EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(DOKTORA TEZİ)

ROTALAMA PROBLEMLERİ İÇİN ALGORİTMİK YAKLAŞIMLAR

Onur UĞURLU

Tez Danışmanı: Prof. Dr. Urfat NURİYEV

İkinci Danışmanı: Doç. Dr. Murat Erşen BERBERLER

Matematik Anabilim Dalı

Sunuş Tarihi: 31.07.2018

Bornova - İZMİR

2018

Onur UĞURLU tarafından doktora tezi olarak sunulan "**Rotalama Problemleri için Algoritmik Yaklaşımlar**" başlıklı bu çalışma E.Ü. Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü. Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 31.07.2018 tarihinde yapılan tez savunma sınavında aday oybirliği/~~oyçokluğu ile~~ başarılı bulunmuştur.

Jüri Üyeleri:		**İmza**
Jüri Başkanı	**: Prof. Dr. Urfat NURİYEV**	.....
Raportör Üye	**: Dr. Öğr. Üyesi Arif GÜRSOY**	.....
Üye	**: Dr. Öğr. Üyesi Zeynep Nihan BERBERLER**	.....
Üye	**: Doç. Dr. Burak ORDİN**	.....
Üye	**: Doç. Dr. Ersin ASLAN**	.....

EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

ETİK KURALLARA UYGUNLUK BEYANI

EÜ Lisansüstü Eğitim ve Öğretim Yönetmeliğinin ilgili hükümleri uyarınca Doktora Tezi olarak sunduğum "**Rotalama Problemleri için Algoritmik Yaklaşımlar**" başlıklı bu tezin kendi çalışmam olduğunu, sunduğum tüm sonuç, doküman, bilgi ve belgeleri bizzat ve bu tez çalışması kapsamında elde ettiğimi, bu tez çalışmasıyla elde edilmeyen bütün bilgi ve yorumlara atıf yaptığımı ve bunları kaynaklar listesinde usulüne uygun olarak verdiğimi, tez çalışması ve yazımı sırasında patent ve telif haklarını ihlal edici bir davranışımın olmadığını, bu tezin herhangi bir bölümünü bu üniversite veya diğer bir üniversitede başka bir tez çalışması içinde sunmadığımı, bu tezin planlanmasından yazımına kadar bütün safhalarda bilimsel etik kurallarına uygun olarak davrandığımı ve aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul edeceğimi beyan ederim.

31 /07/2018

Onur UĞURLU

ÖZET

ROTALAMA PROBLEMLERİ İÇİN ALGORİTMİK YAKLAŞIMLAR

UĞURLU, Onur

Doktora Tezi, Matematik Anabilim Dalı
Tez Danışmanı: Prof. Dr. Urfat NURİYEV
İkinci Danışmanı: Doç. Dr. Murat Erşen BERBERLER
Temmuz 2018, 58 sayfa

Rotalama problemleri, yöneylem araştırması alanındaki en önemli optimizasyon problemleri arasındadır. Rotalama problemlerin optimize edilmesi ile edinilebilecek önemli ekonomik faydalardan dolayı araştırmacılar bu problemlere gitgide daha çok ilgi göstermektedir. Ayrıca, rotalama problemleri NP-zor sınıfına ait olduğundan dolayı, bilgisayar bilimleri literatüründe önemli bir rol oynamaktadır.

Bu tezde, rotalama problemleri araştırılmış ve bu problemlerinin en saf hali olarak görülen Gezgin Satıcı Problemi ve Çoklu Gezgin Satıcı Problemi ele alınmış, problemler için geliştirilen çözüm yöntemleri incelenmiş ve bu problemler için yeni sezgisel algoritmalar önerilmiştir. Önerilen algoritmalar C programlama dilinde kodlanmış, TSPLIB Kütüphanesi örnekleri üzerinde test edilmiş ve literatürde var olan benzer çalışmalar ile kıyaslanmıştır. Hesaplama sonuçları önerilen algoritmaların var olan yöntemlerden daha iyi sonuçlar bulduğunu göstermektedir.

Anahtar sözcükler: Rotalama Problemleri, Kombinatoriyal Optimizasyon, Gezgin Satıcı Problemi, Çoklu Gezgin Satıcı Problemi, Sezgisel Algoritmalar.

ABSTRACT

ALGORITHMIC APPROACHES FOR ROUTING PROBLEMS

UĞURLU, Onur

Ph. D. in Mathematics
Supervisor: Prof. Dr. Urfat NURİYEV
Co-Supervisor: Assoc. Prof. Dr. Murat Erşen BERBERLER

July 2018, 58 pages

Routing problems are among the most important optimization problems in the field of Operations Research. Due to the significant economic benefit that can be achieved by optimizing the routing problems, researchers give more and more attention to these problems. Besides, since routing problems belong to the class of NP-hard, these problems play an important role in computer science literature.

In this thesis, routing problems are investigated and traveling salesman problem and multiple traveling salesman problem, which are considered as pure routing problems, are studied, the solution approaches for the problem are investigated and new heuristic algorithms are proposed for these problems. The proposed algorithms have been implemented in C language, have been tested on the TSPLIB library and compared with a similar literature works. The experimental results show that the proposed algorithms find better solutions than the existing methods.

Keywords: Routing Problems, Combinatorial Optimization, Traveling Salesman Problem, Multiple Traveling Salesman Problem, Heuristic Algorithms.

TEŞEKKÜR

Bu tez çalışmasında her daim desteklerini, bilgi ve tecrübelerini benden esirgemeyen danışman hocalarım Prof. Dr. Urfat NURİYEV'e ve Doç. Dr. Murat Erşen BERBERLER'e, doktora eğitimim boyunca birçok alanda bakış açımı genişleten Prof. Dr. Ali Saffet GÖNÜL'e ve bana yurt dışında araştırma fırsatı sunan Prof. Dr. Ming-Yang KAO'ya en içten teşekkürlerimi sunarım.

Tüm yaşamım boyunca beni sürekli destekleyen ve eğitimim için hiçbir fedakarlıktan kaçınmayan, üzerimde sonsuz emekleri olan annem Elif UĞURLU'ya ve babam Ersoy UĞURLU'ya saygı ve şükranlarımı sunar, varlığıyla bana destek olan sevgili kardeşim Bahar UĞURLU'ya teşekkür ederim.

Son olarak doktora eğitimim süresince 2211/A ve 2214/A burs programları ile eğitimime destek olan TUBİTAK'a ve tez çalışmam boyunca katkılarından dolayı Ege Üniversitesi Rektörlüğü Araştırma Fonu'na teşekkürü bir borç bilirim.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

Sayfa

İÇİNDEKİLER (devam)

Sayfa

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
$\delta(G)$	minimum tepe derecesi.
$\Delta(G)$	maksimum tepe derecesi.

Kısaltmalar

GSP	Gezgin Satıcı Problemi.
TSP	Traveling Salesman Problem.
GSTP	Gezgin Satıcı Tanıma Problemi.
HD	Hamilton Devresi.
HDTP	Hamilton Devresi Tanıma Problemi.
NN	En Yakın Komşu Algoritması.
TWGA	Two-Way Greedy Algorithm.
MDCP	Multiple Depots and Closed Paths.

1. GİRİŞ

Yöneylem araştırması, İkinci Dünya Savaşı'nda yeni bir bilimsel disiplin alanı olarak gelişmiş ve savaştan sonra sanayi, endüstri, üretim, dağıtım gibi birçok alandaki problemlerinin optimizasyonunda yaygın olarak kullanılmaya başlanmıştır (Bakır ve Altunkaynak, 2003). Ulaşım, taşımacılık ve lojistik gibi alanlarda sıkça karşılaşılan rotalama problemleri, yöneylem araştırması alanında en çok çalışılan problemler arasındadır. NP-zor problem ailesine dâhil olmasından dolayı, rotalama problemleri aynı zamanda teorik bilgisayar bilimleri alanında da en çok ilgi gören problemlerdendir.

Rotalama problemlerinin en bilinen ve en ünlü problemi *Gezgin Satıcı Problemi* (*Traveling Salesman Problem*) (GSP), bilgisayar biliminde en çok çalışılan NP-zor problemlerden biridir. Gezgin satıcı problemi, matematik ve bilgisayar bilimindeki uygulamalarının dışında; fizik, kimya, sinirbilim ve bioinformatik gibi birçok bilim dalındaki problemlerin modellenmesinde ve çözümlenmesinde kullanılmaktadır.

Gezgin satıcı problemi, aralarındaki uzaklıkları bilinen n adet noktanın (şehir, tepe veya düğüm) her birinden yalnızca bir kez geçen en kısa veya en az maliyetli turun bulunmasını amaçlayan bir eniyileme problemidir. *Çoklu Gezgin Satıcı Problemi* (*Multiple Traveling Salesman Problem*) ise çok sayıda günlük hayat problemini modellemek için kullanılabilecek gezgin satıcı probleminin daha karmaşık bir türevidir. Çoklu gezgin satıcı probleminde n adet şehir, her biri bir satıcıya atanmak üzere m adet satıcıya/tura bölünmektedir. Geniş uygulama alanı ve teorik bilimsel çalışmalardaki öneminden dolayı, gezgin satıcı problemi için etkin çözüm yöntemlerinin geliştirmesi çok önemlidir.

Bu tezde, rotalama problemlerine günlük hayat uygulamalarından ziyade, teorik anlamda yaklaşılmış ve literatürdeki çözüm yöntemlerinin iyileştirilmesi hedeflenmiştir. Bu doğrultuda rotalama problemlerinin teorik olarak en saf hali olarak görülen gezgin satıcı problemi ve çoklu gezgin satıcı problemi (Anbuudayasankar et al., 2016) ele alınmış, bu problemler için çözüm yöntemleri üzerine araştırmalar yapılmış ve yeni etkin sezgisel algoritmalar üretilmiştir.

Tezin izleyen bölümleri şu şekildedir:

Tezin ikinci bölümünde optimizasyon problemlerinin genel tanımı yapılmış, matematiksel modeli verilmiş ve belirli kriterlere göre oluşturulan sınıflandırmalar anlatılmıştır. Ayrıca optimizasyon problemlerinin karmaşıklık sınıfları ve bu problemleri çözmek için kullanılan yöntemler hakkında bilgi verilmiştir.

Üçüncü bölümde, ilk olarak tezin ilerleyen bölümlerinde kullanılacak olan bazı temel çizge kavramları verilmiştir. Sonrasında rotalama problemlerine genel bir bakış sunularak ilgili sınıflandırmalar verilmiştir ve problemlerin uygulama alanları anlatılmıştır.

Dördüncü bölümde gezgin satıcı problemi ele alınmış, problemin matematiksel modeli verilmiş, NP-tamlığı incelenmiş ve problemin var olan çözüm yöntemlerinden bahsedilmiştir. Ayrıca gezgin satıcı problemi için yeni bir sezgisel algoritma önerilmiş ve bu algoritmayı kullanarak yazılan bilgisayar programıyla yapılan hesaplama denemeleri literatürde var olan çözüm yöntemleriyle kıyaslanarak sunulmuştur.

Beşinci bölümde çoklu gezgin satıcı problemi ele alımış, problemin matematiksel modeli verilmiş ve var olan çözüm yöntemleri incelenmiştir. Problem için açgözlü yaklaşımlı yeni bir sezgisel algoritma önerilmiş ve bilgisayar programı yazılmış, kütüphane örnekleri üzerinde hesaplama denemeleri yapılarak sonuçlar literatürde var olan bir yöntemle karşılaştırmalı olarak sunulmuştur.

Altıncı bölümde tezde elde edilen sonuçlardan bahsedilmiştir.

Tezin son iki bölümünü ise kaynaklar dizini ve önerilen algoritmaların bilgisayar programlarının kaynak kodlarının verildiği ekler oluşturmaktadır.

2. OPTİMİZASYON PROBLEMLERİ

Optimizasyon teorisi literatürü 1665'te Newton'un bir fonksiyonun minimum çözümünü bulan yöntemi önermesiyle başlayıp, İkinci Dünya Şavaşı'ndan sonra sayısal bilgisayarların gelişmesiyle beraber bu alanda büyük ilerlemeler kaydedilmiştir (Gass and Assad, 2004). En genel anlamıyla optimizasyon teorisi, bir problemin çözümünün varlığı, yapısal özellikleri ve çözümünün algoritmik yönleri üzerine çalışmaları kapsamaktadır (Jongen et al., 2004).

Optimizasyon problemleri, karar değişkenlerine bağlı olarak en iyi çözümün araştırıldığı problemlerdir. Optimizasyon problemlerinin amacı; amaç fonksiyonu olarak nitelendirilen fonksiyonun en küçüklenmesi ya da en büyüklenmesidir (Cura, 2008). Genel bir optimizasyon probleminin ifadesi aşağıdaki şekildedir:

Amaç fonksiyonu:

$$Enb\,(veya\,Enk)\ f(x) \tag{2.1}$$

Kısıtlar:

$$g_i(x) \geq b_i, \qquad i = \overline{1,m} \tag{2.2}$$

$$h_j(x) = c_j, \qquad j = \overline{1,n} \tag{2.3}$$

$$p_k(x) \leq d_k, \qquad k = \overline{1,s} \tag{2.4}$$

$$x \in G \tag{2.5}$$

Bir sistemin davranışını matematik dilini kullanarak (2.1) – (2.5) ifadelerinde olduğu gibi tanımlayan soyut modellere *matematiksel model* denir. Bu matematiksel modelde (2.2) – (2.4) modelin kısıtlarını, (2.5) modelin karar değişkeni *x*'i göstermektedir. *x* karar değişkeni herhangi bir *G* kümesinin elemanıdır. (2.2) – (2.5) kısıtları altında (2.1) ifadesi modelin amaç fonksiyonunu göstermektedir (Cura, 2008).

Optimizasyon problemleri çeşitli şekillerde sınıflandırılabilir: $f(x)$ amaç fonksiyonunun x karar değişkeni ile ilgili herhangi bir sınırlama olmaksızın optimizasyonu *kısıtlamasız optimizasyon (unconstrained optimization)*, x karar değişkeni ile ilgili sınırlamanın olduğu optimizasyon problemleri ise *kısıtlamalı optimizasyon (constrained optimization)* olarak adlandırılır (Karaboğa, 2004).

Bir optimizasyon probleminin amaç ve kısıt fonksiyonlarının tümü doğrusal fonksiyon ise bu probleme *doğrusal programlama problemi (linear programming problem)*, bu fonksiyonlardan herhangi biri doğrusal değil ise *doğrusal olmayan programlama problemi (nonlinear programming problem)* denir. Karar değişkenleri negatif olmayan tamsayı değerler alan doğrusal programlama problemine *tamsayılı programlama problemi*, aksi halde *tamsayı olmayan programlama problemi* denir (Karaboğa, 2004).

Optimizasyon problemleri değişkenlerinin türüne göre *sürekli (continiuous) optimizasyon* ve *ayrık (discrete) optimizasyon* problemleri olarak iki kategoriye ayrılır. Sürekli optimizasyonda karar değişkenleri reel sayıların bir kümesi üzerinde bir değer alabilir. Ayrık problemlerde ise sonlu bir kümeden bir nesne ya da sayılabilir sonsuz bir küme (tipik olarak bir tamsayılar kümesi, çizgeler ya da permütasyonlar) üzerinde tanımlanmış bir fonksiyon araştırılır. Ayrık problemlere aynı zamanda kombinatoryal (*combinatorial*) problemler de denir (Karaboğa, 2004; Cura, 2008). Bu tezde ele alınan gezgin satıcı problemi ve çoklu gezgin satıcı problemi kombinatoryal optimizasyon problemleridir.

2.1 Problem Karmaşıklığı, P ve NP Sınıflar

Bir problemin çözümüne yönelik bir algoritma araştırılmadan önce bu problemin sonlu sayıda aşamada çözülüp çözülemeyeceğinin bilinmesi gereklidir. Bu sebeple problemler karmaşıklarına göre belirli sınıflara ayrılmaktadır.

2.1.1 Problem, Algoritma ve Algoritmaların Karmaşıklığı

Problem örneği, bir çözüm bulmak için hakkında yeterli bilgiye sahip olunan veri gruplarıdır. Problem ise genellikle benzer yolla üretilmiş örnekler topluluğudur. Örneğin; gezgin satıcı probleminin örneği, verilen bir uzaklık matrisidir. Gezgin satıcı problemi ile kast edilen ise tüm uzaklık matrislerinin birleşiminden oluşan örnekler topluluğudur (Papadimitriou and Steiglitz, 1998).

Bir optimizasyon problemi, optimizasyon versiyonu ve tanıma versiyonu olma üzere iki türlü ifade edilebilir. Bir problemin optimizasyon versiyonunda her uygun çözüm bir aday çözümdür ve bu çözümler arasında en iyi değere sahip uygun çözüm aranmaktadır. Örneğin, gezgin satıcı probleminde herhangi tepe permütasyonu uygun bir çözüm iken en kısa uzunluklu tur (tepe permütasyonu) aranmaktadır. Tanıma versiyonunda ise, verilen bir sayısal sınır B'den daha iyi maliyete sahip çözüm olup olmadığı aranmaktadır. Örnek olarak, "Gezgin satıcı tanıma probleminde B'den daha az maliyetli bir tur var mıdır?" sorusuna cevap aranır. Tanıma problemleri cevapları "evet" veya "hayır" olan sorulardır (Garey and Johnson, 1979; Cormen et al., 2001).

Algoritma, belirli bir işin veya problemin sonucunu elde etmek için art arda uygulanacak adımları ve koşulları belirten çözüm tekniğidir. Bir algoritmanın *hesaplama karmaşıklığı (computational complexity)*, algoritmanın ihtiyaç duyduğu işlem sayısı ve bilgisayarın belleğini ne kadar kullanacağı hakkında bilgi vermektedir. Hesaplama karmaşıklığı, hesaplamayı yapmak için gerekli zamanı ölçen *zaman karmaşıklığı (time complexity)* değerlendirilmesi ve hesaplama için gerekli bellek alanının ölçümü olarak *alan karmaşıklığı (space complexity)* olmak üzere iki farklı açıdan incelenmektedir. Zaman karmaşıklığı genellikle sadece karmaşıklık olarak adlandırılmaktadır. Bir algoritmanın iyi ya da kötü olmasını o algoritmanın zaman ve alan karmaşıklığı belirler (Nabiyev, 2009).

2.1.2 Asimptotik Notasyonlar

Algoritmaların karmaşıklarını ifade etmek için çeşitli asimptotik ifadeler kullanılmaktadır (Çölkesen, 2002).

Büyük O notasyonu (O):

Bağıntıyı üstten sınırlar; bu sınırlama $n > n_0$ için k çarpanı ile yapılır. Eğer $n_0, k > 0$ ise ve $f(n)$ bağıntısı $k \cdot g(n)$'in altında kalıyorsa yani $f(n) \leq k \cdot g(n)$ ise $f(n) = \mathrm{O}(g(n))$ şeklinde gösterilir.

Büyük Omega notasyonu (Ω):

Bağıntıyı alttan sınırlar; bu sınırlama $n > n_0$ için k çarpanı ile yapılır. Eğer $n_0, k > 0$ ise ve $f(n)$ bağıntısı $k \cdot g(n)$'in üstünde kalıyorsa yani $f(n) \geq k \cdot g(n)$ ise $f(n) = \Omega(g(n))$ şeklinde gösterilir.

Teta notasyonu (Θ):

Bağıntıyı alttan ve üstten sınırlar; bu sınırlama $n > n_0$ için k_1 *ve* k_2 çarpanları ile yapılır. Eğer $n_0, k_1, k_2 > 0$ ise ve $f(n)$ bağıntısı $k_1 \cdot g(n)$ ile $k_2 \cdot g(n)$ arasında kalıyorsa yani $k_1.g(n) \leq f(n) \leq k_2 \cdot g(n)$ ise $f(n) = \Theta(g(n))$ şeklinde gösterilir.

Çalışma zamanının üst sınırı olan *Büyük O* (*Big O*) notasyonu, en kötü durumdaki çalışma zamanının belirlenmesinde kullanılır ve algoritma karşılaştırmalarında genel olarak kullanılan notasyondur (Nabiyev, 2009). Bu tezde kullanılan tüm karmaşıklar *Büyük O* notasyonu cinsindendir.

2.1.3 P, NP sınıflar, NP-tamlık ve NP-zorluk

Bir algoritmanın zaman karmaşıklığı, belli boyutlardaki girdi değerleri için harcanan tüm hesapsal adımların sayısını veren bir fonksiyondur. *n* girdi verisinin boyutunu göstermek üzere, bir algoritmanın çalışma zamanı üstten herhangi bir $P(n)$ polinomu ile sınırlı ise buna *polinom sınırlı algoritma* denir ve bu algoritmalarla çözülebilen tüm problemler *P sınıfı* olarak tanımlanır (Nabiyev, 2009).

Bir problemin çözümü için hazır formüller ve çözüme ilişkin özyinelemeli ifade bilinmediği zaman, çözüm için tarama ve sezgisel seçimlerin yapılması gerekliliği doğar. Bir problemin karmaşıklığı polinom biçimde tanımlanabilirse, bu problemin sonlu sayıda adımla polinom zamanda çözülebileceği söylenebilir. Polinom zamanlı algoritmaları çalıştıran sanal modeller *Deterministik Turing Makineleridir* (DTM) ve *P* sınıfı, DTM ile polinom zamanda çözülebilen problemler sınıfıdır. Polinom zamanda *deterministik olmayan Turing makineleriyle* (NDTM) çözülebilen problemler ise *NP* (*Nondeterministic Polinomial*) sınıfını oluşturur (Garey and Johnson, 1979). *P* sınıf problemlerine *NP* sınıfın bir alt kümesi olarak bakılabilir, $P \subset NP$. Çünkü NDTM, DTM ile çözülebilen tüm problemleri çözebilir (Cormen et al., 2001).

P sınıfı, polinom zamanda çözülebilen problemlerden oluşmaktadır. Bu problemler, *k* bir sabit ve *n* problem girdisinin boyutu olmak üzere $O(n^k)$ zamanda çözülebilen problemlerdir. *NP* sınıfı, polinom zamanda doğrulanabilen problemlerden oluşur. Yani, problemin herhangi bir çözüm sertifikası verildiğinde, problemin girdi boyutuna bağlı olarak bu sertifikanın polinom zamanda doğrulanabildiği problemler sınıfıdır.

Bir Π probleminin NP-tam sınıftan olabilmesi için, öncelikle bu Π probleminin *NP* sınıftan olduğu ve sonrasında da *NP* sınıftaki her problemin polinomiyal zamanda bu probleme indirgenebileceği gösterilmelidir (Garey and Johnson, 1979):

1. $\Pi \in NP$ ve

2. $\forall \Pi' \in NP$ için $\Pi' \leq_P \Pi$.

Bu adımlardan birincisi değil sadece ikincisi sağlanırsa Π problemi '*NP-zor* sınıftandır' denir.

Bir Π probleminin NP-tamlığının ispatı için aşağıdaki adımlar izlenmelidir:

1. Π 'nin *NP* sınıfından olduğu gösterilir,

2. Bilinen bir Π' NP-tam problemi seçilir,

3. Π' probleminden Π problemine bir *f* dönüşümü oluşturulur ve

4. *f* dönüşümünün bir polinomiyal dönüşüm olduğu ispatlanır.

2.2 Kombinatoryal Optimizasyon Problemlerinin Çözüm Yöntemleri

Kombinatoryal optimizasyon problemlerinin çözüm yöntemleri kesin ve yaklaşık çözüm yöntemleri olmak üzere ikiye ayrılmaktadır. Kesin yöntemler optimum çözümü garanti etmelerine rağmen özellikle problemlerin boyutları arttıkça hesaplama zamanı ve bellek gereksinimi nedeniyle makul süreler içinde gerçekleştirilemezler. Böyle durumlarda yaklaşık yöntemler daha çok tercih edilmektedir. Yaklaşık yöntemler optimum çözüme ulaşmayı garanti etmeseler de makul süreler içinde optimum çözüme yakın çözümler üretebilirler.

2.2.1 Kesin yöntemler

Bu sınıftaki yöntemlerden bazıları şunlardır:

- *Sayımlama* (*Enumeration*),
- *Kesen düzlemler* (*Cutting planes*),
- *Dinamik programlama* (*Dynamic programming*),
- *Dal-sınır* (*Branch and bound*),
- *Dal-kesme* (*Branch and cut*).

2.2.1.1 Sayımlama Yöntemi

Sayımlama yöntemi tüm olasılıkları saymaya dayanan bir yöntemdir. Arama uzayındaki her durum için amaç fonksiyonunun değeri hesaplanır ve sonunda bulunan değerler içinden en iyi olan çözüm seçilir. Bu yöntem değişken sayısının az olduğu problemlerde kullanışlı olmasına rağmen, problemin boyutu arttıkça incelenen olasılıklar üstel biçimde artar. Örneğin n değişkenli 0-1 sırt çantası probleminde, her bir nesnenin çözümde olup olmamasına bağlı olarak incelenmesi gereken olasılık sayısı 2^n'dir. Olasılık sayısının üstel olarak artması nedeniyle büyük n değerleri için şu andaki mevcut bilgisayar teknolojisi ile çözüm süresi yüz yıllar alabilir (Woolsey, 1998).

2.2.1.2 Kesen Düzemler Yöntemi

Kesen düzlemler yöntemi tamsayılı doğrusal programlama problemlerini çözmek için ilk önerilen yöntemdir. Bu yöntem, tamsayılı programlama probleminin doğrusal programlama problemine gevşetilmesine dayanır. Yani tamsayılı programlama problemindeki tüm kısıtlar olduğu gibi ele alınırken, sadece değişkenlerin tamsayı olması kısıtı kaldırılır. Daha sonra doğrusal programlama metotlarından herhangi biri kullanılarak sonuç bulunur. Bulunan sonuç tamsayı ise çözüm elde edilmiştir. Aksi halde tamsayı olmayan çözümleri dışarıda bırakacak aynı zamanda tüm tamsayı çözümleri koruyacak yeni bir kısıt (kesme) eklenir. Bu işlemlere optimal tamsayı çözüm bulunana kadar devam edilir.

Kesme yöntemleri genelde düzensizlerdir. Ayrıca tüm verilerin simpleks tablo şeklinde saklanması büyük boyutlu problemler için sorun yaratmaktadır (Papadimitriou and Steiglitz, 1998).

2.2.1.3 Dinamik Programlama

Dinamik programlama optimumluk ilkesine dayalı çözümleme yöntemlerine verilen isimdir. Burada 'programlama' kelimesi bilinen kodlama biçiminden farklı olarak 'planlama' anlamında düşünülebilir. Bu yönteme göre optimum çözüm şu özelliğe sahiptir: başlangıç durumundan bağımsız olarak diğer çözümler ilk çözüm sonuçlarına göre optimum çözümler ardışıklığıdır. Bu tekniğin problem çözümünde kullanılması problemlerin alt problemlere ayrılması anlamına gelir. Yani ilk olarak alt problemlerin çözümü bulunur, sonra daha büyük alt problemler incelenir ve sonunda problemin kendisi çözülmüş olur (Nabiyev, 2009).

Kesin çözüm yöntemlerinin içinde dinamik programlama özel bir yere sahiptir. İleri yineleme ve geri yineleme olarak temelde iki biçimde uygulanabilir. İleri yineleme, problemin başındaki en küçük birimden başlayarak sonuca ulaşma işlemi, geri yineleme ise problemin çözümündeki son aşamadan başlayarak sonuca ulaşma işlemi olarak ifade edilir (Cormen et al., 2001).

2.2.1.4 Dal-Sınır Yöntemi

Dal-sınır yöntemi, optimal çözüme ulaşmak için sistemli bir şekilde uygun çözümler saymaya dayanır. Ancak sayımlama yöntemindeki gibi tüm olasılıkları dikkate almak yerine, olasılıkların daha az bir kısmı incelenir ve gereksiz görülen bazı olasılıklar kesip atılır. Bu yöntemin avantajları; hızlı olması, verilmiş herhangi bir tamsayılı problemin özelliklerinin göz önünde bulundurulması ve yöntemin içinde başka tamsayılı yöntemlerin kullanılmasına imkan vermesidir. Hesaplama sürecinde ortaya çıkan çözümler genellikle iyi yaklaşık çözümlerdir. Ancak dallanma sayısının fazlalığından dolayı bilgisayar belleğine daha çok gerek duyması bu yöntemin dezavantajıdır. Ayrıca değişken sayısının artması hesaplama süresini üstel olarak arttırır ve bulunan çözümün optimal çözüm olduğunu ispatlamak çalışma zamanından daha fazla zaman gerektirebilir (Papadimitriou and Steiglitz, 1998).

2.2.1.5 Dal Kesme Yöntemi

Dal-kesme algoritması dal-sınır yöntemi ile kesen düzlemler yönteminin birlikte kullanımı ile oluşur. Verilen problem için öncelikle kesen düzlemler yöntemi kullanılır, kesen düzlemler bir tamsayı çözüm bulunana kadar veya daha fazla kesme bulunamayıncaya kadar devam eder. Daha sonra dal-sınır yöntemi kullanılır ve bulunan tamsayıya göre dallanarak çözüm bulununcaya kadar çalışmasını sürdürür (Bakır ve Altunkaynak, 2003).

2.2.2 Yaklaşık Yöntemler

Yaklaşık yöntemler, kesin yöntemlerin aksine optimal çözümü garanti etmezler ancak kesin çözüm yöntemlerinin çözüm bulmak için yoğun hesaplama ve çözüm zamanı gerektirdiği durumlarda yaklaşık yöntemler tercih edilmektedir. Bu sınıftaki yöntemlerden bazıları şunlardır:

- *Açgözlü algoritmalar* (*Greedy algorithms*),
- *Yerel arama* (*Local search*),
- *Genetik algoritmalar* (*Genetic algorithms*),
- *Tabu araması* (*Tabu search*),
- *Karınca kolonisi optimizasyonu* (*Ant colony optimization*).

2.2.2.1 Açgözlü Algoritmalar

Açgözlü algoritmalar programlama kolaylığı ve çalışma hızı bakımından oldukça kullanışlıdır. Bu yöntemler, global optimal çözüme erişmek amacıyla yerel optimal seçimler yaparlar ve seçimlerini tekrar gözden geçirmezler. Açgözlü yöntem başlangıç olarak nesneleri bazı kriterlere göre sıraya koyar ve boş kümeden başlayarak çözüm kümesini genişletir (Cormen et al., 2001). Açgözlü algoritmalar her zaman olmasada bazı problemler için optimal çözümü verirler.

2.2.2.2 Yerel Arama

Yerel arama algoritması başka algoritmalar tarafından bulunmuş bir çözümle çalışmaya başlar, daha iyi bir komşu çözüme ulaşmak için iteratif olarak mevcut çözümü geliştirir. Çözümün daha fazla geliştirilemediği durumda algoritma bir yerel optimumda sonlanır (Aarts and Lenstra, 1997).

2.2.2.3 Tabu Arama

Tabu araması hafızaya dayalı bir arama tekniğidir. Önceki adımlarda elde edilen bilgi daha sonraki adımlarda ilerlemeleri belirlemek için kullanılmaktadır. Bu yöntemde amaç, miyop yaklaşımın neden olduğu risklerden kaçınmak için basit yerel aramayı genelleştirmektir. Tabu araması, sonraki adımlarda daha iyi bir sonuca götürebilir düşüncesiyle bulunduğu çözümden daha kötü sonuç veren bir komşuya geçişine de izin vermektedir (Glover and Laguna, 1997).

2.2.2.4 Genetik Algoritmalar

Evrimsel programlamanın bir parçası olan genetik algoritmalar Darwin'in evrim teorisinden esinlenerek ortaya çıkmıştır. Genetik algoritmalar, en iyi olanın yaşaması ve doğal seçim mekanizmasını temel alan bir arama yöntemidir. Temel bir genetik algoritma rastgele aday çözümlerin popülasyonunun oluşturulması ile başlar. Daha sonra bazı (veya tüm) aday çözümlere çaprazlama ve mutasyon işlemleri uygulanır. Her bir aday çözüm için onların ne kadar iyi olduğunu gösteren bir uygunluk fonksiyonu kullanılır. En uygun aday bireylerin çaprazlama ve mutasyon için seçilmesi ile popülasyonun genel uygunluğu arttırılabilir. İstenilen şartlar sağlanıncaya kadar nesillerin yeniden üretilmesine devam edilir (Nabiyev, 2005).

2.2.2.5 Karınca Kolonosi Optimizasyonu

Karınca kolonileri optimizasyonu, karıncaların gerçek beslenme davranışından ilham alan bir yöntemdir. Karıncalar, salgıladıkları feromon maddesini kullanarak, yuvaları ve yemek kaynakları arasındaki en kısa yolu bulma yeteneğine sahiptirler. Bu özellikleri sayesinde karmaşık bir optimizasyon problemiyle benzer özellik gösteren yol bulma problemlerini çözmektedirler. Aynı zamanda kullandıkları bir yol kullanıma kapandığında veya daha kestirme bir yol alternatifi ortaya çıktığında yeni en kısa yolu en kısa zamanda

bulabilmektedirler. Doğadaki bu yapının fark edilmesiyle karınca kolonileri gözlemlenmiş ve optimizasyon problemlerinin çözümlerine yönelik bazı yapay sistemler oluşturulmuştur (Cura, 2008; Başkaya, 2005).

3. ROTALAMA PROBLEMLERİ

Bu bölümde, ilk olarak tezin ilerleyen bölümlerinde kullanılan çizge kuramındaki bazı temel kavramlar verilmiş, sonrasında rotalama problemlerine genel bir bakış sunularak ilgili sınıflandırmalardan ve problemlerin uygulama alanlarından bahsedilmiştir.

3.1 Çizgeler ile İlgili Bazı Temel Kavramlar

Bu bölümde yer alan tanımlar 'Graph Theory and Its Applications' (Gross and Yellen, 2004), 'Graph Theory: Modeling, Applications, and Algorithms' (Agnarsson and Greenlaw, 2007) ve 'Graph Theory' (Bondy and Murty, 2008) adlı kitaplardan alınmıştır.

Tanım 3.1: V tepeler kümesi $V \neq \varnothing$ ve $E \subseteq V \times V$ ayrıtlar kümesini oluşturmak üzere $G = (V, E)$ ifadesine *çizge (graf)* denir. Çizgenin *tepe sayısı* $|V| = n$ ve *ayrıt sayısı* $|E| = m$ şeklinde gösterilir.

Tanım 3.2: Bir G çizgesi üzerindeki u ve v tepeleri, ayrıtlar kümesinde bulunan bir ayrıtla ilişkilendiriliyorsa bu tepeler *komşu tepe* adını alır ve $e = \{u, v\}$ şeklinde gösterilir. Diğer bir anlatımla e ayrıtı u ve v tepelerini birbirine bağlar veya u ve v tepeleri e kenarının *uç tepeleridir* denir.

Tanım 3.3: Bir G çizgesinde herhangi bir $v \in V$ tepesine bitişik ayrıtların sayısına *v tepesinin derecesi* denir ve $\deg(v)$ ile gösterilir. G çizgesinin en küçük tepe derecesi $\delta(G)$, en büyük tepe derecesi ise $\Delta(G)$ ile gösterilir.

Tanım 3.3: Bir G çizgesinde birbirine komşu tepelerin ve ayrıtların ardışık dizisine *yürüyüş(walk)* denir.

Tanım 3.4: Bir G çizgesinde ayrıt tekrarı içermeyen yürüyüşe zincir (*trail)* denir.

Tanım 3.5: Bir G çizgesinde tepe tekrarı içermeyen yürüyüşe *yol (path)* denir.

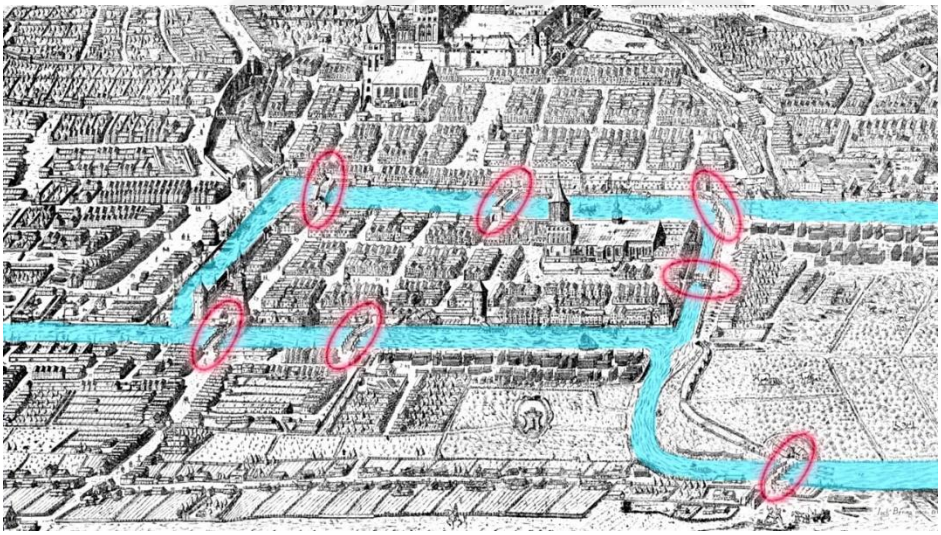
Tanım 3.6: Bir G çizgesinde aynı tepe ile başlayıp biten yürüyüşe *kapalı yürüyüş (closed walk)* denir.

Tanım 3.7: Bir G çizgesinde ara tepeleri tekrar içermeyen kapalı yürüyüşe *çevre/tur/devre (cycle)* denir.

Tanım 3.8: Bir G çizgesinde çizgenin tüm tepelerini içeren basit devreye *Hamilton Devresi* denir.

3.2 Rotalama Problemlerine Tarihsel Bir Bakış

1741 yılında Leonhard Euler '*in the Commentarii of the Saint Peterburg Academy*' isimli makalesinde '*Konigsberg'in Yedi Köprüsü*' adlı problem ile ilgili bir makale yayınlamıştır. Königsberg günümüzde Rusya'da Kaliningrad adıyla yer alan, tarihte ise Alman Doğu Prusya eyaletinin başkenti olan bir şehirdir. Bu şehirde Eski ve Yeni Pregel nehirleri birleşerek Pregolya nehrini oluşturmaktadır. Bu nehirler şehri dört bölgeye ayırmaktadır ve nehirler üzerine kurulan yedi köprü ile bu bölgeler birbirlerine bağlanmıştır. Bu köprülerle ilgili şu soru sorulmuştur: Şehrin herhangi bir bölgesinden başlayıp her köprüden sadece bir kere geçen bir tur oluşturulabilir mi ?



Şekil 3.1. Königsberg'in yedi köprüsü

Euler bu problem için çizge kuramının doğuşu olarak görülen bir matematiksel formülasyon önermiş ve sonrasında Könisberg köprüleri için böyle bir turun olamayacağını göstermiştir. Daha genel bir durumu ele alarak, sonrasında adıyla anılan ve günümüzde 'Euler Turu' olarak bilinen bu turun var olması için bazı gerek ve yeter koşulları sunmuştur. Bu problemle beraber Euler aynı zamanda rotalama problemleri üzerine çalışmaya başlamıştır (Bruno et. al, 2011).

1962'de, Çin Halk Cumhuriyet'inde, gençliğinde postacı olarak çalışmış bir matematikçi olan Mei-Ko Kwan, Euler problemini genişletme fikrini şu şekilde anlatmıştır; 'Varsayalım ki bir mahallede bir postacı olsun. Bu postacı tembel olduğu için, her bir sokaktan en an az bir kere geçen ve başladığı noktada sonlanan en kısa turu bulmak istemektedir'. Alan Goldman problemin doğasından ve problemi ilk kez ortaya koyan araştırmacının milliyetinden etkilenerek, problem için *Çinli Postacı Problemi* (*Chinese Postman Problem*) ismini önermiştir. Goldman bu fikrini kombinatoryal optimizasyonun babası olarak görülen Jack Edmonds'a önermiş ve Edmonds problemi bu isimle kullanmaya başlamıştır. Mei-Ko Kwan'ın problemi Euleryan olmayan çizgeleri de kapsadığı için, Euler'in probleminden daha geniş olduğu açıktır. Könisberg'in yedi köprüsü problemi ve çinli postacı problemi yol ve turlardan oluşan toplama ve dağıtma problemleriyle ilgilenen ayrıt rotalama problemlerini için bir başlangıç noktasıdır (Bruno et. al, 2011).

Gezgin satıcı probleminin ilk hangi tarihte problem olarak ele alındığı net olarak bilinmesede, 1832 yılında Almanya'da basılmış olan 'Der Handlungsreisende' isimli bir kitapta problemin temellerinden bahsedilmiştir. Kitapta problemle ilgili cümleler şu şekildedir (Bruno et. al, 2011):

'Görev gezgin satıcıyı, bugün oraya yarın buraya gönderir ve hiçbir seyahat rotası oluşabilecek tüm durumlar için uygun değildir. Ama bazen uygun seçim ve ayarlamalarla çok zaman kazanılabileceğinden, bu konuda bazı kurallar getirmekten kurtulabileceğimizi sanmıyoruz. Herkes bu kurallardan kendi amacına yarayacak kadarını kullanabilir, yine de şunu garanti ederiz ki, gidiş-dönüş mesafeleri düşünüldüğünde, Almanya üzerinde daha ekonomik bir tur ayarlamak çok olanaklı değildir. Buradaki asıl önemli nokta, aynı yerden iki kere geçmeden olabildiğince çok yeri ziyaret etmektir.'

Bu kitapçık, hiçbir matematiksel hesaplama olmadan sadece gözleme dayalı bilgiyle, 45 Alman şehri arasında 5 olası tur önermiştir. 19. yüzyılın ortalarında, İngiltere'de iki matematikçi probleme teorik katkıda bulunmuştur. 1856'daki makalesinde Thomas Penyngton Kirkman, gezgin satıcı problemi için ilk grafik formülasyonu önermiştir: 'Bir çokyüzlünün çizgesi verildiğinde, her bir tepeden sadece ve sadece bir kere geçen bir kapalı tur elde etmek mümkün müdür?'

1857’de, İrlandalı matematikçi William Rowan Hamilton, biraz ilginç bir işe girişerek, her noktanın bir şehirle ilişkilendirildiği bir düzgün onikiyüzlü üzerinde ‘*Icosaian Oyunu*’ adında bir oyun geliştirir. Oyun, herhangi bir şehirden yani tepeden başlayıp, başlangıç şehrine geri dönmeden her bir şehri sadece ve sadece bir kez ziyaret etmeye dayanmaktadır. Şekil 3.2’de icosian oyununun onikiyüzlünün ayrıt ve tepelerinin şematik gösterimine dayalı olan düzlemsel bir versiyonu gösterilmektedir (Bruno et. al, 2011).

Şekil 3.2. İcosyan oyunu örneği

Hamilton’ın oyununda, Euler’in probleminde olduğu gibi, asıl amaç bir turun oluşturulmasıdır. Dolayısıyla çözümleri rotalama problemleri ile ilişkilidir. Ancak iki problem arasında önemli bir kavramsal fark vardır: Euler probleminde bütün ayrıtlar üzerinden sadece ve sadece bir kere geçen bir tur arıyorken, Hamilton tüm tepelerden sadece ve sadece bir kere geçen bir tur aramaktadır. Dolayısıyla iki problem birbirini tümlemektedir.

Gezgin satıcı probleminin ilk matematiksel formülasyonu, 1930’lu yıllarda Avusturyalı matematikçi Karl Menger tarafından sunulmuştur. Menger problemi ilk olarak *Haberci Problemi* (*Messeger Problem*) olarak isimlendirmiş ve problemin zorluklarını aşağıdaki gibi belirtmiştir (Bruno et. al, 2011).:

‘Tüm ikilileri arasındaki uzunlukları bilinen sonlu sayıdaki nokta için, en kısa yolu bulma problemini, birçok gezginin ve postacının karşılaştığı bir problem olduğu için, haberci problemi olarak adlandırdık. Problem doğal olarak sonlu sayıda deneme ile çözülebilir. Başlangıç noktasından en yakın noktaya, oradan da o noktaya en yakın olana gitme yöntemi her zaman en kısa yol ile sonuçlanmaz.’

3.3 Gezgin Satıcı Probleminin Tanımı ve Uygulama Alanları

Gezgin satıcı problemi kombinatoryal optimizasyon alanında en çok bilinen problemlerden birisidir. Problemin kolay formüle edilmesi, zor çözülmesi ve çok sayıda uygulama alanının olması popülerliğinin artmasını sağlamıştır. Gezgin satıcı problemi eski bir problem olmasına karşın, günümüzde hala birçok araştırmacının ilgisini çeken bir problemdir.

Gezgin satıcı problemi, aralarında uzaklıkları bilinen n tane şehrin her birisinden yalnızca bir kez geçen ve tekrar başlangıç noktasına dönen en kısa maliyetli turun bulunması problemidir. Eğer şehirler tepelerle, şehirler arasındaki yollar ise ayrıtlarla gösterilirse problem bir tam çizge üzerinde tüm tepeleri içeren en az maliyetli turun bulunmasına karşılık gelir. $V = \{v_1,..., v_n\}$ şehirlerin kümesi, $E = \{(i, j) : i, j \in V\}$ şehirler arası mesafelerin kümesi olsun. Eğer $(i, j) \in E$ ayrıtına ait maliyet $d_{ij} = d_{ji}$ ise yani iki şehir arasındaki maliyet her iki yönden de eşit ise bu tür gezgin satıcı problemine *Simetrik Gezgin Satıcı Problemi* denir. Eğer $\exists (i, j) \in E$ ayrıtına ait maliyet her iki yönden de farklı ise yani $d_{ij} \neq d_{ji}$ ise o zaman bu tür gezgin satıcı problemleri *Asimetrik Gezgin Satıcı Problemi* olarak adlandırılır. Bu tezde aksi belirtilmediği sürece gezgin satıcı problemi ile ele alınan simetrik gezgin satıcı problemidir.

Gezgin satıcı problemi ilk bakışta çok karmaşık görünmemesine rağmen, çözümü oldukça zor problemlerden biridir. Çözüm uzayının taranmasını gerektiren bilinen çözüm yöntemleri ile problemin optimum çözümü şehir sayısının artması ile neredeyse imkansız hale gelmektedir. Yani n ziyaret edilmesi gereken şehir sayısı olmak üzere, uygun çözüme ulaşmak için incelenmesi gereken tüm şehirleri kapsayan olası farklı turların toplam sayısı $(n-1)!/2$ dir.

İlk bilgisayarların ortaya çıkışı ve kombinatoryal optimizasyonun alanındaki teorik gelişmeler ile birlikte 1950'lerden sonra gezgin satıcı problemi ile ilgilenen bilim adamı ve araştırmacı sayısı artmıştır. George B. Dantzig, Delbert R. Fulkerson ve Selmer M. Johson tarafından oluşturulan bir araştırma ekibi, 1954'te bir tamsayılı doğrusal programı tasarlamış ve 49 Amerikan Şehrini içeren gezgin satıcı problemi için en iyi sonucu bulan bir yöntem geliştirmiştir (Dantzig et. al, 1954). Günümüzde hızla gelişen bilgisayar teknolojisi ve önerilen yeni çözüm yöntemleri ile birlikte büyük boyutlardaki problemler de çözülebilir hale gelmiştir. 2001'de Applegate, Bixbyi Chvatal ve Cook, Gezgin Satıcı problemini, Princeton

ve Rice Üniversitelerindeki paralel iş istasyonları ağını kullanarak 15.112 Alman şehri için çözmüşlerdir. Bundan üç yıl sonra yine Applegate ve arkadaşları İsveç'in şehir ve köylerini temsil eden 24.978 tepeli bir çizgede 72.500 km'lik bir tur bularak, bu çözümden daha iyi bir çözüm olmayacağını göstermişlerdir (Bruno et. al, 2011).

Gezgin Satıcı Problemi zengin bir uygulama alanına sahiptir. Problem doğası gereği rotalama ve lojistik gibi alanlarda ortaya çıkmış olsa da, ilk bakışta bir ilgisi olmayan pek çok alandaki çeşitli soruların ele alınıp modellemesinde ve çözümünde kullanılmaktadır. Elektronik aletler endüstrisindeki, entegre devre bileşenlerinin üretim süreçleri buna örnek olarak verilebilir. Devre kartlarına delikler açan makineler, delikler ziyaret edilecek tepeler, maliyetler de makine kolunun bir delikten diğerine geçmesi için gereken zaman olarak ele alınarak, Gezgin satıcı problemi ile modellenip programlanmaktadır. İnsan genomu çalışmalarında da, hibrid radrasyon haritalarının yapımında gezgin satıcı problemi kullanılmaktadır. Bu yöntem; yerel haritaları tek bir global haritada toplamaya imkan vermektedir. Gezgin satıcının buradaki kullanımında, local haritalar tepeleri, maliyetler ise genom haritası oluştururken birbirini takip eden haritaların birbirinin ardından gelme olasılığı ile hesaplanmaktadır. (Agarwala et al., 2000). Gezgin satıcı probleminin sıklıkla kullanıldığı bir başka alan da veri kümelemedir. İkililer arasındaki benzerlik oranları maliyet olarak ele alınarak, bu değerlere göre aday kümeler oluşturulabilir (Jain et. Al, 1999).

Gezgin satıcı probleminin kullanıldığı diğer günlük hayat uygulama alanlarından bazıları aşağıda verilmiştir:

- Bilgisayar elektrik hatlarının kurulumu (Computer Wiring) (Lenstra and Kan, 1974)
- Psikoloji (Psychology) (Hubert and Baker, 1978)
- X-Ray Kristalografisi (X-Ray crystallography) (Bland and Shallcross, 1989)
- Gaz Türbini Motorlarının Onarımı (Overhauling Gas Turbine Engines)(Plante, 1988)
- Makine Çizelgeleme (Machine-Scheduling) (Gilmore and Gomory, 1964)

3.3.1 Gezgin Satıcı Probleminin Varyasyonları

Gezgin satıcı probleminin çeşitli günlük hayat ya da potansiyel uygulamalardan dolayı ortaya çıkan birçok varyasyonu vardır. Göreceli olarak basit dönüşümlerle gezgin satıcı problemi olarak tekrar formüle edilebilen varyasyonlardan bazıları bu bölümde verilmiştir. Bu bölümde yer alan tanımlar 'The Travelling Salesman Problem and Its Variations' (Gutin and Punnen,2002) adlı kitaptan alınmıştır.

3.3.1.1 Max GSP

Gezgin satıcı probleminin aksine, buradaki amaç maliyeti maksimum olan bir tur bulmaktır. MAX GSP ayrıtların değerlerini toplamaya göre tersleri ile yer değiştirilerek Gezgin satıcı problemi gibi çözülebilir.

3.3.1.2 Çoklu Ziyaretli GSP (TSP with Multiple Visit)

Problemin bu varyasyonunda, herhangi bir tepeden başlayıp her bir tepeden en az bir kere geçip başladığı tepeye geri dönen minimum maliyetli tur aranmaktadır. Çizgedeki ayrıtların değerleri en kısa yol değerleri ile değiştirilirse problem gezgin satıcı problemi şeklinde formüle edilebilir.

3.3.1.3 Kümelenmiş GSP (Clustered TSP)

Bir G çizgesinin tepeleri $V_1, V_2, ..., V_k$ şeklinde kümelere ayrıldığında, Kümelenmiş GSP her bir kümenin içindeki tepeler birbirini takip edecek şekilde en kısa turun bulunmasını hedefler. Bu problem kümeler arasındaki ayrıtlara büyük bir değer eklenerek Gezgin satıcı problemine dönüştürülebilir.

3.3.1.4 Zaman Bağımlı GSP (The Time Dependent TSP)

Gezgin satıcı probleminin bu varyasyonunda ayrıtların çözüme girme sırası da çözümün değerini etkilemektedir. Her bir $(i, j) \in E$ ayrıtı için $t = 1,..., n$ olmak üzere c_{ij}^t, i şehrinden j şehrine t zamanındaki ziyaretin maliyetidir. Burada amaç c_{ij}^t değerlerine göre en kısa turun bulunmasıdır. Tüm $(i, j) \in E$ ayrıtları için $c_{ij}^1 = c_{ij}^2 = ... = c_{ij}^n$ ise, problem gezgin satıcı problemi şeklinde formüle edilebilir.

3.3.1.5 Siyah ve Beyaz GSP (Black and White TSP)

Bu problemde, tepeler kümesi *V, S* ve *B* olmak üzere iki farklı kümeye ayrılır. *S* kümesinin elemanlarına siyah tepeler, *B* kümesinin elemanlarına ise beyaz kümeler denir. Problem aşağıdaki kısıtlar altında en kısa turu bulmayı amaçlar:

i) İki siyah tepe arasındaki beyaz tepe sayısı pozitif bir tamsayı olan *I* 'dan küçük olmalı.

ii) İki beyaz tepe arasındaki siyah tepe sayısı pozitif bir tamsayı olan *R* 'den küçük olmalı.

Bu problemin uygulama alanlarına örnek olarak telekomünikasyon ve hava yolu endüstrileri verilebilir.

Gezgin satıcı probleminin çeşitli uygulama senaryolarına bağlı olarak çok sayıda farklı varyasyonu daha bulunmaktadır. Bu problemler için 'The Travelling Salesman Problem and Its Variations' (Gutin and Punnen,2002) incelenebilir.

3.4 Çoklu Gezgin Satıcı Probleminin Tanımı ve Uygulama Alanları

Çoklu gezgin satıcı problemi gezgin satıcı probleminin çok sayıda günlük hayat problemini modellemek için kullanılabilecek karmaşık bir türevdir. Bu problemde gezgin satıcı probleminden farklı olarak *m* adet satıcı bulunmaktadır. Genel olarak çoklu gezgin satıcı problemi, *n* adet şehir kümesi verildiğinde, her bir şehrin bir satıcıya atanmak üzere *m* adet tura bölünmesi ve toplamda minimum maliyetin bulunmasıdır. Burada maliyet uzaklık ve zaman açısından tanımlanabilir (Hernandez-Saldana, 2010).

Çoklu gezgin satıcı probleminin birçok varyasyonu vardır. Problemin olası varyasyonları aşağıdaki gibidir (Kara and Bektas, 2006):

- Tek ve çoklu depolar: Tek depo olan çoklu gezgin satıcı probleminde tüm satıcılar turlara tek bir noktadan başlar ve tur sonunda o noktaya geri dönerler. Buna karşılık çoklu depolar olan çoklu gezgin satıcı probleminde her bir satıcı farklı bir noktaya yerleştirilir ve her satıcı tur bitiminde tura başladığı noktaya geri döner.

- Satıcı sayısı: Problemde satıcı sayısı sınırlı değişken veya başta sabitlenmiş bir değişken olabilir.

- Zaman penceresi: Problemin bu varyasyonunda belirli noktalar belirli zaman periyodu içerisinde ziyaret edilmek zorundadır. Çoklu gezgin satıcı probleminin bu varyasyonu önemli ve yaygın kullanılan bir problemdir. Bu problem türü okul servisleri, otobüs ve uçak güzergahlarının belirlenmesinde kullanılır.

Çoklu gezgin satıcı problemi rotalama uygulamalarının dışında üretim ve planlama alanlarında da sıklıkla kullanılmaktadır. Son zamanlarda problemin uyguladığı ilginç alanlardan biri de global navigasyon uydu sistemleridir (Saleh and Chelouah, 2004; Davendra, 2010; Park, 2001). Çoklu gezgin satıcı problemi ile ilgili kapsamlı bir literatür araştırması için 'The Multiple Traveling Salesman Problem: an Overview Of Formulations And Solution Procedures' (Bektas, 2006) incelenebilir.

3.5 Araç Rotalama Problemi

Çoklu gezgin satıcı probleminde, satıcılar kapasiteli araçlarla değiştirilirse problem *Araç Rotalama Problemine* (*Vehicle Routing Problem*) dönüşür. Araç rotalama problemi için optimizasyon kriteri genellikle gidilen toplam uzaklığın en aza indirilmesi olsa da, bazen de araç sayısı *m* minimize edilmelidir (Bruno et al., 2011). Araç rotalama problemi ilk olarak Dantzig and Ramser (1959) tarafından ortaya konmuştur. Ulaşım ve taşıma alanındaki problemlerin neredeyse tümü araç rotalama problemi ile modellenebilir. Problemin gerçek hayat uygulamaları birçok kısıtı beraberinde getirmektedir. Bu kısıtlara örnek olarak aşağıdaki kısıtlar verilebilir.

- Araçlar ile ilgili kısıtlar: Aracın kapasite sınırı, toplam zaman sınırı gibi kısıtlar vs.

- Müşteriler ile ilgili kısıtlar: Müşterilerin farklı ürünler talep etmesi, dağıtım için belirli zaman aralıkları vs.

- Diğer kısıtlar: Aynı araç için birden fazla tur yapılabilmesi, birden fazla depo vs.

Görüldüğü üzere araç rotalama problemi, çoklu gezgin satıcı problemine göre günlük hayat uygulamalarına çok daha uygun bir problemdir. Ancak araçların/satıcıların kapasite kısıtı göz önüne alınmazsa problem çoklu gezgin satıcı problemine dönüşmektedir. Bu tezde rotalama problemlerine teorik anlamda yaklaşıldığı için araç rotalama problemi yerine çoklu gezgin satıcı problemi çalışılmıştır.

3.6 Çinli Postacı Problemi

Gezgin satıcı problemi ve türevleri görüldüğü üzere şehirlerin/tepelerin ziyaret edilmesi gereken *tepe rotalama* (*vertex routing*) problemleridir. Ancak bazı durumlarda ayırtların ziyaret edilmesi gerekmektedir. Bu tür problemlere *ayrıt rotalama* (*edge routing*) problemi denir. Bu problemlerin kısıtsız hali çinli postacı problemidir. Problem bir *G* çizgesindeki tüm ayırtlardan minimum mesafe ile en az bir kere geçilmesini amaçlar. Çöp toplama ve yollardaki karların temizliği gibi uygulamalar bu problemin karşılaşıldığı günlük hayat uygulamalarına örnek olarak verilebilir. Bazı ayrıt rotalama problemlerinde, sadece bazı ayrıtların ziyaret edilmesi gerekmektedir. Bu tür problemler ise *Kırsal Postacı Problemi* (*Rural Postman Problem*) adıyla bilinmektedir (Laporte and Osman, 1995).

4. GEZGİN SATICI PROBLEMİ

Bu bölümde gezgin satıcı problemi ele alınmış, problemin matematiksel modeli verilmiş, problem için literatürde var olan çözüm yöntemlerinden bahsedilmiş ve son olarak problem için yeni bir sezgisel algoritma önerilmiştir.

4.1 Gezgin Satıcı Probleminin Matematiksel Modeli

Gezgin satıcı problemi, eğer simetrikse yönsüz $G=(V,E)$ tam çizgesi üzerinde $V=\{1,...,n\}$ tepe kümesi ve $E=\{(i,j):i,j\in V,i<j\}$ ayrıt kümesi ile gösterilebilir. $C=(c_{ij})$ E üzerinde tanımlanmış maliyet matrisi olmak üzere, eğer $\forall i,j,k$ için $c_{ij}\le c_{ik}+c_{kj}$ koşulunu sağlıyor ise C üçgen eşitsizliğini sağlamaktadır. Özellikle düzlemsel problemler için, tepe noktaları düzlemde $P_i=(X_i,Y_i)$ noktaları halinde verildiğinde maliyet matrisi $c_{ij}=\sqrt{(X_i-X_j)^2+(Y_i-Y_j)^2}$ Öklid uzaklığına göre hesaplanır. Eğer c_{ij}, G çizgesinde i'den j'ye en kısa yolun uzunluğu ise üçgen eşitsizliği sağlanır.

Literatürde birçok gezgin satıcı problemi modeli bulunmaktadır. Dantzig et al., (1954) tarafından geliştirilen model bu modeller arasında en çok kullanılan matematiksel modellerden bir tanesidir. Dantzig et al., (1954) tarafından önerilen matematiksel model aşağıdaki gibidir:

Amaç fonksiyonu:

$$enk \sum_{i<j} c_{ij}x_{ij} \tag{4.1}$$

Kısıtlar:

$$\sum_{i<k} x_{ik}+\sum_{j>k} x_{kj}=2 \qquad (k\in V) \tag{4.2}$$

$$\sum_{i,j\in S} x_{ij}\le |S|-1 \qquad (S\subset V, 3\le |S|\le n-3) \tag{4.3}$$

$$x_{ij}=0 \; veya \; 1 \qquad (i,j)\in E \tag{4.4}$$

Bu modelde, (4.2) kısıtı her bir tepenin derecesinin 2 olmasını, (4.3) kısıtı alt turların oluşmamasını ve (4.4) kısıtı ilgili ayrıtın tur içinde olup olmadığının belirlenmesini sağlar. (4.1) amaç fonksiyonunda kat edilen mesafenin minimum olmasını sağlar.

4.2 Gezgin Satıcı Probleminin NP-tamlığı

Bu bölümdeki teorem ve ispat Garey and Johnson'ın (1979) 'Computer and Intractability' isimli kitabından alınmıştır. Gezgin satıcı probleminin NP-tamlığı incelenmeden önce, ispatta kullanılan tanıma problemlerinin tanımları aşağıda verilmiştir:

Gezgin Satıcı Tanıma Problemi (GSTP):

Koşul: $C = (c_1,...,c_n)$ şehirler kümesi $\forall\, c_i, c_j \in C$ için bu iki şehir arasındaki mesafe $d(c_i, c_j) \in Z^+$ ve $B \in Z^+$ sınırı verilmiş olsun.

Soru: C kümesinin bütün şehirlerden geçen ve uzunluğu B'den büyük olmayan bir tur var mıdır? Yani C kümesinin elemanlarının öyle bir $\langle c_{\pi(1)}, c_{\pi(2)},..., c_{\pi(n)} \rangle$ sıralaması var mı ki, $\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}) \le B$ olsun ?

Hamilton Devresi Tanıma Problemi (HDTP):

Koşul: $G = (V, E)$ çizgesi verilmiş olsun.

Soru: G'de Hamilton Devresi var mı ?

Teorem 5.2.1: Gezgin satıcı problemi NP-tam'dır.

GSTP'nin NP-tam sınıfına ait olduğunu göstermek için dört tane koşulun sağlandığını gösterilmelidir.

i. $GSTP \in NP$.

Gezgin satıcı probleminin çözümü için tahmin edilen bir turun uzunluğunun B'den büyük olup olmadığının kontrolü için, turda bulunan ayrıtların maliyetlerinin toplanıp B'den büyük olup olmadığına bakmak yeterlidir. Bu kontrol polinom sınırlı olduğu için gezgin satıcı probleminin *NP* sınıfına ait olduğunu anlamak kolaydır.

ii. $HDTP \in NP-Tam$.

GSTP'ye bir dönüşüm sağlamak için $NP-Tam$ sınıfına ait olduğu bilinen bir problem belirlenmelidir. Bu problem *Hamilton Devresi Tanıma Problemi*dir.

iii. $f : HDTP \rightarrow GSTP$.

GSTP'nin NP-tamlığını göstermek için, Hamilton Devresi Tanıma Probleminden Gezgin Satıcı Tanıma Problemine bir dönüşüm gösterilmelidir. Açıktır ki bu iki problemde de tüm şehirlerden geçen bir devre aranmaktadır ve GSTP'de HDTP'den farklı olarak aranan turun bir B parametresinden büyük olmaması istenir. Gezgin satıcı problemi için olması gereken çizge tam çizgedir. Bu yüzden Hamilton Devresi için de verilen G çizgesinin $f(G)$ tam çizgesine dönüştürmek gerekir. Bu dönüşüm şu şekilde sağlanabilir: Hamilton devresi için ele alınan G çizgesinde eğer iki tepe arasında bir ayrıt varsa bu ağırlıklandırılarak 1 uzunluklu, eğer iki tepe arasında ayrıt yoksa bu ayrıt 2 uzunluklu olarak ifade edilip $f(G)$ çizgesi oluşturulur. GSTP'de aranan turun uzunluğunun sınırı olan B parametresi, şehir sayısı olan n'ye eşit olarak alınır.

iv. f, polinom sınırlıdır.

(iii). adımda bulunan f dönüşümünün polinom sınırlı olması gerekir. n tepeli bir tam çizgede $n(n-1)/2$ sayıda $d(v_i, v_j)$ ağırlıklarını hesaplamak için yalnız (v_i, v_j) ayrıtının E kümesinde olup olmadığını kontrol etmek gerekmektedir. Yani bu işlem $O(n^2)$ hesaplama karmaşıklığına sahiptir.

Aynı zamanda G çizgesinde Hamilton Devresinin sağlanması için, $f(G)$ çizgesinde tüm tepelerden geçen ve uzunluğu B'den büyük olmayan bir tur olması gerektiğini göstermek gerekmektedir. Varsayalım ki $\langle v_1, v_2, ..., v_n \rangle$ G'de Hamilton Devresi olsun. O halde $\langle v_1, v_2, ..., v_n \rangle$ $f(G)$'de bir tur olacaktır ve turun uzunluğu ise n'ye eşit olacaktır. Varsayalım ki, $\langle v_1, v_2, ..., v_n \rangle$ $f(G)$'de uzunluğu B'den büyük olmayan bir tur olsun. $f(G)$ çizgesinde şehirler arası mesafeler

1 veya 2 uzunluklu olduğundan ve turun uzunluğu n adet bu şekilde ayrıtlardan oluştuğu için, $B = m$ eşitliğinden turun her bir iki tepe arasındaki mesafesinin 1'e eşit olduğu anlaşılır. $f(G)$'nin tanımına göre (v_i, v_{i+1}), $1 \le i \le n$ ayrıtlarının G çizgesinin de ayrıtları olduğu çıkar ve böylece $\langle v_1, v_2, ..., v_n \rangle$ G'de Hamilton Devresidir.

4.3 Gezgin Satıcı Problemi için Çözüm Yöntemleri

Gezgin satıcı problemini tüm olası durumlar için polinom zamanda çözebilecek bir algoritma yoktur. Problemin boyutu arttıkça makul sürelerde çözümlere ulaşmak günümüz bilgisayarları ile çok uzun süreler gerektirmektedir. Bu yüzden gezgin satıcı problemini çözmek için genellikle sezgisel yöntemler kullanılmaktadır. Bu yöntemler en iyi çözümü bulmayı garanti etmeseler bile kaliteli sezgisel yöntemler makul zamanlarda en iyiye yakın sonuçlar vermektedir. Bununla birlikte literatürde problem için önerilmiş etkin kesin algoritmalar ve yaklaşım algoritmaları da bulunmaktadır.

4.3.1 Kesin Algoritmalar

Kesin algoritmalar, en kısa turu veren çözümü bulmayı garanti eden çözüm yöntemleridir. Bu yöntemler genellikle gezgin satıcı probleminin tamsayılı lineer programlama modelinden yararlanan yaklaşımlardır (Applegate et al., 1995). Gezgin satıcı problemi için önerilmiş en etkili kesin algoritmalar, kesen düzlemler veya *yüzey bulma* (*facet-finding*) algoritmalarıdır (Applegate et al., 1995; Grötschel and Holland, 1991; Padberg and Rinaldi, 1991). Bu algoritmalar yüksek bilgisayar gücüne ihtiyaç duymaktadır. Örneğin, 2392 şehirlik bir problemin kesin çözümü güçlü bir süper bilgisayar üzerinde 27 saatten uzun bir süreçte bulunabilmiştir (Grötchel and Holland, 1991). 7397 şehirlik bir problemin kesin çözümü ise çok geniş bir bilgisayar ağı üzerinde yaklaşık 3-4 yıllık CPU zamanı almıştır (Applegate et al., 1995). Günümüzde bilinen en iyi gezgin satıcı çözücüsü Applegate et al. (2003, 2006) tarafından geliştirilen Concorde'dur. Bu program, kısıtların ve değişkenlerin başlangıçta gevşetildiği ve çözüm süreci boyunca dinamik olarak oluşturulduğu '*branch-and-cut-and-price*' yöntemine dayanmaktadır. Concorde tarafından çözülen en büyük örnek büyük ölçekli entegrasyon uygulamalarında karşılaşılan 85900 tepeli bir problemdir (Applegate et al., 2009).

4.3.2 Yaklaşım Algoritmaları

Gezgin satıcı problemi NP-zor problem sınıfına ait olduğu için optimum çözüm bulmak büyük hesaplama karmaşıklığı getirmektedir. Bu yüzden zamandan kazanç sağlamak ve kısa sürelerde kaliteli çözümlerin bulunması için yaklaşım algoritmalarına veya sezgisel algoritmalara ihtiyaç duyulur. Yaklaşım algoritmaları optimum çözüme ulaşmayı garanti etmezler, ancak algoritmanın optimum değerden en fazla ne kadar uzak çözüm bulunabileceğini belirten bir garanti değere sahiptirler.

Gezgin satıcı problemi için en bilinen yaklaşım algoritmaları *Minimum Kapsayan Ağaca Dayalı Algoritma* ve *Christofides Algoritma*sıdır.

4.3.2.1 Minimum Kapsayan Ağaca Dayalı Algoritma

Minimum kapsayan ağaca dayalı algoritmada, ilk olarak çizgenin minimum kapsayan ağacı bulunur ve minimum kapsayan ağaçtaki her bir ayrıt çift yönlü kabul edilir. Daha sonra rastgele bir tepeden başlayarak ağaçtaki ayrıtlar üzerinde bir tur oluşturulmaya çalışılır. Minimum kapsayan ağaca dayalı algoritmanın garanti değeri 2'dir.

Algoritmanın temel adımları aşağıdaki verilmiştir (Papadimitriou and Steiglitz, 1998):

1. Tüm şehirleri içeren bir T minimum kapsayan ağacı oluştur.

2. Tüm $(i, j) \in T$ için yeni bir (i, j) ayrıtı ekle.

3. Herhangi bir tepeden başlayarak bir Euler turu bul ve GSP için uygun bir tur kalana kadar tekrar eden tepeleri sil.

4.3.2.2 Christofides Algoritması

Christofides algoritması minimum kapsayan ağaca dayalı algoritmanın geliştirilmesi ile oluşturulan bir algoritmadır. Algoritmanın garanti değeri $3/2$ 'dir. Bu algoritmanın zaman karmaşıklığı $O(n^3)$ 'tür. Algoritmanın temel adımları aşağıda verilmiştir (Papadimitriou and Steiglitz, 1998):

1. Tüm şehirleri içeren bir *T* minimum kapsayan ağacı oluştur.

2. Tek dereceye sahip olan tepeler arasında minimum ağırlıklı eşlemeyi oluştur ve bu eşleşmedeki ayrıtları *T*'ye ekle.

3. Herhangi bir tepeden başlayarak bir Euler turu bul ve GSP için uygun bir tur kalana kadar tekrar eden tepeleri sil.

Gezgin satıcı problemi için bilinen en iyi yaklaşım algoritması Arora (1998) tarafından önerilmiştir. $c > 1$ olmak üzere, algoritmanın garanti değeri $(1+1/c)$, zaman karmaşıklığı ise $O(n(\log_2 n)^{O(c)})$ 'dir (Matai et al., 2010).

4.3.3 Sezgisel Algoritmalar

Yaklaşık yöntemler içinde en çok kullanılan yöntemlerin başında *sezgisel* (heuristic) algoritmalar gelmektedir. Sezgisel algoritmalar, ele alınan problemin özelliğine göre tasarlanırlar. Bu algoritmalar her zaman en iyi sonucu vermeyi garanti etmezler veya her zaman aynı performans ile çalışmazlar. Sezgisel algoritmalar en kötü durum düşünüldüğünde en iyi çözümden çok uzak sonuçlar bulabilir ancak iyi sezgisel algoritmalar en iyi yaklaşım algoritmaların performanslarını da geçebilmektedir (Elmas, 2007). Bu algoritmalar NP-zor problemlerin çözümünü bulmak için kesin algoritmalara göre daha çok tercih edilmektedirler. Gezgin satıcı problemini için geliştirilen sezgisel algoritmalar iki ana başlık altında toplanabilir:

- Tur oluşturan sezgiseller
- Tur geliştiren sezgiseller

4.3.3.1 Tur Oluşturan Sezgiseller

Tur oluşturan sezgiseller çözüme adım adım tepeler ekleyerek bir tur oluştururlar. Ancak bu algoritmalar bir sonuç buldukları zaman bu sonucu geliştirmek için uğraşmazlar. Tur bulunduktan sonra algoritmaların çalışması sona erer. Bilinen tur oluşturan sezgisel algoritmalara *En Yakın Komşu Algoritması* (*Nearest Neighbor Algorithm*, NN) , *Açgözlü Algoritması (Greedy Algorithm), Ekleme Algoritması (Insertion Algorithm)* ve *Tasarruf Algoritmasını (Savings Algorithm)* örnek olarak gösterebiliriz (Johnson and Papadimitriou, 1985).

En Yakın Komşu Algoritması

Bu algoritma gezgin satıcı problemi için en basit ve en anlaşılır algoritmalardan birisidir. Bu yaklaşım herhangi bir tepe ile başlayıp, her bir adımda daha önce ziyaret edilmemiş en yakın tepenin bulunmasına dayanır. En yakın komşu algoritmasının zaman karmaşıklığı $O(n^2)$'dir. Algoritmanın temel adımları aşağıda verilmiştir (Johnson and McGeoch, 1997):

1. Rasgele bir şehir seç.

2. Önceden ziyaret edilmemiş en yakın şehri bul ve oraya git.

3. Ziyaret edilmemiş şehir kaldı mı? Cevap evetse Adım 2'ye git.

4. Başlangıç şehrine geri dön.

Açgözlü Algoritması

Açgözlü algoritmasında her defasında çözümde olmayan en kısa ayrıt seçilerek tura eklenir. Algoritmada ayrıtlar eklenirken eklenecek ayrıt eğer tura eklendiği zaman şehir sayısından az uzunlukta alt tur oluşturuyorsa veya ayrıt eklenen tepenin derecesi 2'den fazla oluyor ise o zaman bu ayrıt tura eklenmez ve bir sonraki kısa ayrıta geçilir. Açgözlü algoritmasının zaman karmaşıklığı $O(n^2 \log_2(n))$'dir. Algoritmanın temel adımları aşağıdaki verilmiştir (Johnson and McGeoch, 1997):

1. Ayrıtları küçükten büyüğe doğru sırala.

2. En kısa ayrıtı seç ve eğer tura eklendiği zaman alt tur oluşmuyorsa o ayrıtı tura ekle.

3. Turda *n* tane ayrıt var mı? Cevap hayırsa Adım 2'ye git

Ekleme Algoritması

Ekleme sezgiseli oldukça anlaşılır algoritmalardır ve birçok çeşitli varyasyonu vardır. En basit hali ile bu algoritma ilk önce tepeler kümesinin bir alt kümesi ile oluşturulmuş tur ile başlayıp sonrasında bazı kısıtlara göre kalan tepeleri tura ekler. Başlangıç alt turu genellikle üçgendir. Ancak bazı ekleme sezgiselleri ise tek bir ayrıt ile algoritmaya başlar. En bilinen ekleme sezgiseli olan en yakın ekleme sezgiselinin zaman karmaşıklığı $O(n^2)$'dir. Algoritmanın temel adımları aşağıda verilmiştir (Matai et al., 2010):

1. En kısa uzunluklu ayrıtı seç ve alt tur oluştur.

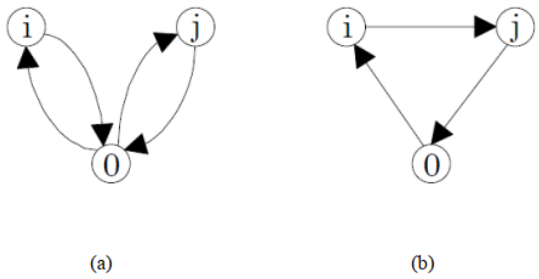
2. Alt turda bulunmayan ve alt turdaki şehirlerden birine en yakın olan bir şehir seç.

3. Eklenecek şehir ile uç noktaları arasındaki uzaklıkları toplamının en az olduğu ayrıtı bul.

4. Eklenecek şehir kalmayıncaya kadar Adım 3'e git.

Tasarruf algoritması

Tasarruf algoritması Clarke and Wright (1964) tarafından araç rotalama problemini çözmek için geliştirilmiş sezgisel bir yöntemdir. Temel olarak tasarruf yöntemi iki rotanın bir rota şeklinde birleştirilmesine ve maliyetten kazanç sağlanmasına dayanır. Şekil 4.1'de görüldüğü gibi 0 başlangıç tepesi olmak üzere başta 2 tane olan rota tasarruf yöntemi ile en iyi kazanç sağlanacak şekilde 1 rota olarak ifade edilir.



Şekil 4.1 Tasarruf algoritmasının çalışma prensibi

i ve j tepeleri arasındaki maliyet c_{ij} olmak üzere şekil 4.1 (a)'daki toplam ulaşım maliyeti D_a aşağıdaki şekilde hesaplanır:

$$D_a = c_{0i} + c_{i0} + c_{0j} + c_{j0}$$

Aynı şekilde şekil 4.1 (b)'deki toplam ulaşım maliyeti:

$$D_b = c_{0i} + c_{ij} + c_{j0}$$

şeklindedir. 2 rotanın 1 rotaya dönüştürülmesindeki kazanç ise:

$$S_{ij} = D_a - D_b$$

formülü ile ifade edilir (Clarke and Wright, 1964).

Algoritmanın adımları aşağıdaki şekildedir (Johnson and McGeoch, 2002):

1. Keyfi seçilmiş başlangıç tepesinden diğer tüm tepelere iki tane ayrıtı bulunan çok katlı bir çizge üzerinde keyfi bir tur oluştur.

2. En çok kazanç sağlayan tepe ikililerini seçerek merkez olmayan iki tepeyi tek bir ayrıt ile birbirine bağlayarak kısaltmalar yap.

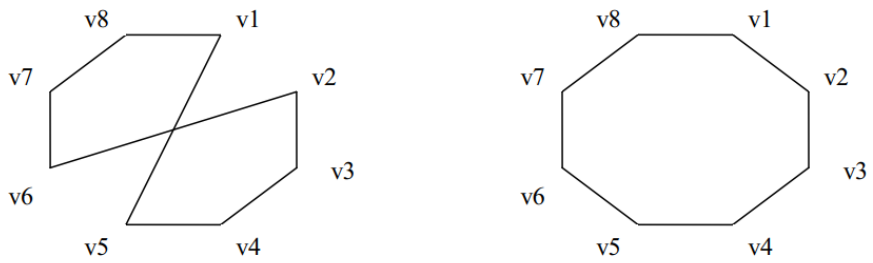
3. Tepe sayısından iki eksik ayrıt eklendiği zaman, merkez tepeden oluşturulmuş yolun ucundaki tepeler birer ayrıt ekleyerek turu tamamla.

4.3.3.2 Tur Geliştiren Sezgiseller

Bu sezgiseller verilen oluşturulmuş bir tur üzerinde çeşitli oynamalar yaparak turu geliştirmeyi amaçlarlar. Bu algoritmalara örnek olarak 2-opt, 3-opt, k-opt ve Lin-Kernighan gibi yerel optimizasyon algoritmalarını verebiliriz. Bu yöntemlerin performansı genel olarak tur oluşturulurken kullanılan sezgiselin performansına bağlıdır.

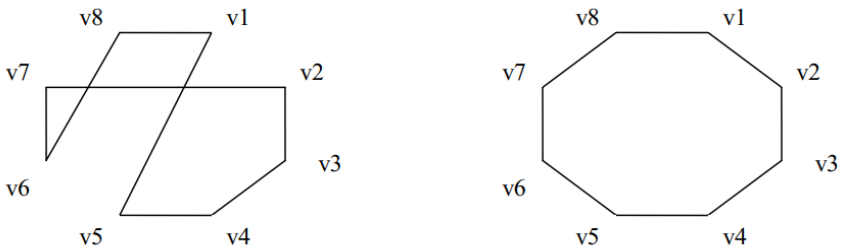
2-opt, 3opt Algoritmaları

2-opt algoritması, turda bulunan iki ayrıtı silip bunlar yerine iki yeni ayrıt ekleyerek yeni bir tur oluşturur (Şekil 4.2). Bu hareketi sadece oluşacak olan yeni tur eğer öncekine göre daha iyiyse yapar. Bu işlem 2-opt adımları ile değiştirilecek ayrıtlar kalmayıncaya kadar devam eder.



Şekil 4.2 2-opt yöntemi

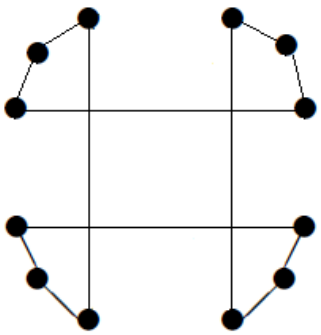
3-opt algoritması da 2-opt algoritması ile benzer şekilde çalışır. Ancak 3-opt yönteminde iki ayrıt yerine üç ayrıt değiştirilir (Şekil 4.3). Bu yöntem 3-opt adımları ile tur daha fazla iyileştirilemeyecek duruma gelene kadar devam edilir (Johnson and McGeoch, 1997).



Şekil 4.3 3-opt yöntemi

k-opt algoritması

Tur oluşturma algoritmaları ile üretilmiş turu iyileştirmek için k-opt yöntemi uygulanabilir. Ancak $k > 3$ değerler için bu yöntemi uygulamak zaman karmaşıklığı açısından daha fazladır. Temel olarak 4-opt yöntemi sıklıkla kullanılan bir yöntemdir ve bu yöntem *çapraz köprüler (the crossing bridges)* olarak adlandırılır (Şekil 4.4). 4-opt yöntemi art arda 2-opt kullanılarak oluşturulamaz (Johnson and McGeoch, 1997).



Şekil 4.4 Çift köprü hareketi

Lin-Kernighan Algoritması

Lin-Kernighan algoritması, 3-opt'un genelleştirilmesi üzerine kurdukları çalışma ile önerilmiştir (Lin and Kernighan, 1973). Bu algoritmada, her iterasyonda uygun bir k değerini belirlenir. Ayrıca, k-opt'taki gibi ilk iyileştirmeyi değil en büyük iyileştirmeyi hedefler. Yani eldeki turdan daha iyi bir tur bulunduğu zaman değişim yapmak yerine tüm olası iyileştirmelerden en iyisini elde ettiği zaman değişimi yapar. Ancak k değerinin belirlenmesi bu algoritmayı karmaşık hale getirmektedir. Bu algoritmanın yaklaşık olarak zaman karmaşıklığı $O(n^{2.2})$'dir (Lin and Kernighan, 1973).

4.3.4 Metasezgisel Algoritmalar

Metasezgisel yöntemler zor optimizasyon problemlerinin çözümü için kullanılan belli bir yöntemle yaratılan aday çözümü yada çözümleri adım adım iyileştirmeye çalışan hesaplama teknikleridir. Gezgin satıcı problemi için en etkin çözüm yöntemlerinden biri de metasezgisel algoritmalardır. Bu algoritmalar, karmaşık komşuluk arama teknikleri, farklı hafıza yapıları ve çözümlerin rekombinasyonu gibi farklı yöntemleri kombine etmektedir. Metasezgisel algoritmalar bir ya da fazla ayarlanabilir parametrelere sahiptir. Kaliteli çözümlere ulaşmak için problemlerin türüne göre bu parametrelerin uygun bir şekilde ayarlanması gerekmektedir. Bu algoritmaların en büyük dezavantajı uzun çalışma sürelerine ihtiyaç duymasıdır (Anbuudayasankar et al., 2016). Literatürde gezgin satıcı problemini çözmek için kullanılan birçok metasezgisel algoritma mevcuttur. Bunlara örnek olarak; tabu araması, genetik algoritmalar, *benzetimli tavlama* (*simulated annealing*) ve karınca kolonisi optimizasyonu gibi yapay zeka yöntemlerini verilebilir (Applegate et al., 2006).

4.3.4.1 Tabu Arama

Tabu arama algoritması gezgin satıcı probleminde daha iyi çözümlere ulaşmak için genellikle 2-opt değişim yöntemini kullanmaktadır. 2-opt ve 3-opt gibi basit komşuluk arama algoritmaları yerel optimumda takılıp kalabilirler. Ancak bu sorun tabu arama algoritması ile kolaylıkla aşılabilmektedir. Bu sorundan kurtulmak için tabu araması kötü değişimler ile elde edilen kötü sonuçları içeren bir yasak listesi kullanır.

Tabu listesini oluşturmak için ise birçok yöntem vardır. Bu algoritmanın en büyük dezavantajı çalışma zamanıdır. Gezgin satıcı problemi için en çok uygulanan tabu arama yönteminin çalışma zamanı $O(n^3)$ 'tür. Bu zaman ise 2-opt yöntemine göre daha yavaştır (Johnson and McGeoch, 1997).

4.3.4.2 Genetik Algoritmalar

Genetik algoritmalar gezgin satıcı problemi için uygulanırken çaprazlama ve mutasyon işlemleri bilinen şekilde yapılır. Bir bireyin uygunluk fonksiyonu ise o çözümün uzunluğu ile hesaplanır. Ayrıca çözümün uzunluğu çaprazlama ve mutasyon işlemleri için de kullanılmaktadır (Johnson and McGeoch, 1997).

4.3.4.3 Benzetimli Tavlama

Benzetimli tavlama yöntemi gezgin satıcı problemi için başarılı bir şekilde uygulanan ve adapte olan bir yöntemdir. Bu yöntem temel olarak tabu araması gibi rasgele yerel aramalar yapan bir yöntemdir. Ancak benzetimli tavlama da çözümü kötüleştiren değişimlere de izin verilmektedir (Johnson and McGeoch, 1997). Benzetimli tavlama yönteminde, algoritmanın çalışma zamanı arttırılarak daha iyi sonuçlar elde edilebilir. Bu yöntemin Lin Kernighan algoritması ile benzer kıyalanabilecek sonuçlar bulduğu gösterilmiştir. Genellikle 2-opt kullanıldığı için algoritmanın zaman karmaşıklığı $O(n^2)$ olarak düşünülebilir (Johnson and McGeoch, 1997).

4.3.4.4 Karınca Kolonisi Optimizasyonu

Karınca kolonisi optimizasyonu kolaylıkla gezgin satıcı problemi için uygulanabilir. Hatta bu yöntem ile küçük boyutlu problemlerde optimum çözümler bulunmaktadır. Gezgin satıcı probleminim çözümünde, ilk olarak bir grup karınca ile başlanır ve bu karıncalar rastgele şehirlere yerleştirilir. Daha sonra karıncaları daha önceden ziyaret edilmemiş diğer şehirlere doğru hareket ettirilerek, en kısa turu bulan karıncaya bulduğu turun uzunluğuna göre ters orantılı olarak feromon izi bırakmasını sağlanır. Bu feromon miktarı karıncanın ilerleyeceği şehri seçerken dikkate alınır ve karıncalar feromon madde miktarının daha fazla olduğu yerlere yönlendirilir. Bu işlem yeterli uzunlukta bir tur bulunana kadar tekrarlanır (Dorigo and Gambardella, 1997).

Literatürde gezgin satıcı probleminin çözümünde kullanılan çok sayıda farklı metasezgisel algoritma bulunmaktadır. *Yapay Arı Kolonisi Algoritması* (*Artificial Bee Colony Algorithm*), *Yapay Sinir Ağları* (*Artificial Neural Network*), *Yapay Bağışıklık Sistemi* (*Artificial Immune Recognition System*), *Parçacık Sürü Optimizasyonu* (*Particle Swarm Optimization*), *Tepe Tırmanma* (*Hill Climbing*), *Kanguru Algoritması* (*Kangaroo Algorithm*), *Memetik Algoritma* (*Memetic Algorithm*) gibi algoritmalar bunlara örnek olarak verilebilir. Bu algoritmalar ile ilgili detaylı bilgiler için "Models for Practical Routing Problems in Logistics" (Anbuudayasankar et al., 2016) adlı kitap ve "Gezgin Satıcı Problemlerinin Metasezgiseller ile Çözümü" (Kuzu vd., 2004) adlı makale incelebilir.

4.4 Gezgin Satıcı Problemi için Yeni Bir Sezgisel Algoritma

Bu bölümde gezgin satıcı problemi için yeni bir iteratif sezgisel algoritma önerilmiştir. Önerilen algoritma başlangıç çözümleri üzerinden ayrıtlara uygunluk değeri atayıp, daha sonra iteratif bir şekilde bu değerlere göre çözümü iyileştirmeye çalışmaktadır. Önerilen algoritmada, başlangıç çözümleri bulunurken hem açgözlü algoritma hem de en yakın komşu algoritması kullanılarak iki yöntemin de avantajlarından faydanılmış ve çözümlerde çeşitlilik sağlanmıştır. Uygunluk değerleri oluşturulduktan sonra, bu değerlere göre açgözlü algoritma çalıştırılıp, her bir iterasyonda uygunluk değerleri güncellenerek çözümlerin iyileştirilmesi amaçlanmıştır. Algoritmanın detaylı anlatımı aşağıda verilmiştir:

$G = (V, E)$ çizgesinde $V = \{v_1, v_2, \ldots, v_n\}$ tepeler kümesi, $E = \{e_1, e_2, \ldots, e_m\}$ ayrıtlar kümesi ve $|V| = n,\ |E| = m$ olsun.

dd en uzun ayrıtı, $u_j = dd / e_j$, $(j = 1,2,\ldots,m)$ e_j ayrıtının uygunluk değerini, X bulunmuş en iyi çözümü ve $F(X)$ amaç fonksiyonun değerini göstermek üzere, algoritmanın temel adımları aşağıdaki gibidir:

1. $iter \leftarrow 1$

2. Tüm $e_j \in E$ ayrıtları küçükten büyüğe sırala

3. e_j değerlerine göre Açgözlü Algoritmasını uygula ve X , F ve u_j değerlerini güncelle.

$X = X^{(1)}$, $F = F^{(1)}$ Tüm $(i, j) \in E$ ayrıtları küçükten büyüğe sırala

$$u_j^1 = \begin{cases} u_j^{iter} + 1, & \text{Eğer, } e_j \text{ ayrıtı } X^{(1)} \text{ çözümünde var ise,} \\ u_j^{iter}, & \text{Aksi durumda} \end{cases}$$

4. $iter \leftarrow iter + 1$

5. En küçük $e_j \in E$ ayrıtının uç tepelerinden başlayarak İki uçtan En Yakın Komşu Algoritmasını uygula ve $t = F / F^{iter}$ olmak üzere X , F ve u_j değerlerini güncelle.

5.1. Eğer $t > 1$ ise $X = X^{(iter)}$ ve $F = F^{(iter)}$

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ çözümünde var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

5.2. Eğer $t \leq 1$ ise

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ çözümünde var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

6. $iter \leftarrow iter + 1$

7. Adım 5'te çözüme son giren $e_j \in E$ ayrıtının herhangi bir uç tepesinden başlayarak İki uçtan En Yakın Komşu Algoritmasını uygula ve $t = F / F^{iter}$ olmak üzere X , F ve u_j değerlerini güncelle.

7.1. Eğer $t > 1$ ise $X = X^{(iter)}$ ve $F = F^{(iter)}$

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ çözümünde var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

7.2. Eğer $t \leq 1$ ise

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ çözümünde var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

8. $iter \leftarrow iter + 1$

9. Adım 5'te çözüme son giren $e_j \in E$ ayrıtının diğer uç tepesinden başlayarak İki uçtan En Yakın Komşu Algoritmasını uygula ve $t = F / F^{iter}$ olmak üzere X , F ve u_j değerlerini güncelle.

9.1. Eğer $t > 1$ ise $X = X^{(iter)}$ ve $F = F^{(iter)}$

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ çözümünde var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

9.2. Eğer $t \leq 1$ ise

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ çözümünde var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

10. $iter \leftarrow iter + 1$

11. Tüm $e_j \in E$ ayrıtları u_j değerine göre küçükten büyüğe sırala

12. u_j değerlerine göre Açgözlü Algoritmasını uygula ve t, X , F ve u_j değerlerini güncelle.

12.1. Eğer $t = 1$ ise

$$u_j^{iter} = \begin{cases} u_j^{iter-1} - 1, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ ve } X \text{ de var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

12.2. Eğer $t > 1$ ise $X = X^{(iter)}$ ve $F = F^{(iter)}$

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ de var ve } X \text{ de yok ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

12.3. Eğer $t < 1$ ise

$$u_j^{iter} = \begin{cases} u_j^{iter-1} + t, & \text{Eğer, } e_j \text{ ayrıtı } X^{(iter)} \text{ de yok ve } X \text{ de var ise,} \\ u_j^{iter-1}, & \text{Aksi durumda} \end{cases}$$

13. Eğer Tüm $iter < iterasyon$ ise Adım 10'a dön.

Önerilen algoritma belirlenen bir *iterasyon* değeri kadar çalıştıktan sonra sonlanmaktadır. Algoritmayı durdurma koşulu istenildiği takdirde en iyi çözümün belli bir adımda değişmemesi gibi farklı parametrelerle de değiştirilebilir.

4.4.1 GSP için Önerilen Algoritmanın Zaman Karmaşıklığı

Önerilen algoritmanın zaman karmaşıklığı şu şekilde hesaplanabilir: $G = (V, E)$ bir tam çizge $n = |V|$ ve $m = |E|$ olsun. Görüldüğü gibi algoritmada sırasıyla 1 kez açgözlü algoritma, 3 kez en yakın komşu algoritması ve k iterasyon sayısı olmak üzere $k-4$ kez açgözlü algoritma uygulanmaktadır. Sonuç olarak algoritma için karmaşıklığı $O(n^2 \lg_2 n + 3 \times n^2 + k \times n^2 \lg_2 n)$ olarak hesaplanabilir. Dolasıyla algoritmanın zaman karmaşıklığı $O(kn^2 \lg_2 n)$ ' tür.

4.4.2 GSP için Önerilen Algoritmanın Hesaplama Denemeleri

Önerilen algoritma C dilinde kodlanmıştır. Hesaplama denemeleri 4.00 GHz Intel Core i7-6700 CPU işlemci ve 8 GB RAM'e sahip işletim sistemi 64-bit Windows olan bir bilgisayar üzerinde yapılmıştır. Algoritmanın çalışma zamanı tüm algoritmalarda ortak olduğu için çizge okumayı ve uzaklık matrisinin oluşturulmasını içermemektedir.

Önerilen algoritmanın performansı TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/) kütüphanesindeki örnekler üzerinde test edilmiştir. Algoritma açgözlü ve en yakın komşu algoritması temelli olduğu için sonuçlar bu algoritmalar ile kıyaslanmış ve bu yöntemleri beraber kullanmanın getirdiği avantajların/dezavantajları tespit edilmeye çalışılmıştır. Ayrıca önerilen algoritma literatürde gezgin satıcı problemi için önerilmiş benzer bir algoritma olan hibrid sezgisel algoritma (Kizilates and Nuriyeva, 2013) ile kıyaslanmıştır.

Çizelge 4.1'de önerilen iteratif sezgisel algoritma için hesaplama denemeleri verilmiştir. Çizelgede her bir satır için, ilk sütun çizgenin ismini, ikinci sütun çizgenin boyutunu, üçüncü sütun en kısa turun uzunluğunu ve geri kalan sütunlarda sırasıyla en yakın komşu algoritmasının, aç gözlü algoritmanın, hibrid algoritmanın ve önerilen algoritmanın bulduğu sonuçları ve programların çalışma süreleri bulunmaktadır. Açgözlü algoritama, en yakın komşu algoritması ve hibrid algoritması için olan sonuçlar 'A New Hybrid Heuristic Algorithm for Solving TSP' (Kizilates and Nuriyeva, 2013) adlı makaleden alınmıştır. Hesaplama denemelerinde önerilen sezgisel algoritma için iterasyon değeri $10 \times n$ alınmıştır.

Çizelge 4.1 Önerilen Algoritmanın Kütüphane Örnekleri Üzerindeki Sonuçları

G	**Optimal**	**NN**	**Açgözlü Algoritma**	**Hibrid Algoritma**	**Önerilen Algoritma**
ulysses22	75,665	86,905 0,000	89,436 0,010	79,157 0,014	79,759 0,046
bayg29	9074,148	9964,781 0,000	9886,208 0,015	9413,957 0,054	9650,891 0,078
att48	33523,708	39236,885 0,000	38849,621 0,125	37737,693 0,662	37154,209 0,562
eil51	429,983	505,774 0,016	481,518 0,125	487,448 0,813	470,330 1,187
berlin52	7544,365	8182,192 0,000	9954,062 0,281	7791,375 0,887	7959,568 1,234
st70	678,597	761,689 0,000	746,044 0,485	754,760 3,893	735,852 4,811
eil76	545,387	612,656 0,016	617,131 0,672	599,682 5,858	611,418 8,091
gr96	512,309	603,302 0,015	580,101 1,609	592,573 17,938	563,539 20,181
rat99	1211	1369,535 0,016	1528,308 1,875	1328,376 20,498	1309,027 21,129
kroA100	21236,951	24698,497 0,016	24197,285 1,937	24447,780 22,392	23438,084 27,445
kroB100	22141	25882,973 0,015	25815,214 2,469	25464,985 23,158	24296,193 34,147
kroC100	20750,762	23566,403 0,016	25313,671 2,610	22984,858 22,806	22686,925 35,679
kroD100	21294,29	24855,799 0,016	24631,533 2,359	24224,136 22,554	24395,024 35,476
kroE100	22068	24907,022 0,016	24420,355 2,609	24411,041 23,018	24377,734 34,162
rd100	7910,396	9427,333 0,015	8702,605 2,922	9321,775 22,891	8492,701 34,570
eil101	642,309	736,368 0,015	789,112 2,609	731,943 23,901	731,161 29,101
lin105	14382,995	16939,441 0,015	16479,785 3,187	16068,265 27,923	15275,108 25,837

Çizelge 4.1'den anlaşılacağı gibi önerilen algoritma en yakın komşu ve açgözlü algoritmalardan çok daha iyi sonuçlar bulmaktadır. Ayrıca önerilen algoritma ile benzer şekilde en yakın komşu ve açgözlü algoritmalarının hibridi olan algoritmadan da örneklerin çoğunda daha iyi sonuçlar bulunduğu görülmektedir.

5. ÇOKLU GEZGİN SATICI PROBLEMİ

Bu bölümde çoklu gezgin satıcı problemi ele alınmış, problemin matematiksel modeli verilmiş, problem için literatürde var olan çözüm yöntemlerinden bahsedilmiş ve son olarak problem için yeni bir sezgisel algoritma önerilmiştir.

5.1 Çoklu Gezgin Satıcı Probleminin Matematiksel Modeli

Çoklu gezgin satıcı problemi, V tepeler kümesi ve E ayrıtlar kümesini, $C = (c_{ij})$, E kümesi üzerinde tanımlanmış maliyet matrisini göstermek üzere bir $G = (V, E)$ çizgesi üzerinde tanımlanabilir. Literatürde çoklu gezgin satıcı problemi için önerilen birçok matematiksel mode bulunmaktadır. Bu modeller arasında en yayın olarak kullanılanlardan olan *atama tabanlı* matematiksel model aşağıdaki gibidir: Modelde ikili değişken olan x_{ij} eğer (i, j) tur içinde kullanıldıysa 1, aksi halde 0 olacak şekilde tanımlanır (Matai et. al, 2010).

Amaç fonksiyonu:

$$enk \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \qquad (5.1)$$

Kısıtlar:

$$\sum_{j=2}^{n} x_{1j} = m \qquad (5.2)$$

$$\sum_{j=2}^{n} x_{j1} = m \qquad (5.3)$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 2,..., n \qquad (5.4)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 2,..., n \qquad (5.5)$$

$$\sum_{i\in S}\sum_{j\in S} x_{ij} \le |S| - 1, \qquad \forall S \subseteq V - \{1\}, \qquad S \neq \phi \qquad (5.6)$$

$$x_{ij} = 0 \; veya \; 1 \qquad (i, j) \in E \qquad (5.7)$$

Bu modelde (5.4), (5.5) ve (5.7) kısıtları atama problemi kısıtlarını, (5.2) ve (5.3) kısıtları her bir satıcının başlangıç noktasına geri dönmesini, sağlar. Kısıt (5.6) alt turların oluşmasını önlemek için kullanılmıştır. (5.1) amaç fonksiyonu da kat edilen mesafenin minimum olmasını sağlar.

5.2 Çoklu Gezgin Satıcı Problemi için Çözüm Yöntemleri

5.2.1 Kesin Algoritmalar

Çoklu gezgin satıcı problemini, herhangi bir GSP dönüşü yapmadan ilk çözüm önerisi Laporte and Nobert (1980) tarafından geliştirilmiştir. Önerilen yöntem bazı kısıtların gevşetilmesine dayanmaktadır. Büyük boyutlardaki çoklu gezgin satıcı problemi için ilk optimum çözüm bulma girişimi Gavish and Srikanth (1986) tarafından önerilen bir *branch and bound* yöntemidir. Gromicho et al., (1992) problem için bir başka kesin algoritma önermiştir. Çoklu gezgin satıcı problemi için çözülmüş en büyük örnek satıcı sayısı 2 den 12'e kadar olan 120 tepeli bir problemdir (Matai et al., 2010).

5.2.2 Sezgisel Algoritmalar

Çoklu gezgin satıcı probleminin karmaşıklığı dolayısıyla, gerçekçi büyüklüklerdeki problemlerin çözülmesi için sezgisel algoritmaların kullanılması gerekmektedir. Problem için önerilen ilk sezgisellerden biri (Russell, 1977), Lin Kernighan algoritmasının genişletilmiş bir varyasyondur. Potvin et al., (1989) değişim prosedürü tabanlı bir başka sezgisel algoritma önermiştir. Fogel (1990) evrimsel hesaplama kullanarak paralel hesaplama yaklaşımı önermiştir. Bu çalışmalarla beraber çoklu gezgin satıcı problemi için birçok metazsezgisel yaklaşımda çalışılmıştır. Problem için ilk genetik algoritma Zhang et. al (1999) tarafından çalışılmıştır. Ayrıca, Yu et al., (2002) yol planlamadaki çoklu gezgin satıcı probleminin çözümü için genetik algoritmaları kullanmıştır. Ryan et al., (1988) zaman pencereli çoklu gezgin satıcının çözümü için tabu araması yöntemini kullanmıştır. Song et al., (2003) problem için genişletilmiş bir benzetimli tavlama yöntemi önermiştir.

5.3 Çoklu Gezgin Satıcı Problemi için Yeni Bir Sezgisel Algoritma

Bu bölümde çoklu gezgin satıcı problemi için yeni bir sezgisel algoritma önerilmiştir. Önerilen algoritma açgözlü yöntem tabanlı basit bir çözüm yaklaşımıdır. Algoritma çalışma prensiplerinden dolayı '*Çift-Yönlü Açgözlü Algoritma'* (*Two-Way Greedy Algorithm*, TWGA) olarak isimlendirilmiştir.

Önerilen yöntem ilk etapta bir başlangıç çizgesi G^* oluşturmaktadır. İkinci etapta G^* çizgesinden gereksiz ayrıtlar çıkarılarak, bağlantısız yollar ve turlar elde edilir. Son adımda, elde edilen yollar ve turlar G^* gezgin satıcı sayısına göre ayarlanır. Algoritmanın sözde-kodu aşağıda verilmiştir.

1. $G^* \leftarrow 0,\ derece() \leftarrow 0$
2. Tüm $(i, j) \in E$ ayrıtları küçükten büyüğe sırala
3. Her bir $(i, j) \in E$ ayıtı için

   Eğer (derece(i) < 2 veya derece(j) < 2) ise

   e_{ij} ayrıtını G^* çizgesine ekle

   derece (i) ve derece(j) değerlerini güncelle
4. Tüm $(i, j) \in E$ ayrıtları büyükten küçüğe sırala
5. Her bir $(i, j) \in E$ ayıtı için

   Eğer (derece(i) > 2 veya derece(j) > 2) ise

   e_{ij} ayrıtını G^* çizgesine çıkar

   derece(i) ve derece(j) değerlerini güncelle
6. Minimum maliyet ile G^* çizgesindeki yolları/çevreleri böl veya birleştir

Algoritmanın 6. Adımında m gezgin satıcı m adet tur ayarlanırken, her bir adımda açgözlü yaklaşımla, o adımdaki en az maliyetli seçim yapılmaktadır. Eğer algoritmada $m = 1$ alınırsa, TWGA gezgin satıcı problemi için de çözüm bulmaktadır. Önerilen algoritmanın etkinliğinin gösterilmesi için TWGA gezgin satıcı problemleri üzerinde de test edilmiştir.

5.3.1 Çoklu GSP için Önerilen Algoritmanın Zaman Karmaşıklığı

Önerilen algoritmanın zaman karmaşıklığı şu şekilde hesaplanabilir: $G=(V,E)$ bir tam çizge $n=|V|$ ve $m=|E|$ olsun. Çizgedeki tüm ayrıtların sıralanması maliyeti $O(m\lg m)$'dir. Dolayısıyla *2-3* ve *4-5* adımları için gereken toplam maliyet $2\times O(m\lg m)$'dir. 6. adımda p yol/çevre sayısı ve k en uzun yoldaki/çevredeki tepe sayısı olmak üzere, yollar/çevreler arasında en az maliyetli birleşimin bulunması $O(kp\times kp)$, en az maliyetli bölünmenin bulunması $O(kp)$ işlem gerektirir. p ve k'nın üst sınırı n olduğu için *6*. Adım için en kötü durum karmaşıklığı $O(n^4)$ olarak ele alınabilir.

Sonuç olarak algoritmanın genel karmaşıklığı $O(m\lg m+m\lg m+kp\times kp)$ 'den dolayı $O(n^4)$' tür. Fakat pratik uygulamalarda p ve k değerleri n değerine göre çok daha küçük değerler alacağı için, örnekler üzerindeki algoritmanın çalışma zamanı çok daha kısa olacaktır.

5.3.2 Çoklu GSP için Önerilen Algoritmanın Hesapalama Denemeleri

Önerilen algoritma C dilinde kodlanmıştır. Hesaplama denemeleri 4.00 GHz Intel Core i7-6700 CPU işlemci ve 8 GB RAM'e sahip işletim sistemi 64-bit Windows olan bir bilgisayar üzerinde yapılmıştır. Algoritmanın çalışma zamanı tüm algoritmalarda ortak olduğu için çizge okumayı ve uzaklık matrisinin oluşturulmasını içermemektedir.

Literatürdeki çoklu gezgin satıcı problemi için bir kütüphane mevcut olmadığından, önerilen algoritmanın performansının analiz edilmesi için TSPLIB'deki (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/) örnekler kullanılmıştır. Ancak bu örneklerin optimum sonuçları çoklu gezgin satıcı için bilinmediğinden algoritmanın verimliliği, literatürde var olan 'Multiple Depots and Closed Paths (MDCP)' algoritması ile karşılaştırılmıştır (Hou and Liu, 2012). MDCP algoritması ile tutarlı bir kıyaslama olması için gezgin satıcı sayısı MDCP algoritması ile benzer şekilde 2 ile 10 arasında alınmıştır.

Çizelge 5.1'de TWGA ile MDCP'nin performansları kıyaslanmıştır. Çizeldege her bir satır için, ilk sütun çizgenin ismini, ikinci sütun çizgenin boyutunu, üçüncü sütun $m=1$ için optimum turun uzunluğunu ve geri kalan sütunlarda m gezgin satıcı sayısına göre sırasıyla MDCP ve TWGA'nın bulduğu sonuçlar bulunmaktadır.

Çizelge 5.1 Önerilen Algoritmanın Çoklu GSP için Hesaplama Denemeleri

			m=2		m=3		m=4		m=5	
	n	**optimum**	**MDCP**	**TWGA**	**MDCP**	**TWGA**	**MDCP**	**TWGA**	**MDCP**	**TWGA**
Eil51	51	426	417	432	451	427	447	425	445	423
St70	70	675	715	697	648	686	747	677	737	669
Eil76	76	538	569	570	548	567	566	565	560	563
Rat99	99	1211	1477	1266	1421	1257	1323	1250	1312	1246
KroA100	100	21282	22189	23623	22137	23245	23182	23038	23187	22670
KroB100	100	22141	22494	23569	22726	23313	24160	23064	23776	22820
Eil101	101	629	620	676	634	672	625	669	629	666
Pr107	107	44303	37371	37467	36780	36639	36232	35811	35572	34983
KroB150	150	26130	27778	27708	26953	27455	26859	27236	27702	27054
KroA200	200	29368	47147	31285	48168	30973	48228	30706	47445	30629
Tsp225	225	3916	4156	4087	3997	4067	3973	4052	4003	4037
A280	280	2579	2723	2737	2884	2731	3009	2723	3050	2715
Lin318	318	42029	49377	45384	48555	45023	47915	44662	46459	44301
	m=6		**m=7**		**m=8**		**m=9**		**m=10**	
	MDCP	**TWGA**	**MDCP**	**TWGA**	**MDCP**	**TWGA**	**MDCP**	**TWGA**	**MDCP**	**TWGA**
Eil51	440	425	454	425	466	429	485	450	481	492
St70	746	672	736	664	722	657	692	651	681	651
Eil76	572	563	571	563	575	565	598	568	600	573
Rat99	1277	1246	1262	1249	1236	1252	1221	1255	1331	1264
KroA100	22597	22331	22359	21998	22730	21698	22186	21425	22187	21161
KroB100	22989	22627	23035	22436	22993	22255	23620	22090	23169	21960
Eil101	691	663	683	660	676	650	673	657	659	656
Pr107	34684	34155	34018	33380	32369	32605	32403	31931	31385	31257
KroB150	28090	26922	28261	26798	27729	26678	27784	26565	27246	26462
KroA200	49021	30398	48570	30300	48379	30167	48206	30035	48990	29921
Tsp225	4121	4022	4186	4010	4080	3998	4163	3986	4445	3975
A280	3029	2712	2983	2708	2998	2705	3028	2702	3048	2700
Lin318	45943	43941	46351	43581	46353	43407	45479	43269	45901	43137

Çizelge 5.1'de MDCP'nin TWGA ile sadece küçük gezgin satıcı sayıları için rekabet edebildiği görüşmelmektedir. Özellikle gezgin satıcı sayısı 5'i geçtikten sonra TWGA MDCP'ya göre çok daha üstün bir performans göstermektedir. Günlük hayat uygulamalarında m sayısının büyük değerler alacağını göz önüne alınırsa, önerilen algoritmanın çok daha etkin ve kullanılabilir olduğu görülmektedir.

TWGA'nın performansının gezgin satıcı problemi üzerinde test edilebilmesi için TSPLIB (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/) üzerindeki sonuçlar; NN algoritması, açgözlü algoritma ve Nuriyeva et al., (2012) tarafından önerilen 3 sezgisel algoritma ile karşılaştırılmıştır. Nuriyeva et al., (2012) tarafından önerilen algoritmalar da 'Alg1, Alg2, Alg3' belirli bir açgözlü kritere göre çözüme her adımda bir ayrıt eklenmektedir.

Çizelge 5.2'de önerilen algoritmanı gezgin satıcı problemi için hesaplama denemeleri verilmiştir. Çizelgede her bir satır için, ilk sütun çizgenin ismini (çizgenin adı çizgenin boyutunu içermektedir), ikinci sütun en kısa turun uzunluğunu ve geri kalan sütunlarda ise sırasıyla en yakın komşu algoritması, açgözlü algoritma, Alg1, Alg2, Alg3 ve önerilen algoritma tarafından bulunan sonuçlar ve programların çalışma süreleri bulunmaktadır (Çizelge 5.1'deki TWGA'nın çalışma süreleri için de Çizelge 5.2'deki çalışma süreleri göz önüne alınabilir.). En yakın komşu algoritması, açgözlü algoritma ve Alg1, Alg2, Alg3 algoritmaları için olan sonuçlar ve çalışma süreleri 'Experimental Analysis of New Heuristics for the TSP' (Nuriyeva et al., 2012) adlı makaleden alınmıştır.

Çizelge 5.2'de görüldüğü gibi; NN algoritması 3 kez, açgözlü algoritma 1 kez, Alg2 6 kez, Alg3 4 kez ve TWGA 19 kez en iyi sonucu bulmuştur. Ayrıca TWGA, NN algoritmasının en iyi değeri bulduğu 3 örnek olan 'Berlin52, pr107, pr152' örnekleri hariç, en iyi sonucu bulamadığı örneklerde de en iyi sonuca çok yakın çözümler bulabilmiştir. NN algoritması hariç, diğer tüm algoritmaların bir tüm çözüm uzayından açgözlü bir kritere göre bir seçim yaparak çalıştığı göz önüne alındığında, TWGA gezgin satıcı algoritması için diğer açgözlü yaklaşımlara göre çok daha etkin bir algoritma olduğu görülmektedir.

Çizelge 5.2 Önerilen Algoritmanın GSP için Hesaplama Denemeleri

G	Optimal	NN	Açgözlü	Alg1	Alg2	Alg3	TWGA
ulysses16	74,108	78,127 0,000	88,923 0,000	75,342 0,000	74,198 0,000	75,138 0,000	74,096 0,000
ulysses22	75,665	86,905 0,000	89,436 0,010	77,915 0,000	77,711 0,000	78,675 0,000	75,680 0,000
bayg29	9074,148	9964,781 0,000	9886,208 0,015	9448,860 0,015	9768,448 0,000	9196,496 0,001	9077,916 0,000
dantzig42	699	822,095 0,000	843,737 0,062	744,155 0,016	784,139 0,015	784,908 0,002	723,168 0,003
att48	33523,708	39236,885 0,000	38849,621 0,125	37500,424 0,015	37057,581 0,016	36325,328 0,003	36144,570 0,006
eil51	429,983	505,774 0,016	481,518 0,125	443,070 0,031	495,628 0,031	440,746 0,000	438,733 0,0005
berlin52	7544,365	8182,192 0,000	9954,062 0,281	9047,211 0,031	9413,732 0,047	8618,198 0,002	8341,849 0,006
st70	678,597	761,689 0,000	746,044 0,485	785,284 0,094	811,974 0,094	727,778 0,011	720,337 0,025
eil76	545,387	612,656 0,016	617,131 0,672	588,074 0,140	606,117 0,140	581,407 0,010	568,415 0,024
gr96	512,309	603,302 0,015	580,101 1,609	581,877 0,235	540,357 0,235	550,495 0,029	544,271 0,076
rat99	1211	1369,535 0,016	1528,308 1,875	1311,904 0,266	1273,747 0,282	1316,432 0,030	1281,516 0,086
kroA100	21236,951	24698,497 0,016	24197,285 1,937	26135,302 0,360	24697,677 0,391	24093,242 0,020	24124,847 0,079
kroB100	22141	25882,973 0,016	25815,214 2,469	24700,544 0,406	23651,697 0,406	23419,490 0,031	23953,521 0,078
kroC100	20750,762	23566,403 0,015	25313,671 2,610	23962,861 0,391	24879,757 0,391	23512,300 0,030	22805,783 0,074
kroD100	21294,290	24855,799 0,016	24631,533 2,359	24783,197 0,422	23201,380 0,390	24758,054 0,010	23925,937 0,078
kroE100	22068	24907,022 0,016	24420,355 2,609	26036,072 0,375	25499,724 0,406	24822,113 0,010	23415,375 0,079
rd100	7910,396	9427,333 0,015	8702,605 2,922	9866,781 0,406	8945,544 0,375	9384,955 0,030	8675,069 0,093
eil101	642,309	736,368 0,015	789,112 2,609	712,461 0,329	694,685 0,359	704,361 0,010	680,001 0,06
lin105	14382,995	16939,441 0,015	16479,785 3,187	19679,294 0,360	17744,411 0,344	18354,693 0,030	16736,615 0,113
pr107	44303	46678,154 0,016	48261,816 2,109	56635,995 0,453	47060,739 0,438	54003,941 0,010	50567,437 0,062
gr120	1666,508	1850,263 0,032	1915,918 4,282	1823,521 0,640	1737,452 0,703	1703,981 0,049	1726,534 0,181
pr124	59030	67056,601 0,031	73952,492 5,281	79142,752 0,797	64254,490 0,797	70395,718 0,056	65898,742 0,197
bier127	118282	133970,646 0,032	136924,859 9,453	134485,140 0,656	134398,465 0,687	129348,929 0,057	129518,921 0,15
ch130	6110,860	7198,741 0,016	7142,045 7,688	6963,303 0,875	6636,392 0,953	6873,837 0,055	6601,554 0,225
pr136	96772	114560,902 0,031	119553,703 6,312	108776,600 0,93	104197,185 0,937	105557,453 0,081	102941,562 0,166
pr144	58537	60963,265 0,031	74613,406 8,438	79638,710 1,125	66682,412 1,109	91107,078 0,057	67818,171 0,31
kroA150	26524	31482,020 0,047	31442,994 11,094	31027,270 1,469	28444,581 1,453	30216,728 0,082	28758,292 0,427
kroB150	26130	31320,340 0,047	31519,083 11,156	31824,589 1,547	30815,938 1,454	29631,101 0,064	29022,5 0,424
pr152	73682	79566,585 0,047	84881,429 10,531	98730 1,297	84999,926 1,329	93118,625 0,07	87531,093 0,34
u159	42080	48586,725 0,031	50238,179 10,735	54194,759 1,641	50014,988 1,656	49710,097 0,102	47410,515 0,236
rat195	2323	2628,561 0,109	2957,176 29,719	2763,001 3,563	2637,019 3,532	2554,672 0,085	2505,651 1,239
d198	15780	17809,725 0,109	18595,822 39,625	18147,697 3,625	17472,707 3,188	17941,087 0,192	17263,056 1,018
kroA200	29368	34547,691 0,125	37650,812 45	35195,046 5,187	35792,822 5,172	33629,972 0,112	31596,593 1,392

6. SONUÇ

Rotalama problemlerinin incelendiği bu tezde, bu problemlerin teorik olarak en saf halleri olarak görülen gezgin satıcı problemi ve çoklu gezgin satıcı problemi ele alınmış, problemler için literatür taraması yapılmış ve yeni sezgisel algoritmalar önerilmiştir. Önerilen algoritmaların C dilinde bilgisayar programları oluşturularak, uluslararası problem kütüphanelerinden alınan test problemleri üzerinde var olan çözüm yöntemleri ile karşılaştırılmaları yapılmıştır. Yapılan karşılaştırmalar sonucunda bu tezde önerilen *İki-Yönlü Açgözlü Algoritma*'nın hem çoklu gezgin satıcı problemi üzerinde hem de gezgin satıcı problemi üzerinde var olan yöntemlerden daha etkin sonuçlar bulduğu görülmüştür. Bu bağlamda önerilen çözüm yöntemlerinin rotalama problemleri literatüre katkıda bulunduğu düşünülmektedir. Bu tezde elde edilen sonuçlar uluslararası kongrelerde sunulmuştur.

.

KAYNAKLAR DİZİNİ

Aarts, E. and Lenstra, J.K., 1997, Local Search in Combinatorial Optimization, John Wiley & Sons, London, 512p.

Agarwala, R., Applegate, D. L., Maglott, D., Schuler, G. D. and Schaffer, A. A., 2000, A fast and scalable radiation hybrid map construction and integration strategy, Genome Research,10, 350–364pp.

Agnarsson, G. and Greenlaw, R., 2007, Graph Theory: Modeling, Applications, and Algorithms, Prentice Hall, 464p.

Anbuudayasankar, S. P., Ganesh, K. and Mohapatra, S., 2016, Models For Practical Routing Problems in Logistics. Springer International Pu.

Applegate, D. L., Bixby, R. E., Chavatal, V. and Cook, W. J., 1995, Finding cuts in the TSP (A preliminary report), DIMACS Technical Report 95-05, DIMACS, Rutgers University, New Brunswick, New Jersey, USA.

Applegate, D.L., Bixby, R.E., Chvatal, V. and Cook, W.J., 2003, Implementing the Dantzig– Fulkerson–Johnson algorithm for large scale traveling salesman problems. Math Program Ser B Vol. 97,91–153pp.

Applegate, D. L., Bixby, R. E., Chavatal, V. and Cook, W. J., 2006, The Travelling Salesman Problem, A Computational Study, Princeton University Press, Princeton and Oxford, 593p.

Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W., Espinoza, D., Goycoolea, M., and Helsgaun, K., 2009, Certification of an optimal tsp tour through 85,900 cities, Operations Research Letters, 37(1), 11 – 15pp.

Arora, S., 1998, Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM (JACM), 45(5), 753-782pp.

Bakır, M.A. ve Altunkaynak, B., 2003, Tamsayılı Programlama, Nobel Yayın Dağıtım, Ankara, 630 s.

Başkaya, Z., 2005, Tamsayılı Programlama Algoritmaları ve Bilgisayar Uygulamalı Problem Çözümleri, Ekin Kitabevi, Ankara, 236s.

Bektas, T., 2006, The Multiple Traveling Salesman Problem: an Overview Of Formulations and Solution Procedures, Omega, 34(3), 209-219pp.

KAYNAKLAR DİZİNİ (devam)

Bland R.G. and Shallcross D.F., 1989, Large traveling salesman problems arising from experiments in X-ray crystallography: A preliminary report on computation. Oper. Res. Lett., 8:125–128pp.

Bondy, J.A. and Murty, U.S.R., 2008, Graph Theory, Springer, New York, 654p.

Bruno, G., Genovese, A. and Improta, G., 2011, Routing problems: a historical perspective. BSHM Bulletin, 26(2), 118-127pp.

Clarke, G. and Wright, J.W., 1964, Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, Operations Research, 12, 568-581pp.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C., 2001, Introduction To Algorithms, The MIT Press, Cambridge, 1180p.

Cura, T., 2008, Modern Sezgisel Teknikler ve Uygulamaları, Papatya Yayıncılık, İstanbul, 173s.

Çölkesen, R., 2002, Veri Yapıları ve Algoritmalar, Papatya Yayıncılık, İstanbul, 424s.

Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M., (1954), Solution of a large-scale Traveling Salesman Problem, Operational Research, 2, 393-410pp.

Dantzig, G. B. and Ramser, J. H., 1959, The truck dispatching problem. Management science, 6(1), 80-91pp.

Davendra, D., 2010, Traveling Salesman Problem, Theory and Applications, India, 324p.

Dorigo, M. and Gambardella, L.M., 1997, Ant Colonies for the Traveling Salesman Problem, BioSystems, 43(2), 73-81pp.

Elmas, Ç., 2007, Yapay Zeka Uygulamalar , Seçkin Yayıncılık, Ankara.

Fogel, D.B., 1990, A parallel processing approach to a multiple travelling salesman problem using evolutionary programming. In: Proceedings of the fourth annual symposium on parallel processing. Fullerton, CA, pp. 318–26pp.

KAYNAKLAR DİZİNİ (devam)

Garey, M. R. and Johnson, D. S., 1979, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 340p.

Gass, S. I. And Assad, A. A., 2004, An Annotated Timeline of Operations Research: An Informal History, International Series in Operations Research & Management Science , Vol. 75, Springer.

Gavish, B. and Srikanth, K., 1986, An optimal solution method for large-scale multiple traveling salesman problems. Operations Research, Vol. 34, No. 5, 698–717pp.

Gilmore, P. C. and Gomory R. E., 1964, Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. Operations Research, 12, 655–679pp.

Glover, F. and Laguna, M., 1997, Tabu Search, Kluwer Academic Publishers, Boston.

Gromicho, J., Paixão, J. and Branco, I., 1992, Exact solution of multiple traveling salesman problems. Combinatorial optimization. Springer, Berlin, Heidelberg, 291–92pp.

Gross, J.L. and Yellen, J., 2004, Handbook of Graph Theory, CRC Press, London, 1167p.

Grötschel, M. and Holland, O., 1991, Solution of large-scale symmetric traveling salesman problems, Mathematical Programming 51, 141-202pp.

Gutin, G. and Punnen, A. (eds.), 2002, The Traveling Salesman Problem and Its Variations, volume 12 of Combinatorial Optimization. Kluwer, Dordrecht, 830p.

Hou, M. and Liu, D., 2012, A novel method for solving the multiple traveling salesmen problem with multiple depots, Chinese Science Bulletin, 57(15), 1886-1892pp.

Hubert L.J. and Baker F.B.,1978, Applications of combinatorial programming to data analysis: the traveling salesman and related problem. Psychometrika, 43:81–91pp.

Jain, A. K., Murty M. N. and Flynn P. J., 1999, Data clustering: A review. ACM Computing Surveys 31, 264–323pp.

KAYNAKLAR DİZİNİ (devam)

Johnson, D.S. and Papadimitriou, C., 1985, Performance Guarantees for Heuristics, In Lawler et al., chapter 5, 145-180p.

Johnson, D.S. and McGeoch, L.A., 1997, The Traveling Salesman Problem: A Case Study in Local Optimization, 215-310pp.

Johnson, D.S. and McGeoch, L.A., 2002, Experimental Analysis of Heuristics for the STSP, The Traveling Salesman and Its Variations, Gutin and Punnen (eds), Kluwer Academic Publishers, 369-443pp.

Jongen, H. T., Meer, K. and Triesch, E., 2004, Optimization Theory, Springer Science + Business Media, Inc., Kluwer Academic Publishers, Boston. 313-357pp.

Kara, I. and Bektas, T., 2006, Integer linear programming formulations of multiple salesman problems and its variations, European Journal of Operational Research, 174(3), 1449-1458pp.

Karaboğa, D., 2004, Yapay Zeka Optimizasyon Algoritmaları, Atlas Yayın Dağıtım, İstanbul, 199s.

Kizilates G., and Nuriyeva F., 2013, A New hybrid heuristic algorithm for solving TSP, Anadolu Unıversity Journal of Science and Technology–B Theoretical Sciences, 2(2), 143-148pp.

Kuzu, S., Önay, O., Şen, U., Tunçer, M., Yıldırım, B. F., ve Keskintürk, T., 2014, Gezgin Satıcı Problemlerinin Metasezgiseller İle Çözümü, Istanbul University Journal of the School of Business, *43*(1), 1-27s.

Laporte, G. and Nobert, Y., 1980, A cutting planes algorithm for the m-salesmen problem. Journal of the Operational Research Society, Vol. 31, 1017–23pp.

Laporte, G. And Osman, I. H., 1995, Routing problems: A bibliography. Annals of Operations Research, *61*(1), 227-262pp.

Lenstra, J.K. and Kan A.H.G.R., 1974, Some Simple Applications of the Travelling Salesman Problem. BW 38/74, Stichting Mathematisch Centrum, Amsterdam.

Lin, S. and Kernighan, B.W., 1973, An effective heuristic algorithm for the traveling salesman problem, Operational Research, 21, 498-516pp.

KAYNAKLAR DİZİNİ (devam)

Matai, R., Singh, S., and Mittal, M. L., 2010, Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches, Traveling Salesman Problem, Theory and Applications, InTech.

Nabiyev, V. V., 2005, Yapay Zeka, Seçkin Yayıncılık, Ankara, 764s.

Nabiyev, V. V., 2009, Teoriden Uygulamaya Algoritmalar, Seçkin Yayıncılık, Ankara, 824s.

Nuriyeva F., Kizilates G., and Berberler M. E., 2012, Experimental Analysis of New Heuristics for the TSP, Proceeding of the fourth International Conference, Problems of Cybernetics and Informatics, PCI 2012, Section 7 Control and Optimization, 4, Baku, Azerbaijan, September 12-14, (2012), 1-5pp.

Padberg, M., and Rinaldi, G., 1991, A branch-and-cut algorithm for the symmetric traveling salesman polytope, Mathematical Programming, 47, 219-257pp.

Papadimitriou, C. H. and Steiglitz, K., 1998, Combinatorial Optimization: Algorithms and Complexity, Dover Publications Inc., New York, 496p.

Park, Y.B., 2001, A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines, Int. Journal of Productions Econ., 73(2):175-188pp.

Plante R.D., 1988, The nozzle guide vane problem. *Oper. Res.*, 36:18–33pp.

Potvin, J. Lapalme, G. and Rousseau, J., 1989, A generalized k-opt exchange procedure for the MTSP. Information Systems 1989;27(4): 474–81pp.

Russell, R.A., 1977, An effective heuristic for the m-tour traveling salesman problemn with some side conditions. Operations Research, Vol. 25, No. 3, 517–24pp.

Saleh, H.A. and Chelouah, R., 2004, The design of the global navigation satellite system surveying networks using genetic algorithms. Engineering Applications of Artificial Intelligence, Vol. 17, 111–22pp.

Song, C. H., Lee, K., and Lee, W. D., 2003, Extended simulated annealing for augmented TSP and multi-salesmen TSP. In Neural Networks, 2003. Proceedings of the International Joint Conference on IEEE, Vol. 3, pp. 2340-2343pp.

KAYNAKLAR DİZİNİ (devam)

TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types, available at: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, (Erişim tarihi: 15 Haziran 2018), Ruprecht-Karls-Universität Heidelberg.

Woolsey, L. A., 1998, Integer programming. Wiley, New York, 264p.

Yu, Z., Jinhai, L., Guochang, G., Rubo, Z. and Haiyan, Y., 2002, An implementation of evolutionary computation for path planning of cooperative mobile robots. Intelligent Control and Automation, Proceedings of the 4th World Congress on. IEEE, 1798-1802pp.

Zhang, T., Gruver, W.A. and Smith, M.H., 1999, Team scheduling by genetic search. Proceedings of the second international conference on intelligent processing and manufacturing of materials, Vol. 2, 839–44pp.

ÖZGEÇMİŞ

5 Nisan 1988 tarihinde Ankara'nın Keçiören ilçesinde doğdu. Lise eğitimini Aldemir Atilla Konuk Anadolu Lisesinde tamamladı. 2007 yılında girdiği üniversite sınavında Ege Üniversitesi Matematik bölümünü kazandı. 2011 yılında Bilgisayar Bilimleri Ağırlık Matematikçi unvanını alarak mezun oldu. 2011-2013 yılları arasında Ege Üniversitesi Fen Bilimleri Enstitüsü Matematik Anabilim Dalı Bilgisayar Bilimleri Bilim Dalında "Minimum Tepe Örtüsü Problemi Üzerine" isimli tez çalışmasıyla Yüksek Lisans eğitimini başarılı bir şekilde tamamladı. 2013 yılında yine Ege Üniversitesi Fen Bilimleri Enstitüsü Matematik Anabilim Dalı Bilgisayar Bilimleri Bilim Dalı'nda Doktora eğitimine başladı.

YAYINLAR

Makaleler

A) Uluslararası Science Citation Index ve Sosial Science Citation Indexce taranan dergilerde yayınlanmış makaleler

1. Tanir D., **Ugurlu O.**, Guler A., Nuriyev U., (2018), "One-Dimensional Cutting Stock Problem With Divisible Items: A Case Study In Steel Industry", *TWMS Journal of Applied and Engineering Mathematics.* Basımda.
2. Durmusoglu E.., **Ugurlu O.**, Akan S., et al,. (2018), "Hippocampal shape alterations in healthy young women with familial risk for unipolar depression". *Comprehensive Psychiatry,* Vol. 82, pp.7-13.
3. Kutucu H., Nuriyeva F., **Ugurlu O.**, (2016), "The Rainbow Connection Problem: Mathematical Formulations", *Ars Combinatoria,* Vol. CXXVII, pp.101-108.
4. Kurt M., Berberler, M.E., **Ugurlu O.**, (2015), "A New Algorithm for Finding Vertex-Disjoint Paths", *The International Arab Journal of Information Technology,* Vol. 12, No. 6, pp.550-555.
5. Isıklı S., **Ugurlu O.**, Durmusoglu E., et al,. (2013), "Altered Hippocampal Formation Shape In First-Episode Depressed Patients At 5-Year Follow-Up". *Journal of Psychiatric Research,* Vol. 47(1), pp.50-55.

B) Uluslararası diğer bilimsel dergilerde yayınlanmış makaleler

1. Nuriyeva F., **Ugurlu O.**, Kutucu H., (2013), "A Mathematical Model for the Rainbow Vertex-Connection Number", *Advances in Computer Science : an International Journal*, Vol. 2, Issue 4, No: 5, pp. 103-106.
2. **Ugurlu O.**, (2013), "A New Heuristic Algorithm to Solve the Maximum Independent Set Problem", *Mathematical and Computational Applications,* Vol 18(3), pp. 495-501.
3. **Ugurlu O.**, Berberler M.E., (2013), "Erratum to: Adjacency Matrix based Method to Compute the Node Connectivity of a Computer Communication Network", *IJCSI International Journal of Computer Science Issues*, Vol. 10(2), No.1, pp. 414-415.
4. **Ugurlu O.**, (2013), "A New Hybrid Genetic Algorithm for Vertex Cover Problem", *Anadolu University Journal of Science and Technology – A Applied Sciences and Engineering*, Vol. 14, No. 3, pp. 277-282.
5. **Ugurlu O.,** (2011), "A New Solution Approach fort he Integer Fair Division Problem", *World Applied Sciences Journal*, Vol. 13, No. 12, pp. 2444 - 2450.

C) Ulusal hakemli bilimsel dergilerde yayınlanmış makaleler

1. Tanir D., **Ugurlu O.**, Kapar M., Nuriyev U., (2017), "Birleşik Stok Kesme ve Patern Sıralama Problemi için Bir Sezgisel Algoritma", *Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi,* 22(1), 300-305.
2. Berberler M.E., **Ugurlu O.**, Kizilates G., (2012), "Macar Algoritmasının Sıfırları Kapatma Alt Yordamı Üzerine", *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi,* Vol. 18, No:2, ss. 85-94.

Kongreler

D) Uluslararası bilimsel toplantılarda yayınlanmış bildiriler:

1. Nuriyev U., **Ugurlu O.,** Nuriyeva F., (2018), "Self-Organizing Iterative Algorithm for Travelling Salesman Problem", *Technology, Culture and International Stability* (TECIS 2018), Baku, Sep. 13-15. Basımda.
2. Nuriyev U., **Ugurlu O.,** Nuriyeva F., (2018), "A Simple Algorithm for The Multi-Depot Multiple Traveling Salesman Problem", *The 6th International Conference on Control and Optimization with Industrial Applications* (COIA 2018), Baku, July 11-13, Volume II, 235-237.
3. **Ugurlu O.,** Tanir D., (2016), "A Hybrid Genetic Algorithm for Minimum Weight Dominating Set", *Recent Developments and the New Direction in Soft-Computing Foundations and Applications.* Springer, Cham, 2018. 137-148.

4. **Ugurlu O.,** Tanir D., Nuri E., (2016), "A Better Heuristic for the Minimum Connected Dominating Set in Ad Hoc Networks", *Proceeding of the 10th International Conference on Application of Information and Communication Technologies* (AICT2016), pp. 1-4, Baku, Azerbaijan, October 12-14, (SCI, IEEE Xplore Digital Library).

5. Durmusoglu T.e., **Ugurlu O.**, et al. (2015), "3D Shape Analysis of the Hippocampus in Individuals at High Familial Risk for Depression", *APA 186th American Psychiatric Association Annual Meeting, Young Investigator Research Poster,* No:8 pp. 46, Toronto, Canada, May 16-20.

6. **Ugurlu O.**, Berberler M.E., Nuriyev U., (2014), "A New Genetic Approach for Maximum Independent Set Problem", *Proceedings of the 4th International Scientific Conference of Students And Young Scientists, "Theoretical and Applied Aspects of Cybernetics",* (TAAC 2014), pp. 43-48, Kiev, Ukrain, November 24-28.

7. **Ugurlu O.**, Nuriyeva F.**,** Kutucu H., (2013), "On the Rainbow Vertex Connection Number of Graphs", *International Conference on Applied Analysis and Mathematical Modelling* (ICAAMM 2013), pp. 262, Istanbul, Turkey, June 2-5.

8. Nuriyeva, F. ., **Ugurlu O.**, Kutucu H., (2013), "A Mathematical Model For Finding The Rainbow Connection Number", *Proceeding of the 7th International Conference on Application of Information and Communication Technologies* (AICT2013)**,** pp. 2 - 4, Baku, Azerbaijan, October 23-25, (SCI, IEEE Xplore Digital Library).

9. **Ugurlu O.**, Nuriyeva F., (2012), "A New Heuristic Algorithm for Rainbow Vertex Connection", *Proceedings of the 2nd International Scientific Conference of Students And Young Scientists, "Theoretical and Applied Aspects of Cybernetics",* (TAAC 2012), pp. 82-84, Kiev, Ukrain, November 12-15.

10. **Ugurlu O.**, (2012), "A New Heuristic Mutation Method for Solving Vertex Cover Problem", *8. Uluslararası İstatistik Günleri Sempozyumu*, pp. 381-382, Eskişehir, Türkiye, October 11-13.

11. **Ugurlu O.**, (2012), "New Heuristic Algorithm for Unweighted Minimum Vertex Cover", *Proceeding of the fourth International Conference "Problems of Cybernetics and Informatics"*, PCI 2012, Section IV, "Control and Optimization", Vol. 4, pp. 69-73, Baku, Azerbaijan, September 12-14, (SCI, IEEE Xplore Digital Library).

12. **Ugurlu O.**, Berberler M.E., Kizilates G., Kurt M., (2012), "A New Algorithm for Finding Minimum Vertex Cut Set", *Proceeding of the fourth International Conference "Problems of Cybernetics and Informatics"*, PCI 2012, Section 7, "Control and Optimization", Vol. 4, pp. 145-148, Baku, Azerbaijan, September 12-14, (SCI, IEEE Xplore Digital Library).

E) Ulusal bilimsel toplantılarda yayınlanmış bildirileri:

1. Nuriyev U., **Ugurlu O.**, Alizade E., (2018), "Sıfır-Bir Sırt Çantası Problemleri için Garanti Değerli Açgözlü Algoritmalar Uzerine", *Yöneylem Araştırması ve Endüstri Mühendisliği* (YAEM-2018) 36.Ulusal Kongresi, ss. 26, Anadolu Üniversitesi, Eskişehir, Türkiye, Haziran 26-29.

2. Deniz T., **Ugurlu O.**, Guler A., Nuriyev U. , (2016), "Çelik Endüstrisinde Karşılaşılan İki Amaçlı Bir Boyutlu Stok Kesme Problemi Üzerine", *Yöneylem Araştırması ve Endüstri Mühendisliği* (YAEM-2016) 36.Ulusal Kongresi, ss. 26, Yaşar Üniversitesi, İzmir, Türkiye, Temmuz l 3-15.

3. Nuri E., **Ugurlu O.**, Nuriyeva F.**,** (2016), "Asal Sayıların Gösterimi Üzerine", *11. Ankara Matematik Günleri*, Ankara Üniversitesi, ss. 49, Ankara, Türkiye, Mayıs 26-27.

4. **Ugurlu O.**, (2012), "Minimum Tepe Örtüsü Problemi için Yeni bir Çözüm Yaklaşımı", *XXV. Ulusal Matematik Sempozyumu,* ss. 88, Niğde Üniversitesi, Niğde, Eylül 5-8.

5. Atlamaz M., Yilmaz A., **Ugurlu O.**, (2012), "Geri Bildirimler Karar Vermeyi Nasıl Etkiliyor? Teknolojik Geri Bildiriminin İnsan Geri Bildirimi ile Karşılaştırılması", *48. Ulusal Psikiyatri Kongresi*, ss. 231, Bursa, Türkiye, Ekim 9-13.

6. Atlamaz M., Yilmaz A., **Ugurlu O.**, (2011), "Depresyonda Hipokampus Alt Bölgelerinin Değişimi 5 Yıllık İzlem Çalışması", *47. Ulusal Psikiyatri Kongresi*, ss. 355, Antalya, Türkiye, Ekim 26-30.

EKLER

Ek 1 Gezgin Satıcı Programı için Önerilen İteratif Sezgisel Algoritmanın C Programlama Dili Kodu

Ek 2 Çoklu Gezgin Satıcı Programı için Önerilen TWGA Algoritmasının C Programlama Dili Kodu

Ek 1 - Gezgin Satıcı Programı için Önerilen İteratif Sezgisel Algoritmanın C Programlama Dili Kodu

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#define N 16

#define max 10000

FILE *f,*f2;
int gidildi[N+1]={0};
double a[N+1][N+1]={0};
double u[N+1][N+1]={0};
double u2[N+1][N+1]={0};
int isr[N+1][N+1]={0};
float p[N+1][3]={0};
int  bestczm[N+1][N+1]={0};

int  k[N+1][N+1]={0};
int  git[N+1]={0};
int  cozum2[N+1][N+1]={0};
int  ban[N+1][N+1]={0};
int  iz[N+1]={0};
int  use[N+1]={0};

int eij[(N*(N-1)/2)+1][3]={0};
float aij[(N*(N-1)/2)+1]={0};

int
i,j,bas,enki,tepe,maxi,maxj,as=0,t1,t2,z,syc1=0,g1=0,zz,sakla1,sakla2,ss1,ss2,ns,ilk,tarz,a
ss;
double
enk,top,sonuc,sbt,syc,syc2,maxx,sontop=0,sontop2=0,besttop,lasttop,tpara,maxayrit,mina
yrit,maxu,minu,tt;
```

```
int main()
{
	srand(time(NULL));

	f=fopen("ulysses16.txt","r");
	f2=fopen("sonuclar.txt","w");

	fscanf(f,"%d",&i);
	if (i!=N)
	{
		printf("Boyut uyumsuz !");
		return 0;
	}

	for(i=1;i<=N;i++)
	{
		fscanf(f,"%f",&p[i][0]);
		fscanf(f,"%f",&p[i][1]);
		fscanf(f,"%f",&p[i][2]);

	}
   fclose(f);

	maxayrit=0;
	minayrit=10000;
	for(i=1;i<N;i++)
	{
		//a[i][i]=max;
		for(j=i+1;j<=N;j++)
		{
			a[i][j]=sqrt((p[i][1]-p[j][1])*(p[i][1]-p[j][1])+(p[i][2]-
p[j][2])*(p[i][2]-p[j][2]));
			a[j][i]=a[i][j];
			if (maxayrit<a[i][j]) maxayrit=a[i][j];
			if (minayrit>a[i][j])
			{
```

```
                                minayrit=a[i][j];
                                sakla1=i;
                                sakla2=j;
                        }

                }
        }
clock_t start, end;
start = clock();

        for(i=1;i<=N;i++)
        {
                for(j=1;j<=N;j++)
                {
                        u[i][j]=maxayrit/a[i][j];
                }
        }

        z=0;
        maxayrit=0;
        for(i=1;i<=N;i++)
                for(j=i+1;j<=N;j++)
                {
                        z++;
                        eij[z][0]=z;
                        eij[z][1]=i;
                        eij[z][2]=j;
                        aij[z]=u[i][j];
                }

        for(i=1;i<=N*(N-1)/2;i++)
                for(j=i+1;j<=N*(N-1)/2;j++)
                        if (aij[i] < aij[j])
                        {
                                aij[0]=aij[i];
```

```
                    aij[i]=aij[j];
                    aij[j]=aij[0];

                    eij[0][0]=eij[i][0];
                    eij[i][0]=eij[j][0];
                    eij[j][0]=eij[0][0];

                    eij[0][1]=eij[i][1];
                    eij[i][1]=eij[j][1];
                    eij[j][1]=eij[0][1];

                    eij[0][2]=eij[i][2];
                    eij[i][2]=eij[j][2];
                    eij[j][2]=eij[0][2];
                }

        ///////////////İlk Greedy/////////////

        zz=0;
        as=0;
        syc=1;

greedydevam:
        zz++;
        maxi=eij[zz][1];
        maxj=eij[zz][2];

        if(syc==1)
        {
            cozum2[maxi][maxj]=1;
            cozum2[maxj][maxi]=1;

            k[maxi][0]++;
            k[maxi][k[maxi][0]]=maxj;
```

```
            k[maxj][0]++;
            k[maxj][k[maxj][0]]=maxi;

            as++;
            syc=0;

            use[maxi]=1;
            use[maxj]=1;
}
else
{
            for(i=0;i<=N;i++)
            {
                        git[i]=0;
                        iz[i]=0;
            }
            t1=maxi;
            t2=maxj;

            syc1=0;

            git[0]++;
            git[git[0]]=t1;
            iz[t1]=1;

            for(i=1;i<=git[0];i++)
            {
                        z=git[i];
                        for(j=1;j<=k[z][0];j++)
                        {
                                    if(iz[k[z][j]]==0)
                                    {
                                                iz[k[z][j]]=1;
                                                git[0]++;
                                                git[git[0]]=k[z][j];
```

```
                                if(k[z][j]==t2) syc1=1;

                        }
                }
        }
        if(use[t1]<2 && use[t2]<2 && syc1==0)
        {
                as++;
                cozum2[maxi][maxj]=1;
                cozum2[maxj][maxi]=1;

                use[t1]++;
                use[t2]++;

                k[maxi][0]++;
                k[maxi][k[maxi][0]]=maxj;

                k[maxj][0]++;
                k[maxj][k[maxj][0]]=maxi;
        }
}

if(as!=N-1)     goto greedydevam;

for(i=1;i<=N;i++)
{
        for(j=1;j<=N;j++)
        {
                if(use[i]==1 && use[j]==1 && i!=j)
                {
                        cozum2[i][j]=1;
                        cozum2[j][i]=1;

                        use[i]++;
```

```
                        use[j]++;
                        as++;
                        k[i][0]++;
                        k[i][k[i][0]]=j;

                        k[j][0]++;
                        k[j][k[j][0]]=i;
                }
        }
}

for(i=1;i<=N;i++)
        for(j=1;j<=N;j++)
                bestczm[i][j]=cozum2[i][j];

top=0;
for(i=1;i<N;i++)
{
        for(j=i+1;j<=N;j++)
        {
                if (cozum2[i][j]==1)
                {
                        u[i][j]++;
                        u[j][i]++;
                        top+=cozum2[i][j]*a[i][j];
                }
        }
}
besttop=top;
//////////////////////////////////////
```

```
        /////////İlk NN//////////////////////
        bas=sakla1;
        tepe=1;
        gidildi[bas]=1;

        tepe++;
        ilk=sakla2;
        gidildi[ilk]=1;

        isr[bas][ilk]=1;
        isr[ilk][bas]=1;

ilkk:
        enk=max;
        for(j=1;j<=N;j++)
        {
                if(a[bas][j]<enk && gidildi[j]==0)
                {
                        enk=a[bas][j];
                        enki=j;
                        tarz=0;
                }
        }

        for(j=1;j<=N;j++)
        {
                if(a[ilk][j]<enk && gidildi[j]==0)
                {
                        enk=a[ilk][j];
                        enki=j;
                        tarz=1;
                }
        }
```

```
        if (tarz==0)
        {
                gidildi[enki]=1;
                isr[bas][enki]=1;
                isr[enki][bas]=1;

                bas=enki;
                tepe++;

        }else if (tarz==1)
        {
                gidildi[enki]=1;
                isr[ilk][enki]=1;
                isr[enki][ilk]=1;

                ilk=enki;
                tepe++;
        }

        if(tepe<N) goto ilkk;
        else goto son;
son:
        isr[bas][ilk]=1;
        isr[ilk][bas]=1;
        ss1=bas;
        ss2=ilk;
        top=0;
        ass=0;
        for(j=1;j<N;j++)
                for(z=j+1;z<=N;z++)
                        if (isr[j][z]==1)
                        {
                                top+=a[j][z];
                                ass++;
                        }
```

```
tpara=besttop/top;
for(i=1;i<N;i++)
{
	for(j=i+1;j<=N;j++)
	{
		if (isr[i][j]==1)
		{
			u[i][j]+=tpara;
			u[j][i]+=tpara;
		}
	}
}

if (besttop>top)
{
	besttop=top;
	for(i=1;i<=N;i++)
		for(j=1;j<=N;j++)
			bestczm[i][j]=isr[i][j];
}

for(j=1;j<=N;j++)
{
	gidildi[j]=0;
	for(z=1;z<=N;z++)
		isr[j][z]=0;
}

///////////////2.NND//////////////////////
bas=ss1;
tepe=1;
gidildi[bas]=1;
ilk=ss1;
```

```
ilk11:
        enk=max;
        for(j=1;j<=N;j++)
        {
                if(a[bas][j]<enk && gidildi[j]==0)
                {
                        enk=a[bas][j];
                        enki=j;
                        tarz=0;
                }
        }

        for(j=1;j<=N;j++)
        {
                if(a[ilk][j]<enk && gidildi[j]==0)
                {
                        enk=a[ilk][j];
                        enki=j;
                        tarz=1;
                }
        }

        if (tarz==0)
        {
                gidildi[enki]=1;
                isr[bas][enki]=1;
                isr[enki][bas]=1;

                bas=enki;
                tepe++;

        }else if (tarz==1)
        {
                gidildi[enki]=1;
                isr[ilk][enki]=1;
```

```
            isr[enki][ilk]=1;

            ilk=enki;
            tepe++;
        }

        if(tepe<N) goto ilk11;
        else goto son11;

son11:
        isr[bas][ilk]=1;
        isr[ilk][bas]=1;
        top=0;
        ass=0;
        for(j=1;j<N;j++)
            for(z=j+1;z<=N;z++)
                if (isr[j][z]==1)
                {
                    top+=a[j][z];
                    ass++;
                }

        tpara=besttop/top;
        for(i=1;i<N;i++)
        {
            for(j=i+1;j<=N;j++)
            {
                if (isr[i][j]==1)
                {
                    u[i][j]+=tpara;
                    u[j][i]+=tpara;
                }
            }
        }
```

```
if (besttop>top)
{
	besttop=top;
	for(i=1;i<=N;i++)
		for(j=1;j<=N;j++)
			bestczm[i][j]=isr[i][j];
}

for(j=1;j<=N;j++)
{
	gidildi[j]=0;
	for(z=1;z<=N;z++)
		isr[j][z]=0;
}

////////////////////////////////////

///////////////3.NND/////////////////////

bas=ss2;
tepe=1;
gidildi[bas]=1;
ilk=ss2;

ilk22:
	enk=max;
	for(j=1;j<=N;j++)
	{
		if(a[bas][j]<enk && gidildi[j]==0)
		{
			enk=a[bas][j];
			enki=j;
			tarz=0;
		}
	}
```

```
for(j=1;j<=N;j++)
{
	if(a[ilk][j]<enk && gidildi[j]==0)
	{
		enk=a[ilk][j];
		enki=j;
		tarz=1;
	}
}

if (tarz==0)
{

	gidildi[enki]=1;
	isr[bas][enki]=1;
	isr[enki][bas]=1;

	bas=enki;
	tepe++;

}else if (tarz==1)
{
	gidildi[enki]=1;
	isr[ilk][enki]=1;
	isr[enki][ilk]=1;

	ilk=enki;
	tepe++;
}

if(tepe<N) goto ilk22;
else goto son22;
```

```
son22:
        isr[bas][ilk]=1;
        isr[ilk][bas]=1;
        top=0;
        ass=0;
        for(j=1;j<N;j++)
                for(z=j+1;z<=N;z++)
                        if (isr[j][z]==1)
                        {
                                top+=a[j][z];
                                ass++;
                        }

        tpara=besttop/top;
        for(i=1;i<N;i++)
        {
                for(j=i+1;j<=N;j++)
                {
                        if (isr[i][j]==1)
                        {
                                u[i][j]+=tpara;
                                u[j][i]+=tpara;
                        }
                }
        }

        if (besttop>top)
        {
                besttop=top;
                for(i=1;i<=N;i++)
                        for(j=1;j<=N;j++)
                                bestczm[i][j]=isr[i][j];
        }
```

```
	for(j=1;j<=N;j++)
	{
		gidildi[j]=0;
		for(z=1;z<=N;z++)
			isr[j][z]=0;
	}

	for(i=0;i<=N;i++)
	{
		for(j=0;j<=N;j++)
		{
			cozum2[i][j]=0;
			k[i][j]=0;
			ban[i][j]=0;
		}
		use[i]=0;
	}
	as=0;

	////////////////////////////////////

	g1=5;
greedybas:
	maxu=0;
	for(i=1;i<=N;i++)
	{
		for(j=1;j<=N;j++)
		{
			u2[i][j]=u[i][j];
		}
	}

	syc=1;
greedy:
	maxx=-10000;
```

```
for(i=1;i<N;i++)
{
	for(j=i+1;j<=N;j++)
	{
		if(u2[i][j]>maxx && ban[i][j]==0)
		{
			maxx=u2[i][j];
			maxi=i;
			maxj=j;
		}
	}
}
ban[maxi][maxj]=1;
if(syc==1)
{
	cozum2[maxi][maxj]=1;
	cozum2[maxj][maxi]=1;

	k[maxi][0]++;
	k[maxi][k[maxi][0]]=maxj;

	k[maxj][0]++;
	k[maxj][k[maxj][0]]=maxi;

	as++;
	syc=0;

	use[maxi]=1;
	use[maxj]=1;
}
else
{
	for(i=0;i<=N;i++)
	{
		git[i]=0;
```

```
        iz[i]=0;
}
t1=maxi;
t2=maxj;

syc1=0;

git[0]++;
git[git[0]]=t1;
iz[t1]=1;

for(i=1;i<=git[0];i++)
{
        z=git[i];
        for(j=1;j<=k[z][0];j++)
        {
                if(iz[k[z][j]]==0)
                {
                        iz[k[z][j]]=1;
                        git[0]++;
                        git[git[0]]=k[z][j];

                        if(k[z][j]==t2) syc1=1;

                }
        }
}
if(use[t1]<2 && use[t2]<2 && syc1==0)
{
        as++;
        cozum2[maxi][maxj]=1;
        cozum2[maxj][maxi]=1;

        use[t1]++;
        use[t2]++;
```

```
                k[maxi][0]++;
                k[maxi][k[maxi][0]]=maxj;

                k[maxj][0]++;
                k[maxj][k[maxj][0]]=maxi;

        }
}

if(as!=N-1)     goto greedy;

for(i=1;i<=N;i++)
{
        for(j=1;j<=N;j++)
        {
                if(use[i]==1 && use[j]==1 && i!=j)
                {
                        cozum2[i][j]=1;
                        cozum2[j][i]=1;

                        use[i]++;
                        use[j]++;
                        as++;
                        k[i][0]++;
                        k[i][k[i][0]]=j;

                        k[j][0]++;
                        k[j][k[j][0]]=i;
                }
        }
}
```

```
top=0;
for(i=1;i<N;i++)
{
	for(j=i+1;j<=N;j++)
	{
		if(cozum2[i][j]==1)
		{
			top=top+a[i][j];
		}
	}
}
//////////////////////////////////////////

if(top==besttop)
{
	for(j=1;j<N;j++)
		for(z=j+1;z<=N;z++)
			if (cozum2[j][z]==1 && bestczm[j][z]==1)
			{
				u[j][z]--;
				u[z][j]--;
			}

}else if(top>besttop)
{
	tpara=besttop/top;
	for(j=1;j<N;j++)
		for(z=j+1;z<=N;z++)
			if (cozum2[j][z]==0 && bestczm[j][z]==1)
			{
				u[j][z]+=tpara;
				u[z][j]+=tpara;;
			}

}else if (top<besttop)
```

```
{
        tpara=besttop/top;
        for(j=1;j<N;j++)
                for(z=j+1;z<=N;z++)
                        if (cozum2[j][z]==1 && bestczm[j][z]==0)
                        {
                                u[j][z]+=tpara;
                                u[z][j]+=tpara;
                        }
}

//////////////////////////////////////////
if (besttop>top)
{
        besttop=top;
        for(i=1;i<=N;i++)
                for(j=1;j<=N;j++)
                        bestczm[i][j]=cozum2[i][j];
}


if (g1<(N*10))
{
        g1++;
        lasttop=top;

        for(i=0;i<=N;i++)
        {
                for(j=0;j<=N;j++)
                {
                        cozum2[i][j]=0;
                        k[i][j]=0;
                        ban[i][j]=0;
                }
                use[i]=0;
```

```
                }
                as=0;
                goto greedybas;
        }

        end = clock();
        printf("The time was: %f\n\n", (float) (end - start) / CLK_TCK);

        printf("Sonuc:     %lf\n\n ",besttop);

        for(i=1;i<N;i++)
                for(j=i+1;j<=N;j++)
                        if (bestczm[i][j]==1) fprintf(f2,"%d-%d\n",i,j);
}
```

Ek 2 - Çoklu Gezgin Satıcı Programı için Önerilen TWGA Algoritmasının C Programlama Dili Kodu

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#define n 51
#define mtsp 10

FILE *f;

int czm[n+1][n+1]={0};
int k[n+1][n+1]={0};
int d[n+1]={0};
int use[n+1]={0};

int git[n+1]={0};
int iz[n+1]={0};
int rota[n+1]={0};

int pp=0;
int par[n+1][n+5]={0};

int eij[(n*(n-1)/2)+1][3]={0};
int yij[(n*(n-1)/2)+1][3]={0};
float xy[n+1][2]={0};
float a[n+1][n+1]={0};
float aij[(n*(n-1)/2)+1]={0};
float min,fcost,top1,top2,max,top;

int
i,j,j5,ksay,v1,v2,v3,ps,sakla,sakla1,sakla2,ayritsay,a1,a2,a3,b1,b2,t3,t4,say2,eski,d0say,p2
say;
int
hangi,t1,t2,h1,h2,kp1,kp2,z,sayac,saklayol,saklai,saklaj,i5,sirai,siraj,carpraz,rs,birs,elsay;
```

```
int main()
{

	f=fopen("eil51.txt","r");

	fscanf(f,"%d",&z);
	if (z!=n)
	{
		printf("Boyut uyumsuz ...");
		return 0;
	}
	for(i=1;i<=n;i++)
	{
		fscanf(f,"%d %f %f",&z,&xy[i][0],&xy[i][1]);
	}
	fclose(f);

	z=0;
	for(i=1;i<=n;i++)
		for(j=i+1;j<=n;j++)
		{
			z++;
			eij[z][0]=z;
			eij[z][1]=i;
			eij[z][2]=j;
			aij[z]=sqrt( pow(xy[i][0]-xy[j][0],2) + pow(xy[i][1]-xy[j][1],2) );

			a[i][j]=aij[z];
			a[j][i]=a[i][j];
		}

   clock_t start, end;
   start = clock();
```

```
        for(i=1;i<=n*(n-1)/2;i++)
                for(j=i+1;j<=n*(n-1)/2;j++)
                        if (aij[i] > aij[j])
                        {
                                aij[0]=aij[i];
                                aij[i]=aij[j];
                                aij[j]=aij[0];

                                eij[0][0]=eij[i][0];
                                eij[i][0]=eij[j][0];
                                eij[j][0]=eij[0][0];

                                eij[0][1]=eij[i][1];
                                eij[i][1]=eij[j][1];
                                eij[j][1]=eij[0][1];

                                eij[0][2]=eij[i][2];
                                eij[i][2]=eij[j][2];
                                eij[j][2]=eij[0][2];
                        }

        z=0;
more:
        z++;
        if ((d[eij[z][1]]<2) || (d[eij[z][2]]<2))
  {
                h1=eij[z][1];
                h2=eij[z][2];

                czm[h1][h2]=1;
                czm[h2][h1]=1;

                d[h1]++;
                d[h2]++;
        }
```

```
for(i=1;i<=n;i++)
	if (d[i]<2) goto more;

for(i=z;i>0;i--)
{
	h1=eij[i][1];
	h2=eij[i][2];

	if (czm[h1][h2]==1)
	{
		if (d[h1]>2 && d[h2]>2)
		{
			czm[h1][h2]=0;
			czm[h2][h1]=0;

			d[h1]--;
			d[h2]--;
		}
	}
}

for(i=1;i<=n;i++)
	if (d[i]>2)
	{
		min=10000.0    ;
		for(j=1;j<=n;j++)
			if (czm[i][j]==1 && a[i][j]<min)
			{
				hangi=j;
				min=a[i][j];
			}

		for(j=1;j<=n;j++)
			if (czm[i][j]==1 && hangi!=j)
```

```
                    {
                        t1=i;
                        h1=j;
                        d[t1]--;
                        d[h1]--;
                        czm[t1][h1]=0;
                        czm[h1][t1]=0;
                    }
                }

        //Birlestirme
varmi:
        a3=0;
        for(i=1;i<=n;i++)
                if (d[i]<2)  a3=1;

        d0say=0;
        for(i=1;i<=n;i++)
                if (d[i]==0) d0say++;

        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        if (czm[i][j]==1)
                        {
                                k[i][0]++;
                                k[i][k[i][0]]=j;
                        }
pardvm:

        pp++;
        for(i=1;i<=n;i++)
                if (iz[i]==0 && d[i]!=0)
                {
                        par[pp][0]++;
                        par[pp][par[pp][0]]=i;
```

```
                iz[i]=1;
                goto dvmm;
        }
dvmm:

        for(i=1;i<=par[pp][0];i++)
        {
                z=par[pp][i];
                for(j=1;j<=k[z][0];j++)
                {
                        if(iz[k[z][j]]==0)
                        {
                                iz[k[z][j]]=1;
                                par[pp][0]++;
                                par[pp][par[pp][0]]=k[z][j];
                        }
                }
        }

        for(i=1;i<=n;i++)
                if (iz[i]==0 && d[i]!=0) goto pardvm;


        for(i=1;i<=pp;i++)
        {
                birs=0;
                par[i][par[i][0]+1]=par[i][1];
                for(j=1;j<=par[i][0];j++)
                        {
                                if (d[par[i][j]]==1) birs++;
                        }
                if (!(birs==2 || birs==0)) printf("!!! %d",birs);
                if (birs==2)
                {
                        sayac=0;
```

```
                    for(j=1;j<=par[i][0];j++)
                    {
                        if (d[par[i][j]]==1)
                        {
                            sayac++;
                            if (sayac==1) h1=par[i][j];
                                else h2=par[i][j];
                        }
                    }
                    czm[h1][h2]=1;
                    czm[h2][h1]=1;

                    d[h1]++;
                    d[h2]++;

                    k[h1][0]++;
                    k[h1][k[h1][0]]=h2;

                    k[h2][0]++;
                    k[h2][k[h2][0]]=h1;
                }
            }

            for(i=1;i<=pp;i++)
            {
                if (par[i][0]>3)
                {
                    for(j=0;j<=n;j++)
                        rota[j]=0;

                    rota[0]=par[i][1];
                    rota[1]=par[i][1];
                    j5=0;

sirala:
```

```
                j5++;
                z=rota[j5];
                for(j=1;j<=k[z][0];j++)
                {
                        if (rota[j5-1]!=k[z][j]) rota[j5+1]=k[z][j];
                }
                if (j5<(par[i][0]-1)) goto sirala;

                for(j=1;j<=par[i][0];j++)
                        par[i][j]=rota[j];
        }

}

if (d0say!=0)
{
        for(z=1;z<=n;z++)
                if (d[z]==0)
                {
                        min=10000.0;
                        for(i=1;i<=pp;i++)
                                for(j=1;j<=par[i][0];j++)
                                {
                                        top1=a[par[i][j]][z];
                                        top2=a[par[i][j+1]][z];
                                        top=top1+top2-a[par[i][j]][par[i][j+1]];
                                        if (min>top)
                                        {
                                                min=top;
                                                saklai=i;
                                                saklaj=j;
                                        }
                                }
                        for(i=par[saklai][0]+1;i>saklaj;i--)
                                par[saklai][i+1]=par[saklai][i];
```

```
                                par[saklai][saklaj+1]=z;
                                par[saklai][0]++;
                                d[z]++;
                                d[z]++;
                        }
        }
        p2say=0;
        for(i=1;i<=pp;i++)
                if (par[i][0]==2) p2say++;

enough:

        if (p2say>0)
        {
                min=10000.0;
                for(i=1;i<=pp;i++)
                        if (par[i][0]==2)
                                for(j=1;j<=pp;j++)
                                        if (i!=j)
                                        {

                                                for(j5=1;j5<=par[j][0];j5++)
                                                {
                                                        t1=par[i][1];
                                                        t2=par[i][2];

                                                        t3=par[j][j5];
                                                        t4=par[j][j5+1];

                                                        top1=( a[t1][t3] + a[t2][t4] )-(
a[t1][t2] + a[t3][t4] );
                                                        top2=( a[t1][t4] + a[t2][t3] )-(
a[t1][t2] + a[t3][t4] );

                                                        if (top1<min)
                                                        {
```

```
                                        saklai=i;
                                        saklaj=j;
                                        siraj=j5;
                                        carpraz=1;
                                        min=top1;
                                }
                                if (top2<min)
                                {
                                        saklai=i;
                                        saklaj=j;
                                        siraj=j5;
                                        carpraz=0;
                                        min=top2;
                                }
                        }
                }

for(i=par[saklaj][0]+2;i>siraj;i--)
                        par[saklaj][i+2]=par[saklaj][i];

if (carpraz==1)
{
        par[saklaj][siraj+1]=par[saklai][1];
        par[saklaj][siraj+2]=par[saklai][2];
}else{
        par[saklaj][siraj+1]=par[saklai][2];
        par[saklaj][siraj+2]=par[saklai][1];
}
par[saklaj][0]++;
par[saklaj][0]++;
if (saklai!=pp)
{
        for(i=0;i<=par[pp][0]+1;i++)
                par[saklai][i]=par[pp][i];
}
```

```
                if (saklai!=pp)
                {
                        for(i=0;i<=par[pp][0];i++)
                                par[saklai][i]=par[pp][i];
                }
                pp--;

                for(i=1;i<=pp;i++)
                        if (par[i][0]==2) goto enough;
        }

inc:
        if(pp<mtsp)
        {
                min=10000;
                for(i5=1;i5<=pp;i5++)
                        if (par[i5][0]>5)
                        {
                                z=0;
                                for(j5=1;j5<=par[i5][0];j5++)
                                {
                                        z++;
                                        yij[z][0]=z;
                                        yij[z][1]=par[i5][j5];
                                        yij[z][2]=par[i5][j5+1];
                                }

                                for(i=1;i<z;i++)
                                        for(j=i+1;j<=z;j++)
                                        {
                                                a1=yij[i][1];
                                                a2=yij[i][2];
                                                b1=yij[j][1];
                                                b2=yij[j][2];
```

```
                                        if (a1!=b1 && a1!=b2 && a2!=b1 &&
a2!=b2)
                                                if (czm[a1][b1]!=1 &&
czm[a1][b2]!=1 && czm[a2][b1]!=1 && czm[a2][b2]!=1)
                                                        {
        top1=a[a1][a2]+a[b1][b2];
        top2=a[a1][b2]+a[a2][b1];
                                                                top=top2-top1;
                                                                if (top<min)
                                                                {
                                                                        min=top;
        saklayol=i5;
        sakla1=yij[i][0];
        sakla2=yij[j][0];
                                                                }
                                                        }
                                        }
                }
        if (min==10000) printf("\n\nBoyut satici sayisi icin uygun degil!\n\n");
        else
        {
                pp++;
                par[pp][0]=0;
                for(i=sakla1+1;i<=sakla2;i++)
                {
                        par[pp][0]++;
                        par[pp][par[pp][0]]=par[saklayol][i];
                }
                        par[pp][par[pp][0]+1]=par[pp][1];
```

```
                    for(i=1;i<=(par[saklayol][0]-sakla2);i++)
                    {
                        par[saklayol][sakla1+i]=par[saklayol][sakla2+i];
                    }
                    par[saklayol][0]=par[saklayol][0]-par[pp][0];
                    par[saklayol][par[saklayol][0]+1]=par[saklayol][1];
                    goto inc;
                }
            }

sonkez:
            fcost=0.0;
            elsay=0;
            ayritsay=0;
            for(j=1;j<=pp;j++)
            {
                elsay=elsay+par[j][0];
                for(i=1;i<=par[j][0];i++)
                {
                    fcost+=a[par[j][i]][par[j][i+1]];
                    ayritsay++;
                }
            }
            printf("\nParca Sayisi: %d Eleman Sayisi: %d,%d COST:
%f",pp,ayritsay,elsay,fcost);

            min=100000.0;

            if (pp>1)
            {
                for(i=1;i<pp;i++)
                for(j=i+1;j<=pp;j++)
                {
                    for(i5=1;i5<=par[i][0];i5++)
                        for(j5=1;j5<=par[j][0];j5++)
                        {
```

```
                                        t1=par[i][i5];
                                        t2=par[i][i5+1];

                                        t3=par[j][j5];
                                        t4=par[j][j5+1];

                                        top1=( a[t1][t3] + a[t2][t4] )-( a[t1][t2] + a[t3][t4]
);
                                        top2=( a[t1][t4] + a[t2][t3] )-( a[t1][t2] + a[t3][t4]
);
                                        if (top1<min)
                                        {
                                                saklai=i;
                                                sirai=i5;
                                                saklaj=j;
                                                siraj=j5;
                                                carpraz=1;
                                                min=top1;
                                        }
                                        if (top2<min)
                                        {
                                                saklai=i;
                                                sirai=i5;
                                                saklaj=j;
                                                siraj=j5;
                                                carpraz=0;
                                                min=top2;
                                        }
                                }
                        }

                        if (carpraz==0)
                        {
                                rota[0]=par[saklaj][0];
                                rs=0;
                                j=siraj;
```

```
rotdev0:
                    if (j==par[saklaj][0]) j=0;
                    j++;
                    rs++;
                    rota[rs]=par[saklaj][j];

                    if (rs<par[saklaj][0]) goto rotdev0;
                }else
                {
                    rota[0]=par[saklaj][0];
                    rs=0;
                    j=siraj+1;
rotdev1:
                    j--;
                    rs++;
                    rota[rs]=par[saklaj][j];
                    if (j==1) j=par[saklaj][0]+1;
                    if (rs<par[saklaj][0]) goto rotdev1;
                }

                for(i=par[saklai][0];i>sirai;i--)
                    par[saklai][i+rota[0]]=par[saklai][i];

                for(i=1;i<=rota[0];i++)
                    par[saklai][sirai+i]=rota[i];

                par[saklai][0]+=rota[0];
                par[saklai][par[saklai][0]+1]=par[saklai][1];

                if (saklaj!=pp)
                {
                    for(i=0;i<=par[pp][0];i++)
                        par[saklaj][i]=par[pp][i];

                    par[saklaj][par[saklaj][0]+1]=par[saklaj][1];
```

```
            }
            pp--;
            if (pp>1) goto sonkez;
        }

        end = clock();
        printf("\n\nThe time was: %f\n\n", (float) (end - start) / CLK_TCK);

        ayritsay=0;
        fcost=0.0;
        for(i=1;i<=par[1][0];i++)
        {
            printf("%d ",par[1][i]);
            use[par[1][i]]=1;
            fcost+=a[par[1][i]][par[1][i+1]];
            ayritsay++;
        }
        printf("\nCOST: %d %f",par[1][0],fcost);

}
```