
MessageBox.Show(this, "Geçersiz Format",
"Geçersiz Format",
MessageBoxButtons.OK, MessageBoxIcon.Error);}}
catch (Exception ex) {GUIUtilities.ShowErrorMessage(this, ex);}
graphic.imageMutex.ReleaseMutex();}}}
public string velid()
{return sonuc;}}}
```

```
using System;
using System.Collections;
using HeuristicLab.Framework;
using HeuristicLab.Framework.Exceptions;
namespace HeuristicLab.VRP {
[OperatorAttribute("Container")]
public class VRPOperators {
[OperatorAttribute("Initialization")]
public static void RandomInitialization(Problem problem, Solution solution, Random
random) {ArrayList remainingCities;
VRP vrp = (problem as VRP);
VRPSolution vrpSolution = (solution as VRPSolution);
int index; if (random == null) {random = new Random();}
remainingCities = new ArrayList(vrp.CityCount+vrp.NrOfTrucks-2);
for (int i = 1; i < vrp.CityCount; i++) {remainingCities.Add(i);}
for (int i=0; i < vrp.NrOfTrucks-1; i++) {remainingCities.Add(0);}
vrpSolution.Route.Add(0); while (remainingCities.Count > 0)  {
index = random.Next(remainingCities.Count);
vrpSolution.Route.Add((int)remainingCities[index]);
remainingCities.RemoveAt(index);}
vrpSolution.Route.Add(0); vrpSolution.Route.cleanRoute();}
[OperatorAttribute("Initialization")]
public static void PushForwardInsertionHeuristicInitialization(Problem problem,
Solution solution, Random random) {VRP vrp = (problem as VRP);
VRPSolution sol = (solution as VRPSolution);
int indexOfMinimumCost = -1; int indexOfCustomer = -1;
double cost = 0; double minimumCost = double.MaxValue; ArrayList unroutedList =
new ArrayList(); ArrayList costList = new ArrayList(); int index;
if (vrp == null) {throw(new InvalidSolutionException("Given problem is not a
VRPProblem."));} if (sol == null) {throw(new InvalidProblemException("Given
solution is not a VRPSolution."));} if (random == null) {random = new
Random();}for (int i = 1; i < vrp.CityCount; i ++) {
index = 0; distance = vrp.DistanceOperator(vrp.CityLocations[i,0],
vrp.CityLocations[i,1], x0, y0); if (vrp.CityLocations[i,0] < x0) distance = -distance;
if (costList.Count == 0) {costList.Add(cost); unroutedList.Add(i);
minimumCost = cost; indexOfMinimumCost = i;continue;}else while (index <
costList.Count && (double)costList[index] < cost) index++;costList.Insert(index,
cost);unroutedList.Insert(index, i);
if (cost < minimumCost) minimumCost = cost; indexOfMinimumCost = i;}}
int routeIndex = 0; int currentRoute = 0; int c; int customer = -1;
_Route route = new _Route(); route.VRPData = vrp; minimumCost =
double.MaxValue; indexOfMinimumCost = -1; route.Add (0); route.Add (0);
route.InsertAt(1, (int)unroutedList[0]); unroutedList.RemoveAt(0);
currentRoute = routeIndex; routeIndex++; do {for (c = 0; c < unroutedList.Count;
c++) {for (int i = currentRoute+1; i < route.Count; i++)
{route.InsertAt(i,(int)unroutedList[c]); if (route[currentRoute] != 0) { throw new
Exception("currentRoute not deport."); } cost = route.TravelDistance(currentRoute);
```

```
if (cost < minimumCost && route.SubrouteTardinessOK(currentRoute) &&
route.SubrouteLoadOK(currentRoute)) {minimumCost = cost;indexOfMinimumCost
= i;customer = (int)unroutedList[c];indexOfCustomer = c;}
route.RemoveAt(i);}}
if (indexOfMinimumCost != -1) {
route.InsertAt(indexOfMinimumCost,customer);
routeIndex++; unroutedList.RemoveAt(indexOfCustomer);
costList.RemoveAt(indexOfCustomer);}else { routeIndex++;
route.InsertAt(routeIndex, 0); currentRoute = routeIndex;
route.InsertAt(route.Length-1, (int)unroutedList[0]);
unroutedList.RemoveAt(0);routeIndex++;}
minimumCost = double.MaxValue;
indexOfMinimumCost = -1; indexOfCustomer = -1; customer = -1;
}while (unroutedList.Count > 0);
sol.Route = route;
sol.Evaluate();}
#region Distance Operators (Distance)
#region EuclideanRounded
[OperatorAttribute("Distance")]
public static double EuclideanRounded(double x1, double y1, double x2, double y2)
{ double xDistance, yDistance;
double length;
xDistance = x1 - x2;
yDistance = y1 - y2;
length = Math.Sqrt((xDistance * xDistance) + (yDistance * yDistance));
return(Math.Round(length));}
#region Euclidean
[OperatorAttribute("Distance")]
public static double Euclidean(double x1, double y1, double x2, double y2) {
double xDistance, yDistance;
double length;
xDistance = x1 - x2;
yDistance = y1 - y2;
length = Math.Sqrt((xDistance * xDistance) + (yDistance * yDistance));
return(length);}
[OperatorAttribute("Similarity")]
public static double Hamming(Solution sol1, Solution sol2)
{VRPSolution solution1 = (sol1 as VRPSolution);
VRPSolution solution2 = (sol2 as VRPSolution);
int[] oldRoute1, oldRoute2;
int[,] edgeList;
int length, index, currentEdge, similarityCount;
if ((solution1 == null) || (solution2 == null))
{throw(new InvalidSolutionException("ERROR: Given parent isn't a
VRPSolution"));}
if (solution1.Route.Length != solution2.Route.Length){
```

```
throw(new InvalidSolutionException("ERROR: Given routes have a different
length"));}
oldRoute1 = solution1.Route.ToArray();
oldRoute2 = solution2.Route.ToArray();
length = oldRoute1.Length;
edgeList = new int[length, 4];
for (int i = 0; i < length; i++)
{ index = 0;
while ((index < length) && (oldRoute1[index] != i)) { index++;}
edgeList[i, 0] = oldRoute1[(index - 1 + length) % length];
edgeList[i, 1] = oldRoute1[(index + 1) % length];
index = 0; while ((index < length) && (oldRoute2[index] != i))
{ index++;} currentEdge = oldRoute2[(index - 1 + length) % length];
if ((edgeList[i, 0] != currentEdge) && (edgeList[i, 1] != currentEdge))
{edgeList[i, 2] = currentEdge; }
else {edgeList[i, 2] = -1;}
currentEdge = oldRoute2[(index + 1) % length];
if ((edgeList[i, 0] != currentEdge) && (edgeList[i, 1] != currentEdge))
{  edgeList[i, 3] = currentEdge;}
else {edgeList[i, 3] = -1;}}
similarityCount = 0;
for (int i = 0; i < length; i++)
{for (int j = 2; j < 4; j++)
{if (edgeList[i, j] == -1)
{similarityCount++;}}}
return((double)(((2 * length) - similarityCount) / 2));}

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Windows.Forms;
using System.IO;
using System.Globalization;
using System.Text.RegularExpressions;
using HeuristicLab.Framework;
using HeuristicLab.Framework.Exceptions;
using HeuristicLab.Framework.GUI;
namespace HeuristicLab.VRP {
public class SolomonImporter : HeuristicLab.Framework.GUI.ImportProblemDialog
{private System.Windows.Forms.OpenFileDialog openFileDialog;
private System.Windows.Forms.TextBox fileTextBox;
private System.Windows.Forms.Label fileLabel;
private System.ComponentModel.IContainer components = null;
private System.Windows.Forms.Button okButton;
private System.Windows.Forms.Button cancelButton;
private System.Windows.Forms.Button chooseFileButton;
```

```
private System.Windows.Forms.GroupBox groupBox1;
private System.Windows.Forms.Panel panel1;
private System.Windows.Forms.CheckBox[] penaltyBoxes;
private System.Windows.Forms.NumericUpDown[] penaltyWeights;
private System.Windows.Forms.RadioButton[] initBoxes;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
Delegate[] penaltyOperators;
private System.Windows.Forms.CheckBox onlyFeasibleSolutionsCheckBox;
private System.Windows.Forms.TextBox NrOfVehicles;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.TextBox bestKnownQualityTextBox;
private System.Windows.Forms.Label bestKnownQualityLabel;
private System.Windows.Forms.CheckBox bestKnownQualityAvailableCheckBox;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Label label4;
Delegate[] initOperators;
public SolomonImporter() {
int nrOfOperators;
InitializeComponent();
openFileDialog.InitialDirectory = Settings.DefaultImportableProblemsPath;
initOperators =
HeuristicLab.Framework.FrameworkUtilities.ExtractOperators("VRP",
"Initialization", typeof(InitializationOperator));
nrOfOperators = initOperators.Length;
initBoxes = new RadioButton[nrOfOperators];
for (int i=0; i < nrOfOperators; i++) {
initBoxes[i] = new RadioButton();
initBoxes[i].Width = 300;
initBoxes[i].Location = new System.Drawing.Point (16, 46+i*30);
panel1.Controls.Add (initBoxes[i]);}
try{initBoxes[0].Text = "En Yakın Komşuluk  Algoritması";
initBoxes[1].Text = "INSERT1";
initBoxes[1].Checked = true;}
catch(Exception){}
protected override void Dispose( bool disposing ) {
if( disposing ) {
if (components != null) {components.Dispose();}}
base.Dispose( disposing );}
private void import() {
VRP vrp = null;
StreamReader reader = new StreamReader(fileTextBox.Text,
System.Text.Encoding.ASCII);
vrp = SolomonFileImporter (reader);
reader.Close();
if (vrp == null) {this.DialogResult = DialogResult.Cancel;
return;}
```

```
if (bestKnownQualityAvailableCheckBox.Checked) {
vrp.BestKnownSolutionQuality = double.Parse(bestKnownQualityTextBox.Text);
vrp.OptimalNrOfTrucks = int.Parse(NrOfVehicles.Text);
} else {vrp.BestKnownSolutionQuality = double.MaxValue;
vrp.OptimalNrOfTrucks = int.MaxValue;}
vrp.OnlyFeasibleSolutions = onlyFeasibleSolutionsCheckBox.Checked;
Problem = vrp;
vrp.DistanceOperator = new DistanceOperator(VRPOperators.Euclidean);
int i = 0;
for (int k = 0; k < penaltyOperators.Length; k++) {
if (penaltyBoxes[k].Checked == true) {i++;}}
PenaltyOperator[] myPO = new PenaltyOperator[i];
vrp.PenaltyWeights = new double[i];
for (int k=0, j=0; k < penaltyOperators.Length; k++) {
if (penaltyBoxes[k].Checked == true) {
myPO[j] = (PenaltyOperator)(penaltyOperators[k]);
vrp.PenaltyWeights[j] = (double)penaltyWeights[k].Value;
j++;}}
vrp.PenaltyOperator = myPO;
for (int k=0; k < initBoxes.Length; k++) {
if (initBoxes[k].Checked == true) {
vrp.InitOperator = (InitializationOperator)(initOperators[k]);
break;}}
this.DialogResult = DialogResult.OK;}
private VRP SolomonFileImporter (StreamReader reader)
{VRP vrp = new VRP();
char[] whiteSpaces = {' ', '\t'};
string line;
int nr = 0;
Regex reg = new Regex(@"\d+");
MatchCollection m;
if (reader == null) { return null; }
vrp.ProblemName = reader.ReadLine();
for (int i=0; i<3; i++) {reader.ReadLine();}
line = reader.ReadLine();
m = reg.Matches(line);
if (m.Count != 2) {
MessageBox.Show (this, "Seçilen Dosya Solomon Değil!", "Geçersiz Format",
MessageBoxButtons.OK, MessageBoxIcon.Error);
this.DialogResult = DialogResult.Cancel;
return null;}
vrp.NrOfTrucks = int.Parse(m[0].Value);
vrp.Capacity = int.Parse(m[1].Value);
for (int i=0; i<4; i++) {reader.ReadLine();}
ArrayList coord = new ArrayList();
ArrayList demand = new ArrayList();
ArrayList readytime = new ArrayList();
```

```
ArrayList duedate = new ArrayList();
ArrayList servicetime = new ArrayList();
int[,] xy = new int[1,2];
line = reader.ReadLine();
while ((line != null) && (line.Length > 5))
{m = reg.Matches(line);
if (m.Count != 7) {continue;}
xy = new int[1,2];
xy[0,0] = int.Parse(m[1].Value);
xy[0,1] = int.Parse(m[2].Value);
coord.Add (xy);
demand.Add (int.Parse(m[3].Value));
readytime.Add (int.Parse(m[4].Value));
duedate.Add (int.Parse(m[5].Value));
servicetime.Add (int.Parse(m[6].Value));
line = reader.ReadLine();}
vrp.CityCount = nr = servicetime.Count;
vrp.CityLocations = new double[nr, 2];
for (int i = 0; i < nr; i++)
{vrp.CityLocations[i,0] = ((int[,])coord[i])[0,0];
vrp.CityLocations[i,1] = ((int[,])coord[i])[0,1];}
vrp.Demand = new int[nr];
Array.Copy(demand.ToArray(),0,vrp.Demand,0,nr);
vrp.ReadyTime = new int[nr];
Array.Copy(readytime.ToArray(),0,vrp.ReadyTime,0,nr);
vrp.DueDate = new int[nr];
Array.Copy(duedate.ToArray(),0,vrp.DueDate,0,nr);
vrp.ServiceTime = new int[nr];
Array.Copy (servicetime.ToArray(),0,vrp.ServiceTime,0,nr);
return vrp;}
private void chooseFileButton_Click(object sender, System.EventArgs e) {
if (openFileDialog.ShowDialog(this) == DialogResult.OK) {
fileTextBox.Text = openFileDialog.FileName;
fileTextBox.Focus();
okButton.Enabled = true;}}
private void bestKnownQualityAvailableCheckBox_CheckedChanged(object sender,
System.EventArgs e) {if (bestKnownQualityAvailableCheckBox.Checked) {
bestKnownQualityTextBox.Text = "0";
NrOfVehicles.Text = "0";
} else {bestKnownQualityTextBox.Text = "MAX. Uzunluk";
NrOfVehicles.Text = "-";}
bestKnownQualityTextBox.Enabled =
bestKnownQualityAvailableCheckBox.Checked;
NrOfVehicles.Enabled = bestKnownQualityAvailableCheckBox.Checked;}
private void okButton_Click(object sender, System.EventArgs e) {
this.Cursor = Cursors.WaitCursor;
import();
```

```
this.Cursor = Cursors.Default;}
private void cancelButton_Click(object sender, System.EventArgs e) {
Problem = null;
this.DialogResult = DialogResult.Cancel;}
private void bestKnownQualityTextBox_Validating(object sender,
System.ComponentModel.CancelEventArgs e) {
try {Double.Parse(bestKnownQualityTextBox.Text);
} catch (Exception) {e.Cancel = true;
MessageBox.Show(this, "Geçersiz Değer!", "Geçersiz Değer",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
bestKnownQualityTextBox.SelectAll();}}
private void button1_Click(object sender, System.EventArgs e)
{if (openFileDialog.ShowDialog(this) == DialogResult.OK)
{fileTextBox.Text = openFileDialog.FileName;
fileTextBox.Focus();
okButton.Enabled = true;}}
private void onlyFeasibleSolutionsCheckBox_CheckedChanged(object sender,
System.EventArgs e)  {
VRP vrp = Problem as VRP;
if (vrp != null) {vrp.OnlyFeasibleSolutions =
onlyFeasibleSolutionsCheckBox.Checked;}}
private void openFileDialog_FileOk(object sender,
System.ComponentModel.CancelEventArgs e){}}
using System;
using System.Collections;
using HeuristicLab.Framework;
namespace HeuristicLab.VRP
{[Serializable]
public class _Route : CollectionBase {
private int [] ReadyTime, ServiceTime, DueDate, Demand;
private double [,] CityLocations;
private int CityCount;
private int Capacity;
private DistanceOperator myDistanceOperator;
private bool myOnlyFeasibleSolutions;
public VRP VRPData {
set {if (value != null) {
ReadyTime = ((VRP)value).ReadyTime;
ServiceTime = ((VRP)value).ServiceTime;
DueDate = ((VRP)value).DueDate;
Demand = ((VRP)value).Demand;
CityLocations = ((VRP)value).CityLocations;
myDistanceOperator = ((VRP)value).DistanceOperator;
CityCount = ((VRP)value).CityCount;
Capacity = ((VRP)value).Capacity;
myOnlyFeasibleSolutions = ((VRP)value).OnlyFeasibleSolutions;}}}
public bool OnlyFeasibleSolutions {
```

```
get { return myOnlyFeasibleSolutions; }}
public _Route(ArrayList al) : base() {
foreach( int i in al) {List.Add (i);}}
public _Route(int[] al) : base() {
foreach (int i in al) {
List.Add(i);}}
public _Route() : base() {}
public override string ToString() {
System.Text.StringBuilder temp = new System.Text.StringBuilder();
int load = 0;
int i=1;
while (i < List.Count) {
if (this[i] != 0) {
temp.Append(this[i].ToString() + " ");
load+=Demand[this[i]];
i++;
}else {temp.Append("// " +load.ToString() + System.Environment.NewLine);
load = 0;
while (i < List.Count && this[i] == 0) i++;}}
return temp.ToString();}
public override bool Equals(object o)
{_Route r;if (!(o is _Route)) { return false; }
r = o as _Route;
if (Length != r.Length) { return false; }
for (int i=0; i < Length; i++){
if (this[i] != r[i]) return false;}
return true;}
public override int GetHashCode()
{return List.GetHashCode ();}
public void Add (int customer)
{List.Add(customer);}
public void Remove (int customer) {
List.Remove(customer);}

public new void RemoveAt (int index) {
if (index >= Count || index < 0) {
throw(new Exception("Index out of range: " +index.ToString() + " \n" +
List.Count.ToString()));}
List.RemoveAt (index);}
public _Route RemoveSubroute (int begin, int end) {
_Route r = new _Route();
for (int i=begin; i < end; i++) {
r.Add(this[begin]);
this.RemoveAt(begin);}
return r;}
public void InsertAt (int index, int val) {
if (index < 0 || index > Count) {
```

```
throw(new Exception("Index out of range."));
}List.Insert(index, val);}
public void InsertAt (int index, int[] route) {
if (index < 0 || index > Count) {,
throw(new Exception("Index out of range."));
}  for (int i=0; i < route.Length; i++) {
List.Insert(index+i, route[i]);}}
public void ReplaceRoute(int[] newRoute) {
List.Clear();
foreach(int i in newRoute) List.Add(i);}
public void ReplaceRoute(_Route route) {
foreach(int i in route) List.Add(i);}
public int[] SubRoute (int begin, int end) {
int[] t = new int[end-begin];
for (int i=0; i < t.Length; i++) {
t[i] = (int)List[begin+i];}
return t;}
public virtual int this[int index] {get {
return (int) this.List[index]; }
set {if (value != 0) {
this.List[index] = value;}}}
public int Length {get { return this.List.Count; }}
public void cleanRoute() {int i=0;
while (i < List.Count-1) {if (this[i] == 0 && this[i+1] == 0) {
this.RemoveAt(i);
continue;}i++;}}
public int EndOfSubRoute(int index) {
if (index >= List.Count) return List.Count-1;
else if (index <= 0) return 0; index++;
while (index < List.Count && this[index] != 0) index++;
return index;}
public int BeginOfSubRoute(int index) {
if (index >= List.Count) return List.Count-1;
else if (index <= 0) return 0;
else if (this[index]==0) return index; index--;
while (index > 0 && this[index] != 0) index--;
return index;}
public int[] ToArray() {
int[] n = new int[List.Count];
for (int i=0; i < List.Count; i++) {
n[i] = this[i];}
return n;}
public double TravelDistance() {
double distance = 0;
for (int i=0; i < Length-1; i++) {
distance += myDistanceOperator (CityLocations[this[i], 0], CityLocations[this[i], 1],
CityLocations[this[i+1], 0], CityLocations[this[i+1], 1]);}
```

```
return distance;}
public double TravelDistance(int begin) {
double distance = 0;
for (int i=begin; i < Length-1 && (i==begin || this[i]!=0); i++) {
distance+= myDistanceOperator (CityLocations[this[i], 0], CityLocations[this[i], 1],
CityLocations[this[i+1], 0], CityLocations[this[i+1], 1]);}
return distance;}
public int NrOfVehicles() {int nr = 0; int i=1;
while (i < List.Count) {if (this[i]==0) {nr++;
while (i < List.Count && this[i]==0) { i++; continue; }}i++;} return nr;}
public int Overload() {int o = 0;
int t = 0; for (int i=0; i < List.Count; i++) {
if (this[i] == 0) {if (t > Capacity) {o += (t-Capacity);}
t = 0;}t += Demand[this[i]];}
return o;}
public double Tardiness() {
double t = 0; double total = 0; for (int i=1; i < List.Count; i++) {
t+=myDistanceOperator(CityLocations[this[i-1], 0], CityLocations[this[i-1], 1],
CityLocations[this[i], 0], CityLocations[this[i], 1]);
if (this[i] == 0) {if (t > DueDate[0]) {total += (t - DueDate[0]);}  t = 0;}
else if (t < ReadyTime[this[i]]) { t = ReadyTime[this[i]] + ServiceTime[this[i]]; }
else if (t < (DueDate[this[i]])) { t += ServiceTime[this[i]]; }else total += (t-
(DueDate[this[i]]));}return total;}
public bool SubrouteTardinessOK(int begin) {
double t = 0;int i; for (i=begin+1; i < List.Count && this[i]!=0; i++) {
t+=myDistanceOperator(CityLocations[this[i-1], 0], CityLocations[this[i-1],
1],CityLocations[this[i], 0], CityLocations[this[i], 1]);
if (t < ReadyTime[this[i]]) { t = ReadyTime[this[i]] + ServiceTime[this[i]]; }
else if (t < (DueDate[this[i]] /*- vrp.ServiceTime[this[i]]*/)) { t +=
ServiceTime[this[i]]; }
else { return false; }}
t+=myDistanceOperator(CityLocations[this[i-1], 0], CityLocations[this[i-1], 1],
CityLocations[this[i], 0], CityLocations[this[i], 1]);
if (t > DueDate[this[i]]) { return false; }return true;}
public bool SubrouteLoadOK(int begin) {int t = 0; for (int i=begin+1; i < Length &&
this[i] != 0; i++) {t += Demand[this[i]];}return (t <= Capacity);}
public void Repair(int newRouteIndex) {
int[] cities = new int[CityCount];int i;
int endIndex = EndOfSubRoute(newRouteIndex);
int beginIndex = BeginOfSubRoute(newRouteIndex);
if (endIndex > List.Count || beginIndex < 0) {
throw (new IndexOutOfRangeException("ERROR: Index of Subroute out of
range."));}
i = beginIndex;
while (i < endIndex) {if (cities[this[i]] == 0) { cities[this[i]]++;}
else { this.RemoveAt(i); endIndex--; continue; }i++;} i=0;
```

```
while (i < List.Count-1) {if (i >= beginIndex && i < endIndex) { i++; continue; } if
(cities[this[i]] == 0) { cities[this[i]]++;
}else if (this[i] != 0) { this.RemoveAt(i); beginIndex--; endIndex--;
continue;}i++;}
double bcost = int.MaxValue, mcost = int.MaxValue, acost;
int index = -1, customer = -1;
int current = 0; for (i=0; i < cities.Length; i++) {
if (cities[i] == 0) { customer = i;
current = 0; bcost = TravelDistance(current);
for (int j=1; j < List.Count-1; j++) {
List.Insert(j, i); acost = TravelDistance(current);
if (acost-bcost < mcost && SubrouteLoadOK(current) &&
SubrouteTardinessOK(current)) {
index = j; mcost = acost-bcost;}
List.RemoveAt(j);
if (this[j] == 0) { current = j; bcost = TravelDistance(current); }}
if (index != -1) {List.Insert (index, customer);
} else { List.Add (customer);
List.Add (0);}
mcost = int.MaxValue;
index = -1;}} cleanRoute();}
public int Select (Random random) {ArrayList routes = new ArrayList();
ArrayList probabilities = new ArrayList(); int count = 0; int sum = 0;
double probsum = 0;
for (int i=1; i < this.Length; i++) {count++; sum++;
if (this[i] == 0) {routes.Add (count);count = 0;}}
double temp;
foreach ( int t in routes) {
temp = (double)1/((double)t/(double)sum);
probabilities.Add(temp);
probsum += temp;}
int rand = random.Next ((int)probsum);
int sum1 = 0; int lower = 0;  int upper = 0;  int v = 0; int r = 0;
foreach (double d in probabilities) {
upper += (int)routes[v];
if (rand <= sum1) r = random.Next(lower, upper);
lower += (int)routes[v]; v++;}return r;}
public bool validRoute() {int[] cities = new int[CityCount];
for (int i=0; i < Length; i++) {cities[this[i]]++;}
for (int i=1; i < cities.Length; i++) {if (cities[i] == 0) return false;
if (this[cities[i]]!= 0 && cities[i] > 1) return false;}return true;}
public double getWaitingTime() {double waitingTime = 0; double arrivalTime = 0;
for (int i=1; i < List.Count; i++) {
arrivalTime += myDistanceOperator(CityLocations[this[i-1],0], CityLocations[this[i-
1],1], CityLocations[this[i],0], CityLocations[this[i],1]);
waitingTime += Math.Max(0, ReadyTime[this[i]]-arrivalTime);
if (arrivalTime < ReadyTime[this[i]]) arrivalTime = ReadyTime[this[i]];
```

```
arrivalTime += ServiceTime[this[i]];
if (this[i] == 0) arrivalTime = 0;}return waitingTime;}
public double getRouteTime() {
double travelTime = 0;
double sum = 0;
for (int i=1; i<List.Count; i++) {travelTime +=
myDistanceOperator(CityLocations[this[i-1],0], CityLocations[this[i-
1],1],CityLocations[this[i],0], CityLocations[this[i],1]);
if (travelTime < ReadyTime[this[i]]) travelTime = ReadyTime[this[i]];
travelTime += ServiceTime[this[i]];
if (this[i]==0) {sum += travelTime; travelTime = 0;}}
return sum;}}}
```

ÖZGEÇMİŞ

Mehmet Aydın TOKAYLI, 1978 yılında Bolu'da doğdu. Ankara Anadolu Lisesi (Almaca) bitirdi. 1996 yılında Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Endüstri Mühendisliği Bölümünü kazandı. 2001 yılında mezun olduktan sonra aynı Gazi Üniversitesi Fen Bilimleri Enstitüsü'nde yüksek lisans eğitimine başladı.