KARADENİZ TEKNİK ÜNİVERSİTESİ * SOSYAL BİLİMLER ENSTİTÜSÜ

EKONOMETRİ ANABİLİM DALI

EKONOMETRİ PROGRAMI

TOPLU TAŞIMA AĞLARI ÜZERİNDE GÜZERGÂH OPTİMİZASYONU PROBLEMİNE EN AZ AKTARIM ODAKLI BİR ÇÖZÜM YAKLAŞIMI

DOKTORA TEZİ

ALİ AKAY

TEMMUZ 2015

TRABZON

KARADENİZ TEKNİK ÜNİVERSİTESİ * SOSYAL BİLİMLER ENSTİTÜSÜ

EKONOMETRİ ANABİLİM DALI

EKONOMETRİ PROGRAMI

TOPLU TAŞIMA AĞLARI ÜZERİNDE GÜZERGÂH OPTİMİZASYONU PROBLEMİNE EN AZ AKTARIM ODAKLI BİR ÇÖZÜM YAKLAŞIMI

DOKTORA TEZİ

ALİ AKAY

Tez Danışmanı: Doç Dr. Tuba YAKICI AYAN

TEMMUZ 2015

TRABZON

ONAY

Ali AKAY tarafından hazırlanan Toplu Taşıma Ağları Üzerinde Güzergâh Optimizasyonu Problemine En Az Aktarım Odaklı Bir Çözüm Yaklaşımı adlı bu çalışma __ / __ / 2015 tarihinde yapılan savunma sınavı sonucunda (oybirliği /oyçokluğu) ile başarılı bulunarak jürimiz tarafından EKONOMETRİ ANABİLİM DALI ‘ında doktora tezi olarak kabul edilmiştir

[i m z a]
[Un vanı Adı ve Soyadı]

[i m z a]
[Un vanı Adı ve Soyadı]

[i m z a]
[Un vanı Adı ve Soyadı]

[i m z a]
[Un vanı Adı ve Soyadı]

[i m z a]
[Un vanı Adı ve Soyadı]

Yukarıdaki imzaların, adı geçen öğretim üyelerine ait olduklarını onaylarım. __ / __ / 2015

Prof. Dr. Ahmet ULUSOY
Enstitü Müdürü

BİLDİRİM

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını, aksinin ortaya çıkması durumunda her tür yasal sonucu kabul ettiğimi beyan ediyorum.

Ali AKAY

26.05.2015

ÖNSÖZ

Son zamanlarda hızla yaygınlaşmakta olan yolculuk planlama sistemlerinin, optimizasyon yöntemlerinin konu edildiği bu çalışmada, sistemlerin daha kaliteli çözümleri daha hızlı üretebilmelerine yönelik yeni yaklaşımlar geliştirilmiştir. Çalışma, akademik danışmanlığını yürüttüğüm, "Toplu Ulaşım Tercihlerinde Karar Destek Sistemi" projesi kapsamında, Ar-Ge, İnovasyon ve Endüstriyel Uygulama Destek Programı çerçevesinde KOSGEB tarafından desteklenmiştir (Proje No:2012-22).

Çalışmada geliştirilen yöntemlerin, girişimciler ve ilgili otoriteler tarafından da takdir edilerek son kullanıcıların hizmetine sunulması ve böylece akademik alandaki katkısının yanında, sosyal bir fayda sağlaması temennisi ile çalışmaya destek veren Kocaeli KOSGEB müdürü Ertuğrul ÇETİNKAYA'ya, KOSGEB projemizi destekleyen BURULAŞ YBS Koordinatörü Alper GÜVERCİN'e ve tüm BURULAŞ çalışanlarına, ihtiyaç duyduğumuzda teknik imkânlarını ve tecrübelerini esirgemeyen İstanbul Büyükşehir Belediyesi Bilgi İşlem Dairesi Başkanı Hakkı TOK'a, Bilgi İşlem Şube Müdürü Harun KAYA'ya ve İstanbul Büyükşehir Belediyesine, randevu talebimizi geri çevirmeyip bizi ağırlama nezaketinde bulunan manevi desteğini ve tecrübelerini esirgemeyen Ulaştırma Bakanlığı Strateji Geliştirme Başkanlığı Başkan Vekili Erol YANAR'a teşekkürlerimi sunarım.

Proje fikrini üretip bu yola çıkan ve bu çalışmanın hazırlanmasına vesile olan yol arkadaşlarım ve kardeşlerim, Teknar Tekloji Ltd. Şti'nin sahipleri, Gökhan AKYOL ve Eren KÜÇÜK'e ayrıca teşekkür ederim.

Ve elbette, akademik kariyerimi borçlu olduğum, bilgisi ve tecrübesiyle her zaman yanımda hissettiğim, çalışma boyunca beni daima motive eden ve destek olan değerli hocam Doç. Dr. Tuba Yakıcı AYAN'a şükran ve minnetlerimi arz ederim.

Son ve özel olarak, çalışma süresince zaman zaman ihmal ettiğim, yine de desteklerini ve dualarını hiçbir zaman eksik etmeyen, zor anlarımda hep yanımda olan, bu çalışmanın başarısında –varsa- şahsıma düşen payın asıl sahipleri, sevgili eşim Fatma'ya, oğlum Mirza Hasancan'a ve minik kızım Feyza'a sonsuz şükranlarımı sunarım.

Temmuz 2015 Ali AKAY

İÇİNDEKİLER

İKİNCİ BÖLÜM

ÜÇÜNCÜ BÖLÜM

ÖZET

Bu çalışmada, toplu ulaşım ağları üzerinde yolculuk planlama problemi için yeni bir algoritma geliştirilmiştir. TUR olarak adlandırılan yeni algoritma, problemi en az aktarımlı çözümler üretmeye odaklanarak ele almakta ve bir yolculuğu aktarım sayısı ile birlikte varış zamanına göre optimize etmektedir. TUR algoritmasında, literatürdeki CSA ve RAPTOR yöntemlerinde olduğu gibi ağ, bir graf olarak değil, algoritma için tasarlanan özel veri yapılarıyla ele alınmıştır. Londra, İstanbul, Ankara, İzmir ve Bursa şehirleri toplu ulaşım verileri kullanılarak üretilen algoritma çözümleri, gerçek dünya uygulamalarıyla karşılaştırılmıştır. Karşılaştırma sonuçları, algoritmanın gerçek dünya uygulamalarıyla benzer sonuçlar ürettiğini göstermiştir. Çözüm karşılaştırmalarının ardından TUR algoritmasının sorgulama performansı test edilmiş ve algoritmanın, aynı optimizasyon kriterlerini kullanan ön işlem adımsız algoritmalardan yaklaşık 4 kat daha hızlı olduğu görülmüştür.

Anahtar Kelimeler: TUR, en kısa yol algoritmaları, toplu ulaşım, yolculuk planlama

ABSTRACT

In this study, a new algorithm is developed for trip planning problem on public transport networks. The new algorithm named TUR, addresses the problem focusing on the minimum transfer count and optimize a trip using transfer count and arrival time criterions together. In TUR, network is treated not as a graph but with custom-built data structures that specially designed for the algorithm, as in RAPTOR and CSA. Generated solutions using public transit networks of London, İstanbul, Ankara, İzmir and Bursa are compared with real-world applications. The comparison results show that the algorithm has been generated similar solutions to real-world applications. After the solution comparisons, query performance of TUR algorithm is tested and shown that the algorithm is faster about 4 times then algorithms has no pre-computation and uses same optimization criterions.

Key Words: TUR, shortest path algorithms, public transport, trip planning

TABLOLAR LİSTESİ

ŞEKİLLER LİSTESİ

KISALTMALAR LİSTESİ

ACSA	: Hızlandırılmış Bağlantı Tarama Algoritması (Accelerated Link Scanning Algorithm)
AHA	: Anayol Hiyerarşileri Algoritması
AS	: Aktarım Sayısı
BDL	: Başlangıç Durakları Listesi
BDZ	: Başlangıç Zamanı Değişkeni
CSA	: Bağlantı Tarama Algoritması
ÇAL	: Çözüm Adımları Listesi
DHA	: Daralma Hiyerarşileri Algoritması
EBGAS	: En Büyük Genel Aktarım Sayısı
EKY	: En Kısa Yol
HDL	: Hareket Durakları Listesi
HHZ	: Hat Hareket Zamanı
HVZ	: Hat Varış Zamanı
HVD	: Hat Varış Durağı
HHDVZ	: Hat Hareket Durağına Varış Zamanı
GAS	: Genel Aktarım Sayısı
GDA	: Geçiş Düğümleri Algoritması
GKA	: Geometrik Kalıplar Algoritması
KBA	: Kenar Bayrakları Algoritması
KEKY	: K En Kısa Yol
LD	: Katmanlı Dijkstra Algoritması (ingilizce: Layered Dijkstra Algorithm)
POI	: Önemli Nokta - Point of Interest
SN	: Sefer Numarası
T. Patterns	: Transfer Pattern
TN	: Transfer Noktası
TUA	: Toplu Ulaşım Ağı
VD	: Varış Durakları
VDGH	: Varış Duraklarından Geçen Hatlar

ZGY : Zaman Genişletme Yaklaşımı

ZBY : Zaman Bağımlı Yaklaşım

TUR : Toplu Ulaşım Rehberi

GİRİŞ

Toplu ulaştırma hizmetleri, özellikle hava kirliliği ve trafik yoğunluğu açılarından önem arz etmekte ve şehirlerdeki yaşam kalitesini ciddi düzeyde etkilemektedir. Belediyeler, şehirlerde yaşanan trafik yoğunluğunun azaltılmasına katkı sağlamak amacıyla, büyük yatırımlar yaparak, şehrin toplu ulaşım seçeneklerini çeşitlendirmeye çalışmaktadırlar. Böylece toplu ulaşım ağlarının (TUA) boyutu hızla artmakta ve ağ yapıları günden güne daha karmaşık bir hale dönüşmektedir.

İstanbul gibi yoğun nüfuslu kentlerde TUA'ların çok büyük olması sebebiyle toplu ulaştırma hizmetlerinin etkin bir biçimde kullanılabilmesi için ciddi miktarda bilgiye gerek duyulmaktadır. Çünkü bu tür bir TUA, binlerce duraktan ve yüzlerce hattan oluşan büyük bir ağdır. Özellikle tek hat kullanılarak seyahat edilemeyen güzergâhlarda, bir yolculuğun etkin bir şekilde planlanması, sıradan bir vatandaş için neredeyse imkânsızdır. Bu sebeple büyük şehirlerde toplu ulaştırma hizmetleri etkin olarak kullanılamamakta ve yapılan yatırımların geri dönüşü sınırlı kalmaktadır.

Son yıllarda, toplu ulaştırma hizmetlerinin kullanım etkinliğinin artırılması amacıyla, çeşitli çalışmalar yapılmaktadır. Yolcuları seyahat hakkında bilgilendiren araç içi seslendirme sistemleri, araçların duraklara ne kadar süre sonra geleceğini gösteren akıllı durak uygulamaları gibi çalışmalara yakın bir zamanda, otomobiller için uzun zamandır kullanımda olan yol tarifi (navigasyon) sistemlerine benzer bir hizmeti, TUA üzerinden sunan uygulamalar eklenmiştir. Genel olarak yolculuk planlama sistemleri olarak adlandırılabilecek bu sistemler, başlangıçta belirli bir başlangıç durağından belirli bir noktaya hangi hatlarla gidilebileceğinin kullanıcıya sunulmasına yönelik olarak tasarlanmıştır. Son birkaç yılda ise uygulamalar, tüm yolculuğu belli bir hareket ya da varış saatine göre planlayabilen sistemler haline dönüşmüştür.

Yolculuk planlama sistemleri, bu alanda yapılan başarılı akademik çalışmaların da desteğiyle, her geçen gün gelişmekte ve sayıları hızla artmaktadır. Başta Amerika, Avrupa ülkeleri ve Avustralya olmak üzere, birçok ülkede bu sistemler, büyük teknoloji şirketleri

ve devlet tarafından desteklenmekte ve vatandaşlar tarafından yoğun bir şekilde kullanılmaktadır. Londra Belediyesi tarafından geliştirilen TFL (Transport for London) uygulaması bu alandaki en başarılı uygulamalardan biri olarak gösterilmektedir. Ayrıca Google, Microsoft, Yahoo ve Yandex gibi teknoloji devi şirketler, dijital harita uygulamalarında, dünya çapındaki birçok şehir için seyahat planlama hizmeti sunmaktadır.

Ülkemizde yolculuk planlama sistemleri yeni yeni gelişmektedir. Özellikle yurtdışındaki başarılı uygulamalar, konuyla ilgili otoritelerin dikkatini çekmiş ve son beş yıldır bu alandaki yatırımlar belediyeler nezdinde artış göstermiştir. Bununla birlikte, Türkiye'de hali hazırda yolculuk planlama sistemi olarak nitelenebilecek çok az sayıda uygulama vardır ve bunlardan birkaçı da yurtdışı kökenlidir. İstanbul, Ankara, İzmir, Bursa, Kayseri ve Şanlıurfa büyükşehir belediyelerine ait olan yolculuk planlama sistemleri ulusal olarak hizmet sunan sistemlerdir. Bursa ve Şanlıurfa sistemleri, KOSGEB desteği ile bu tez çalışmasında geliştirilen TUR algoritmasının ilk versiyonunu kullanmaktadır. Uygulamalara belediyelerin resmi web sitelerinden erişilebilir. BuradanOraya[1] ve Trafi[2] uygulamaları ise yabancı girişimcilerin Türkiye için geliştirdikleri yolculuk planlama sistemleridir. Ayrıca, Ulaştırma Bakanlığı nezdinde yürütülen ve Türksat tarafından geliştirilen Ulusal Ulaştırma Portalı[3] projesi de, şehirden şehire yolculuk planlaması yapan bir sistem olarak ulusal sistemler arasında sayılabilir.

TUA'larda yolculuk planlama problemine ilişkin akademik çalışmalar, bu alandaki uygulamalara paralel olarak son on yılda dünya genelinde hızlı bir artış göstermiştir. Başlangıçta problemi klasik bir en kısa yol problemi olarak ele alan araştırmacılar, son yıllarda yeni bir bakış açısı ile, TUA'lara özgü yeni yaklaşımlar geliştirmişlerdir. Türkiye'de ise bu alandaki akademik çalışmaların sayısı yok denecek kadar azdır. Bu tez çalışması, alanda yapılan ulusal akademik çalışmalar arasındaki ilk yöntem çalışmasıdır.

Problem, başlangıçta klasik en kısa yol problemine özgü yöntemlerle çözülmeye çalışıldığı için graf teorisi kavramları kullanılarak tanımlanmıştır. Bu sebeple çalışmanın birinci bölümünde, graf teorisine ilişkin temel kavramlar verildikten sonra en kısa yol problemi tanımlanmış ve bu probleme ilişkin literatürdeki çözüm yöntemleri bugüne kadar

[1] http://www.buradanoraya.com/
[2] http://www.trafi.com
[3] http://ulasim.gov.tr/

yapılan geliştirme çalışmalarıyla birlikte kronolojik bir sırada anlatılmıştır. Ardından TUA'larda yolculuk planlama problemine geçilerek, probleme özgü geliştirilen yöntem ve bu yöntemlere ilişkin karşılaştırma çalışmaları, yine kronolojik sıralama gözetilerek aktarılmıştır.

Çalışmanın ikinci bölümünde ise TUA'larda yolculuk planlama problemi için bu tez çalışması kapsamında geliştirilen ve TUR olarak adlandırılan algoritma ve ayrıntılı olarak açıklanmıştır. Üç adımdan oluşan algoritmanın her adımı, tüm alt işlem adımlarıyla birlikte ele alınarak basit bir örnek TUA üzerinden işletilmiş ve sonuçlar tablolar halinde gösterilmiştir. Bu bölümde ayrıca, TUA'larda yolculuk planlama problemi birçok farklı yönden yeniden ele alınarak, probleme özgü çözüm yöntemleri geliştirilirken göz önünde bulundurulması gereken hususlara dikkat çekilmiştir. Özellikle çözümlerde kullanılacak aktarım sayısının, yöntemlerin çözüm kalitesi üzerindeki etkisi vurgulanmıştır. Bu bağlamda en az aktarım içeren çözümlerin hesaplanmasına yönelik olarak çalışma kapsamında geliştirilen yeni bir yaklaşım olan "ters akım algoritması" ayrı bir başlık olarak ele alınmıştır.

Üçüncü bölümde, TUR algoritması mevcut sistem ve yöntemlerle karşılaştırılarak, algoritmanın, problemin çözümündeki başarısı test edilmiştir. Bu amaçla iki aşamalı bir karşılaştırma süreci takip edilmiştir. Birinci aşamada TUR algoritmasını kullanarak TUA'lar üzerinde yolculuk planlaması yapan bir uygulama geliştirilmiş ve algoritma çözümlerinin gerçek dünya uygulamalarıyla karşılaştırılması sağlanmıştır. TUR uygulaması ile çeşitli internet sitelerinden ve benzer uygulamalardan derlenen İstanbul, Ankara, İzmir ve Bursa illeri TUA verileri ile her bir TUA için rastlantısal 10 durak çifti arasında yolculuk planlaması yapılmıştır. Bu yolla üretilen çözümler, aynı illerde, yolculuk planlama hizmeti sunan Trafi uygulaması çözümleri ile karşılaştırılmıştır.

Karşılaştırma ve testin ikinci aşamasında TUR algoritması çözüm üretme performansı, literatürdeki yöntemlerin performansları ile karşılaştırılarak sonuçlar tablolar halinde sunulmuştur. Bu noktada TUR uygulaması ile rastgele seçilen 10000 durak çifti için çözümler hesaplanmış ve algoritmanın ortalama çözüm üretme süresi belirlenmiştir. Ardından bu süre literatürdeki yöntemlerin performans sonuçları ile karşılaştırılmıştır.

Literatürdeki yöntemler genellikle Londra şehri TUA'sını kullandığından TUR uygulaması performansı da bu TUA verileri ile elde edilmiştir.

BİRİNCİ BÖLÜM

1. TEMEL KAVRAMLAR

1.1. Graf Teorisi

1.1.1. Graf

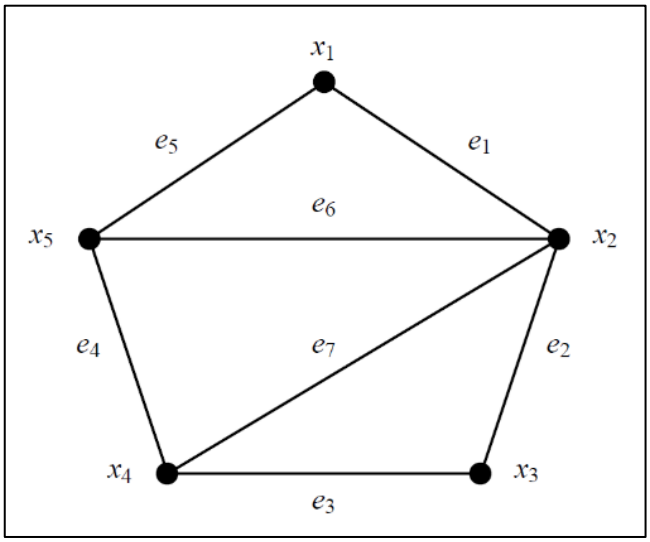
Biçimsel olarak graf, birbirleriyle bağlantılı düğümler ve bu düğümler arasındaki bağlantıları tanımlayan kenarlardan oluşan bir diyagram olarak tanımlanabilir (Rouhonen 2013 s. 1). Kavramsal olarak ise bir G grafı;

V: Düğümler (Vertices) kümesi

E: İkililerden oluşan kenarlar (Edges) kümesi olmak üzere

G = (V, E) şeklinde tanımlanır (Joyner ve diğerleri, 2013: 3; Diestel, 2005: 2, Rouhonen, 2013: 1, Voloshin, 2009: 2 ve Harju, 2011: 4).

Şekil 1: Örnek Bir Graf



Kaynak: Voloshin, 2009: 1.

Örnek olarak Şekil 1'deki graf şu şekilde tanımlanabilir:

$V = \{ x_1, x_2, x_3, x_4, x_5\}$, $E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$, $e_1 = (x_1, x_2)$, $e_2 = (x_2, x_3)$, $e_3 = (x_3, x_4)$, $e4 = (x_4, x_5)$, $e_5 = (x_1, x_5)$, $e_6 = (x_2, x_5)$, $e_7 = (x_2, x_4)$

1.1.1. Komşuluk ve Ayrıklık

Bir düğüm çiftini birbirine bağlayan bir kenar var ise bu iki düğüme komşu düğümler denir. Aksi durumda bu iki düğüm ayrıktır.

$(x_1, x_2) \in E$ ise x_1 ve x_2 komşu düğümlerdir; $(x_1, x_2) \notin E$ ise x_1 ve x_2 ayrık düğümlerdir.

1.1.2. Komşuluk Derecesi, Komşuluk Fonksiyonu ve Komşuluk Matrisi

G = (V, E) şeklinde tanımlanan bir graf olsun. G grafındaki bir düğümün komşuluk derecesi, bu düğüme komşu olan düğüm sayısı şeklinde tanımlanır ve d(x) ile gösterilir. Örneğin Şekil 1'deki x_4 düğümü için $d(x_4) = 3$'tür.

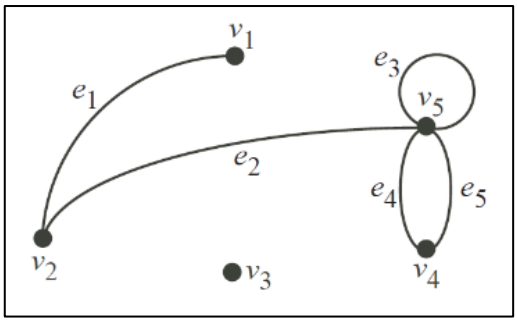
Bir grafın maksimum derecesi ise max d(x) olarak tanımlanır ve en büyük komşuluk derecesine sahip olan düğümün derecesine eşittir. Örneğin Şekil 1'in maksimum derecesi $d(x_2) = 4$'tür.

Komşuluk fonksiyonu ise bir graftaki bir düğümün komşu düğümlerinin kümesini veren fonksiyon olarak tanımlanır ve N(x) şeklinde gösterilir. Örnek olarak Şekil 1'deki x_4 düğümü için $N(x_4) = \{x_2, x_3, x_5\}$'dir. Bazı kaynaklarda ise komşuluk fonksiyonu yerine komşuluk matrisi tanımlanmaktadır (Steen, 2010: 2-14). Komşuluk matrisi; m düğüm sayısı olmak üzere m*m boyutunda bir A matrisi ile gösterilir. A matrisinin, a_{ij} elemanı, i. düğüm ile j. düğüm arasında bir bağlantı varsa 1 aksi halde 0 olacak şekilde oluşturulur (Bondy ve Murty, 1982: 7).

1.1.3. Paralel Kenarlar ve Döngüler

Aynı iki düğüm arasında birden fazla kenar var ise bu kenarlara paralel kenarlar (parallel edges) denir. Bir düğümden yine kendisine tanımlanan e = (x, x) şeklindeki kenarlar ise döngü (loop) olarak adlandırılır.

Şekil 2: Paralel Kenarlar ve Döngüler



Kaynak: Rouhonen, 2013: s. 2.

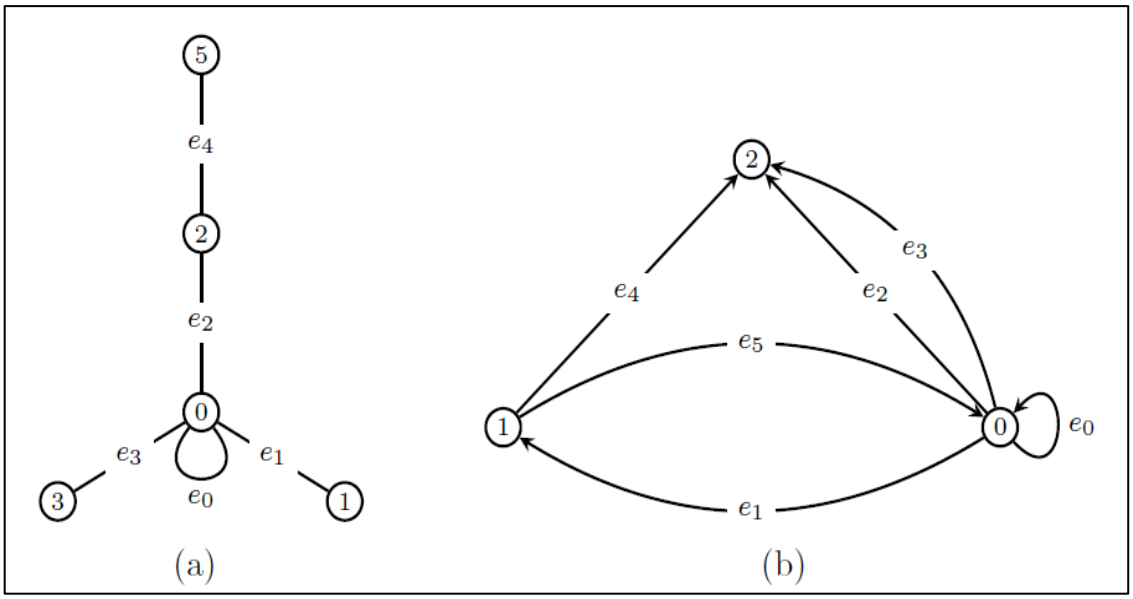
Şekil 2'de e_4 ve e_5 paralel kenarları; e_3 ise bir döngüyü göstermektedir.

1.1.4. Yönsüz, Yönlü ve Ağırlıklı Graflar

İki düğüm arasındaki bağlantıları tanımlayan e_i= (x, y) kenarlarında sıralamanın önemli olmadığı durumlarda e_i = (x, y) ve e_j = (y, x) aynı kenarı ifade eder. Bu tür graflara yönsüz graflar denir. Yönlü graflarda ise x düğümünden y düğümüne bağlantı var iken y düğümünden x düğümüne bağlantı tanımlı olmayabilir. Bu durum, özellikle kenarları için iki düğüm arasındaki maliyeti ifade eden ağırlıklar atanmış olan ağırlıklı (weighted) graflarda önem kazanır.

Şekil 3'teki (a) grafı yönsüz ağırlıklı; b grafı ise yönlü ağırlıklı grafı göstermektedir.

Şekil 3: Yönlü ve Yönsüz Ağırlıklı Graflar



Kaynak: Joyner ve diğerleri, 2013: 6.

1.2. En Kısa Yol Problemi

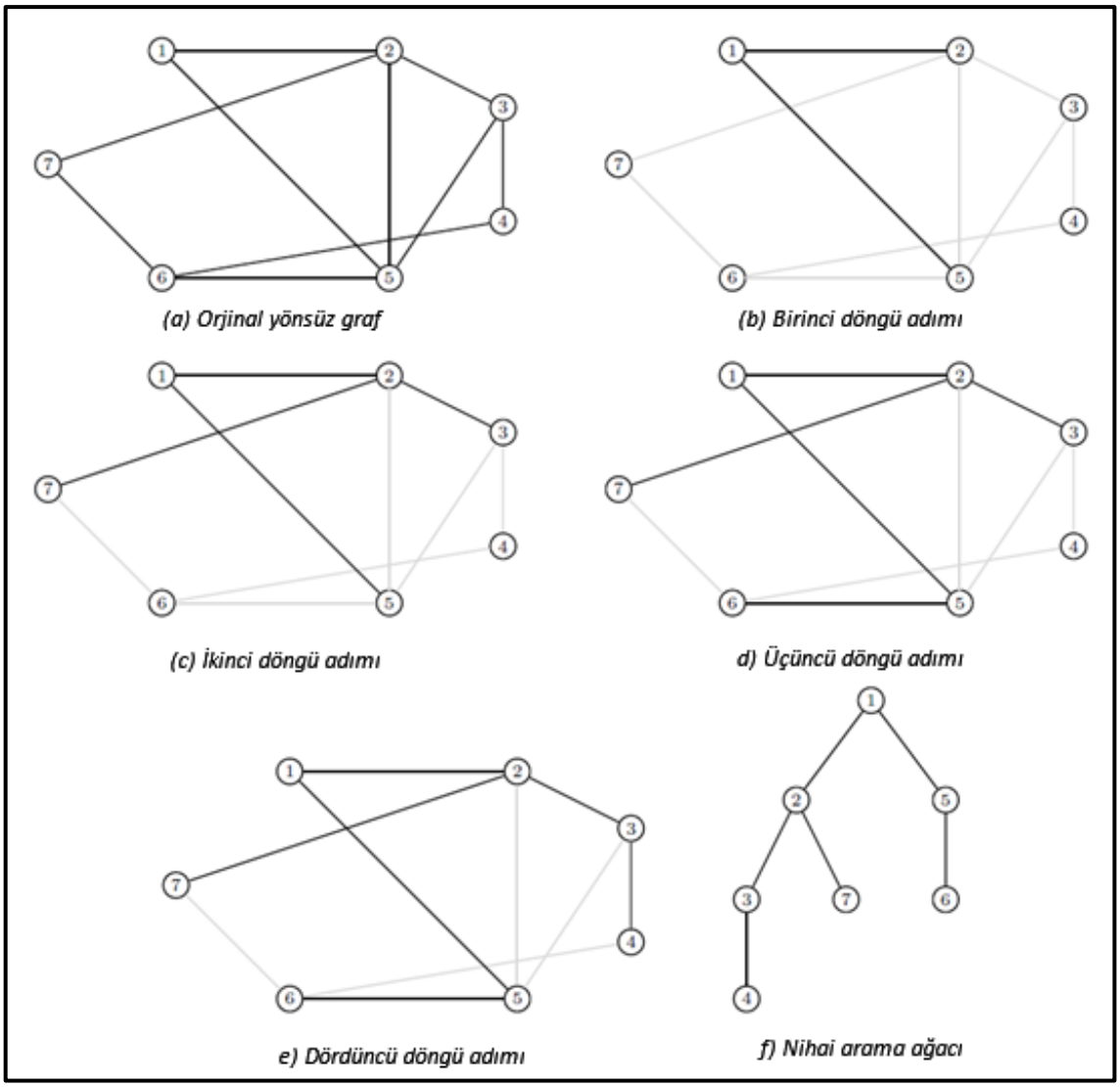
En kısa yol (EKY) problemleri, graf teorisinin en önemli uygulama alanlarından biridir. Problem, ağırlıklı bir graf üzerinde iki düğüm arasındaki toplam ağırlığı en düşük olan bağlantıları bulmayı amaçlayan bir optimizasyon problemidir.

En kısa yol algoritmalarına geçmeden önce ağırlıksız ağlar üzerinde yol bulma için tasarlanmış Genişlik Öncelikli (Breadth-First) ve Derinlik Öncelikli (Dept-First) arama algoritmalarına değinmek faydalı olacaktır.

1.2.1. Genişlik Öncelikli Arama

Bu algoritma graf üzerinde önceden belirlenen bir düğümden başlayarak, bu düğümün komşulukları üzerinden diğer tüm düğümlere en az kaç adımda ulaşılabileceğini hesaplar. Algoritmanın temel kriteri bir düğüme yalnız bir kere uğranılmasıdır.

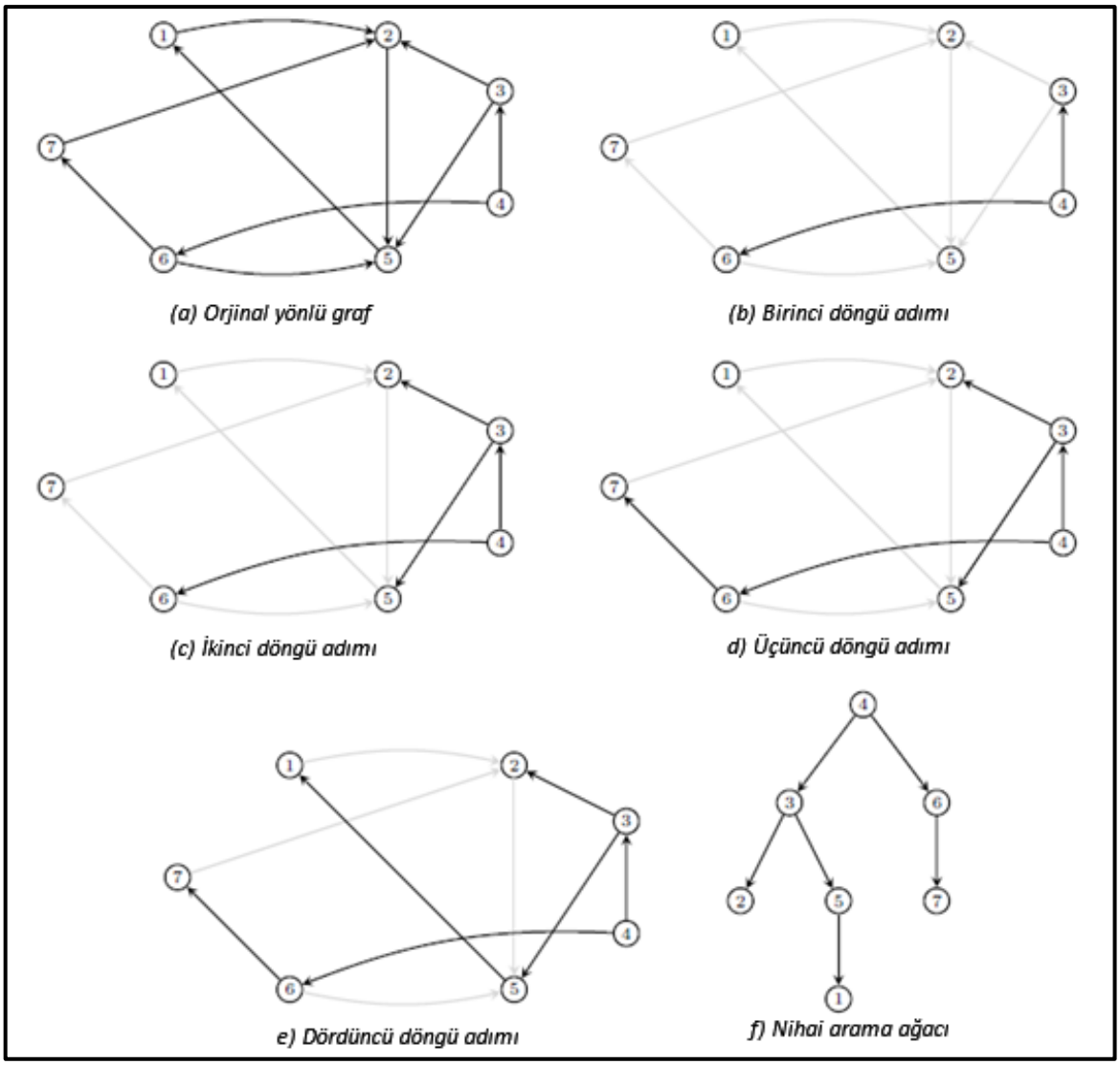
Şekil 4: Yönsüz Graf için Genişlik Öncelikli Arama Algoritması Adımları



Kaynak: Joyner ve diğerleri, 2013: 111.

Genişlik öncelikli arama algoritmasının yönsüz ve yönlü graflardaki uygulama adımları sırasıyla Şekil 4 ve Şekil 5'de gösterilmiştir.

Şekil 5: Yönlü Graf için Genişlik Öncelikli Arama Algoritması Adımları



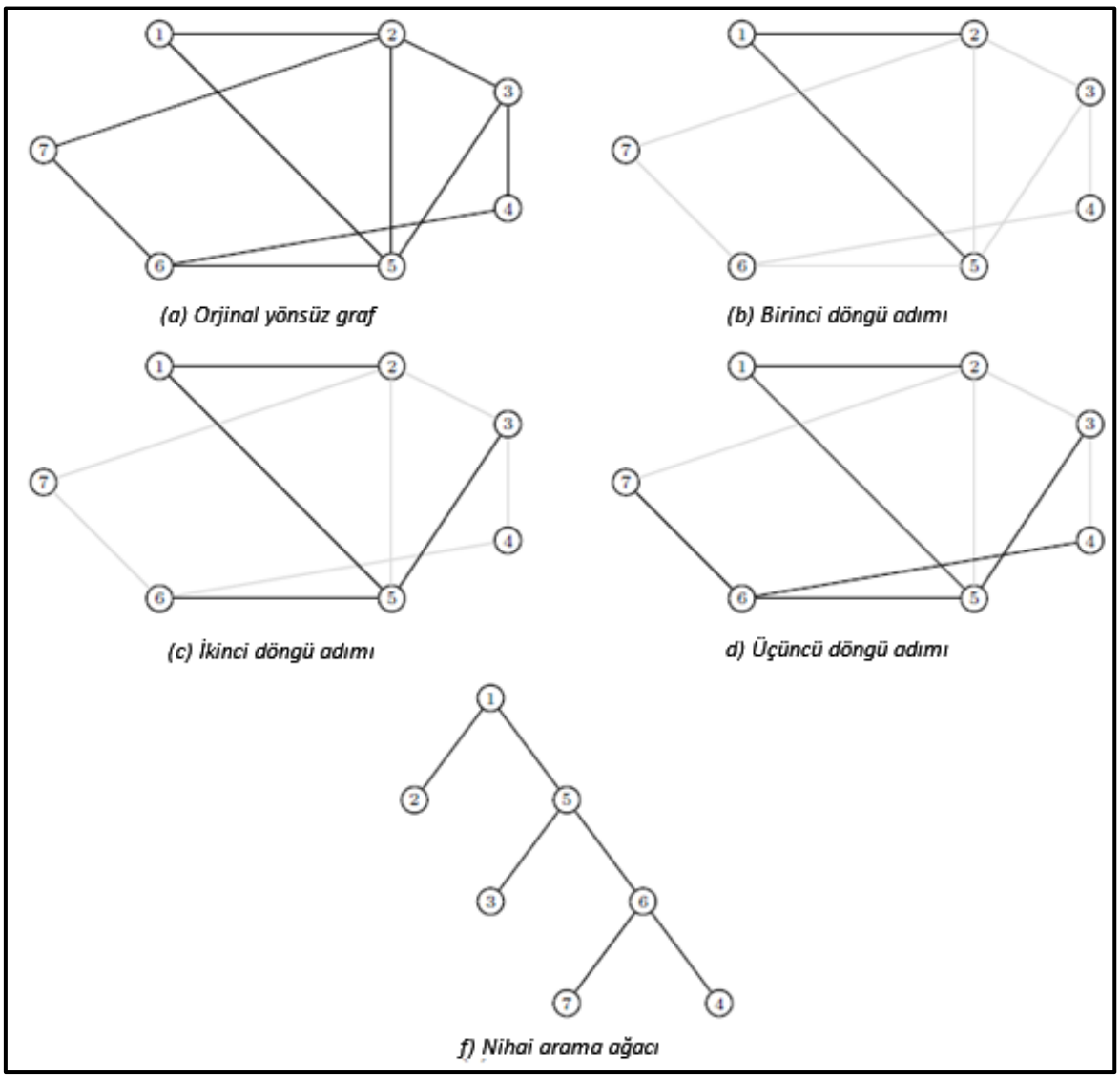
Kaynak: Joyner ve diğerleri, 2013: 112.

1.2.2. Derinlik Öncelikli Arama

Derinlik öncelikli arama algoritması da genişlik öncelikli algoritması gibi bir başlangıç düğümünden diğer tüm düğümlere en az adımla ulaşmayı hedefler. Ancak bu iki algoritmanın uygulama adımları birbirinden farklıdır. Genişlik öncelikli arama algoritmasında iterasyon başlangıç düğümünden yan dallara doğru ilerlerken derinlik öncelikli arama algoritmasında ilerleme başlangıç düğümünden alt dallara doğrudur.

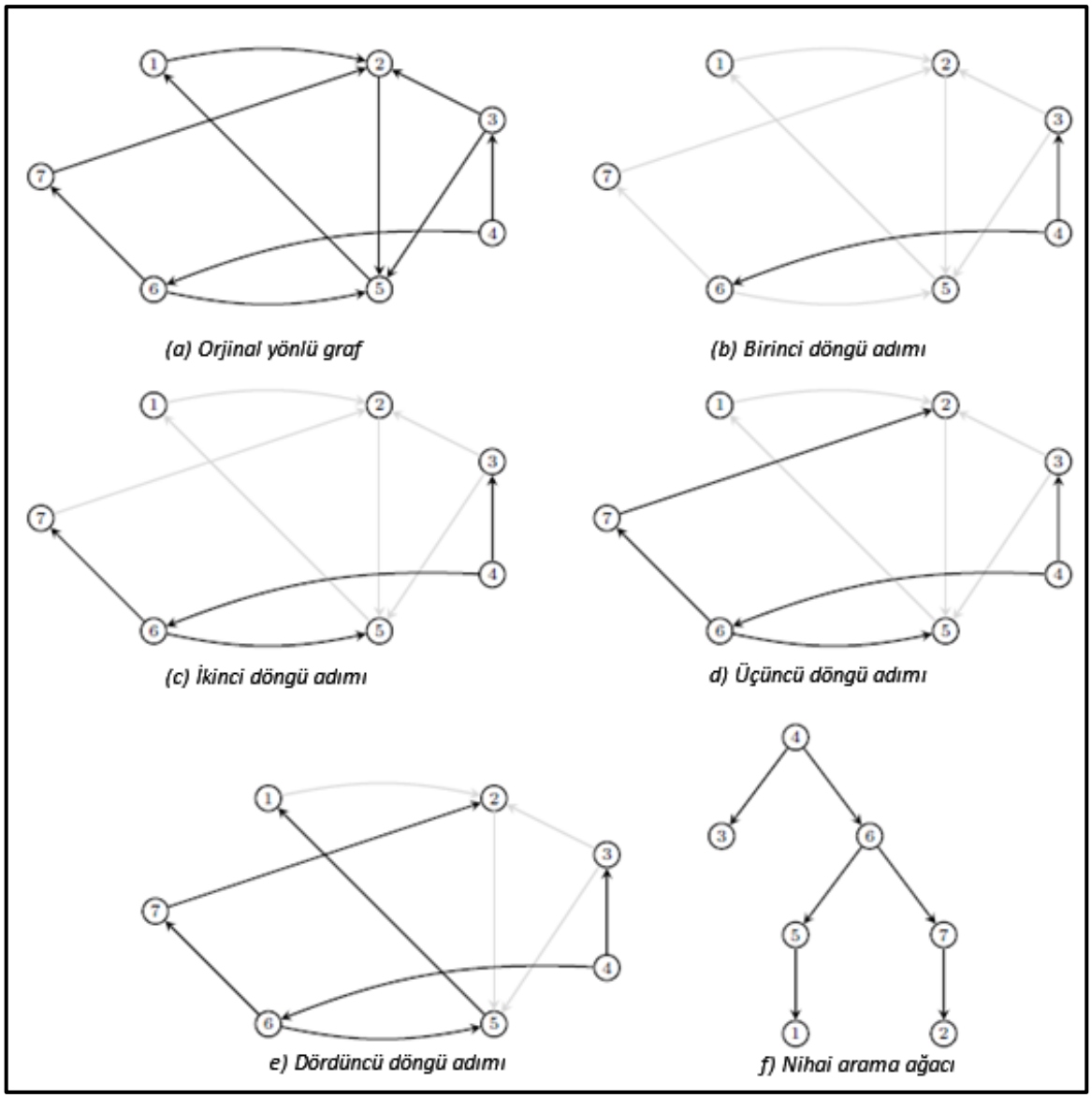
Genişlik öncelikli arama algoritmasının yönsüz ve yönlü graflardaki uygulama adımlar sırasıyla Şekil 6 ve Şekil 7'de gösterilmiştir.

Şekil 6: Yönsüz Graf için Derinlik Öncelikli Arama Algoritması Adımları



Kaynak: Joyner ve diğerleri, 2013: 117.

Şekil 7: Yönlü Graf için Derinlik Öncelikli Arama Algoritması Adımları



Kaynak: Joyner ve diğerleri, 2013: 118.

1.3. Çift Yönlü Arama

Çift yönlü arama, yönlü bir grafta bir kaynak düğümden bir hedef düğüme en kısa yolu bulan bir arama algoritmasıdır. Biri kaynak düğümden ileriye doğru, diğeri hedef düğümden geriye doğru olmak üzere eş zamanlı başlatılan iki aramanın bir orta noktada kesişmesini sağlayarak hızlı bir arama gerçekleştirmek üzere tasarlanmıştır. Klasik en kısa

yol algoritmaları çoğu durumda çift yönlü olarak çalıştırıldığında daha hızlı sonuç üretebilmektedirler (Pohl, 1969: 8-14).

1.4. En Kısa Yol Problem Türleri

1.4.1. Tek-Çiftli En Kısa Yol Problemi

Bir kaynak (başlangıç noktası)ve hedef (varış noktası) düğüm çifti arasındaki en kısa yolu bulma problemidir.

G = (V, E) olarak tanımlanan bir ağırlıklı grafta P(S, T), S düğümünden T düğümüne ulaşan bir yol üzerindeki düğümler kümesi olsun. P(S, T) yolu üzerindeki ara düğümler N_i (i = 1, ..,n) ile gösterilsin. L(N_i, N_{i+1}) = C(E_i) iki düğüm arasındaki maliyet (bağlantı ağırlığı) olarak tanımlanırsa L(S, T) = $\sum_{i=1}^{n-1} C(E_i)$, P(S, T) yolunun toplam maliyeti olur. Böylece problem toplam maliyeti en küçük yapacak U_i düğümlerinin bulunması olarak tanımlanır.

Çoğu EKY probleminde maliyet olarak iki düğüm arasındaki mesafe ele alınır. Bununla birlikte maliyet, iki düğüm arasındaki süre, parasal maliyet, sarf edilen efor gibi farklı kavramlar üzerinden de tanımlanabilir.

1.4.2. Tek-Kaynaklı En Kısa Yol Problemi

Bir kaynak düğümden diğer tüm düğümlere en kısa yolları bulma problemidir. Yol maliyetleri tek-çiftli problemde olduğu gibi tanımlanır. Çoğu EKY algoritması, graftaki tek kaynaklı en kısa yolları hesaplama prensibi ile çalışmaktadır.

1.4.3. Tüm-İkililer En Kısa Yol Problemi

Tüm kaynak-hedef düğüm ikilileri için en kısa yolu bulma problemidir. Bu problemde de toplam maliyetler tek-çiftli problemdeki gibi tanımlanır. Bu problem için geliştirilen algoritmalar genellikle büyük graflarda tek-çiftli problemin çözümünün hızlandırılması için önişlem adımı olarak kullanılırlar.

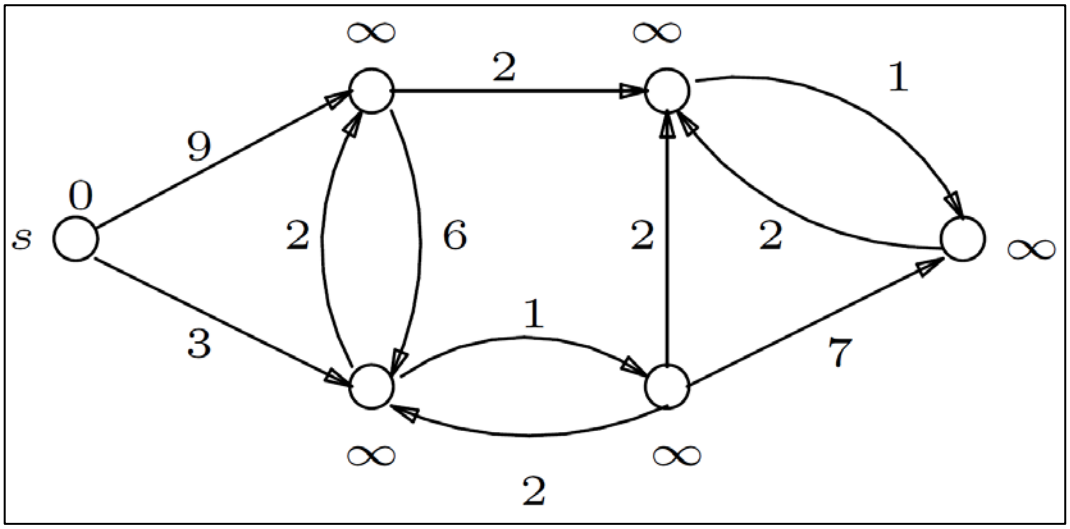
1.5. En Kısa Yol Algoritmaları

1.5.1. Dijkstra Algoritması

Dijkstra algoritması bir kaynak düğümden diğer tüm düğümlere en kısa yolu ve bu yolun toplam maliyetini hesaplar. Negatif ağırlık, paralel kenar ve döngü içeren ağlarda kullanılamaz. D. W. Dijkstra tarafından geliştirilmiş olan (Dijkstra, 1959: 269-271) algoritma, genişlik öncelikli arama algoritmasının genelleştirilmiş halidir (Joyner ve diğerleri, 2013: 124) ve etiketleme ve etiket düzeltme adımlarından oluşmaktadır.

Burada başlangıçta tüm düğümlerin toplam maliyetlerini gösteren etiketler ∞ (sonsuz) olarak atanır. Seçilen başlangıç noktası (kaynak) düğümü ise 0 olarak etiketlenir (Şekil 8). Ardından başlangıç düğümüne komşu düğümlerin etiketleri kenar ağırlıkları ile değiştirilerek işlem devam eder. Kaynak düğüm etiketine kenar ağırlığı eklenerek komşu düğüm için maliyet etiketi elde edilir. Eğer yeni maliyet değeri önceki maliyet değerinden küçükse etiket değiştirme işlemi gerçekleştirilir (Harju, 2011: 14). Algoritma adımları yönlü bir örnek graf için Şekil 9'da gösterilmiştir.

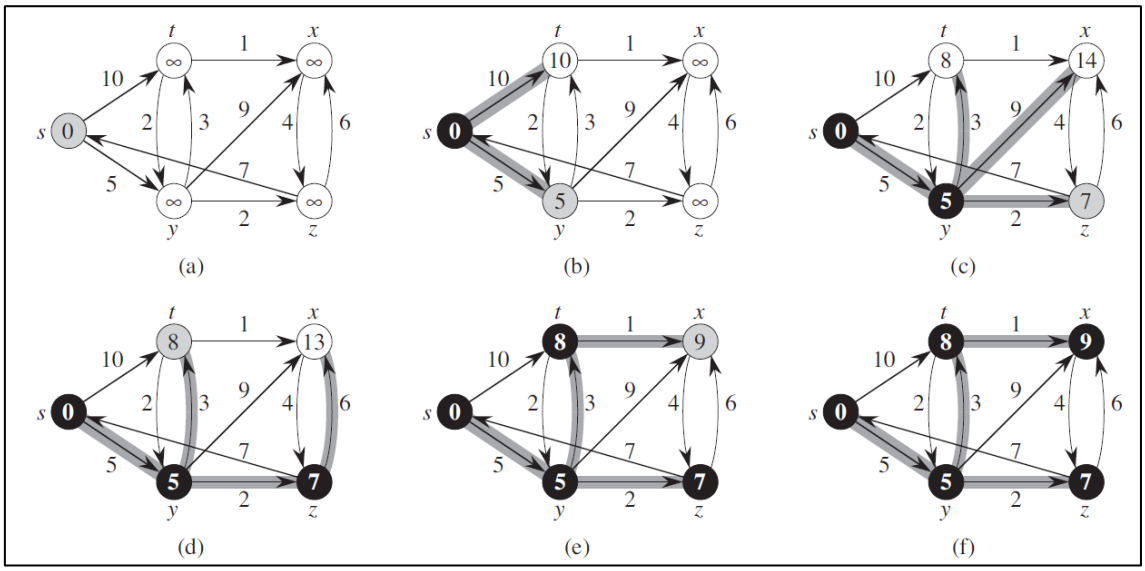
Şekil 8: Dijkstra Algoritması Örneği



Kaynak: Bagn-Jensen ve Gutin, 2007: 55.

Dijkstra algoritması, zaman içinde farklı araştırmacılar tarafından ele alınmış ve daha hızlı sonuçlar üretebilen gelişmiş sürümleri üretilmiştir. Aslında çoğu en kısa yol algoritması Dijkstra algoritmasının temel yaklaşımı olan genişlik öncelikli arama ile etiketleme üzerine kuruludur. Bunun yanında etiketleme temeline dayanan ancak derinlik öncelikli arama kullanan en kısa yol algoritmaları da geliştirilmiştir (Bkz. A* Algoritması).

Şekil 9: Dijkstra Algoritması Adımları



Kaynak: Cormen ve diğerleri, 2009: 659.

Hiyerarşik (ağaç) modellerde kullanılan tekniklerin gelişmesine bağlı olarak yeni yığın güncelleme teknikleri Dijkstra algoritmasına uyarlanmıştır. Bu şekilde algoritmanın temel yaklaşımını değiştirmeden, özellikle en kısa yol düğüm listesinin güncellenme yönteminin hızlandırılmasına dayalı iyileştirmeler elde edilmiştir. Bu kapsamdaki ilk iyileştirme çalışması, tüm kenar ağırlıkları tamsayı olan graflar için yapılmış ve algoritmanın çalışma zamanının azaltılması sağlanmıştır (Boas ve diğerleri s. 99-127). Takip eden yıllarda ise algoritmanın geçici maliyet yığınının güncellenmesinde farklı yığın tekniklerinin algoritmaya uyarlanması ile algoritmanın çalışma zamanının daha da azaltılabileceği gösterilmiştir (Ahuja ve diğerleri, 1990: 222, Thorup, 2000: 86-109).

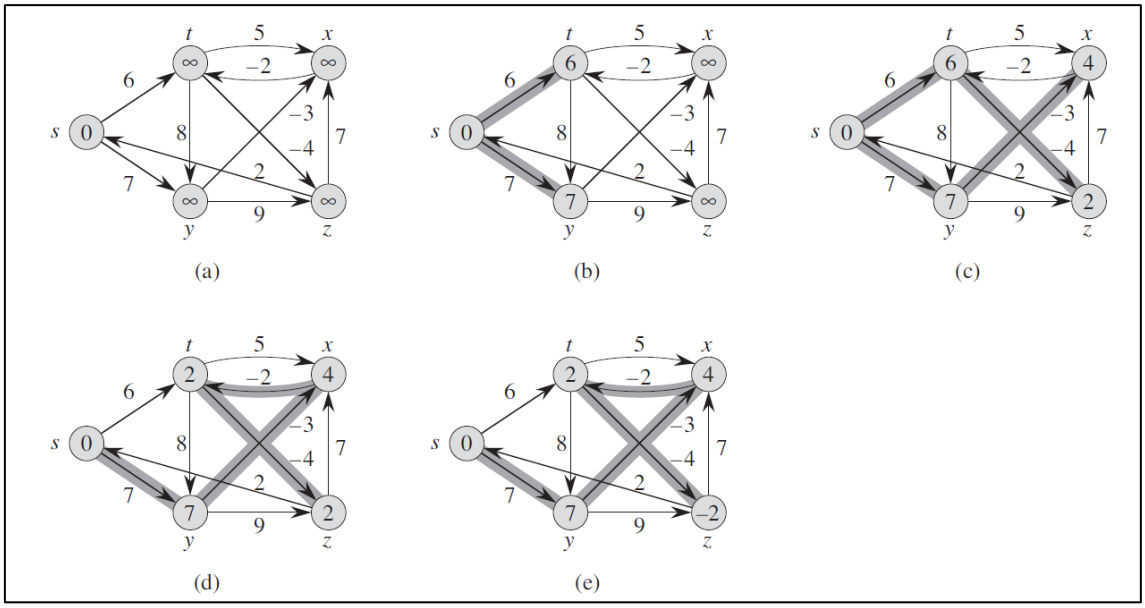
Chen ve diğerleri (2007), Dijkstra algoritmasını farklı yığın modelleri ile test etmiş ve bu modelleri Amerika'nın 11 farklı bölgesindeki ulaşım ağları üzerinde uygulayarak modeller arasında ayrıntılı bir kıyaslama çalışması yapmışlardır.

Shu-Xi, (2012: 1186-1190), Dijkstra algoritmasının yönlü graflarda sonsuz döngüye girmesini eleştirerek algoritmanın yönlü ve yönsüz graflarda sorunsuz çalışabilmesi için bir geliştirme yaklaşımı önermiştir.

1.5.2. Bellman-Ford Algoritması

Bellman-Ford algoritması negatif ağırlıklı ağlarda da çalışabilmektedir. Algoritma akışı Dijkstra algoritması ile benzerdir (Bkz. Şekil 10).

Şekil 10: Bellman-Ford Algoritması Adımları



Kaynak: Cormen ve diğerleri, 2009: 652.

Negatif ağırlıkların olduğu bir grafta, bir düğümden kendisine ulaşan bir yolun toplam ağırlığının negatif olması muhtemeldir. Bu durum, negatif döngü olarak adlandırılmaktadır. Negatif döngü içeren bir graf üzerinde, bir düğüm çifti arasındaki yolun toplam maliyeti, yolun negatif döngü üzerinden her geçişinde azalacağı için böyle

graflarda en kısa yolun bulunması olanaksızdır. Bellman-Ford algoritması, grafta negatif döngü olup olmadığını tespit edebilmektedir.

Yen (1970: 526-530), Bellman-Ford algoritmasının etiketleme tekniği, algoritmanın her adımda iki düğümü etiketlemesini sağlayacak şekilde geliştirilerek çalışma süresi azaltılmıştır. Grafın, topoloji sırasına göre iki alt grafa ayrılmasına dayanan bu yöntem, Bannister ve Epstein (2011: 41-47) tarafından geliştirilerek yöntemin performansı daha da artırılmıştır.

1.5.3. Floyd–Warshall Algoritması

Floyd-Warshall algoritması, bir graftaki tüm düğümler arasındaki en kısa yolu hesaplar. Bellman-Ford algoritmasında olduğu gibi negatif değerli ağırlıklarla da çalışabilir (Bang-Jensen ve Gutin, 2007: 58-59).

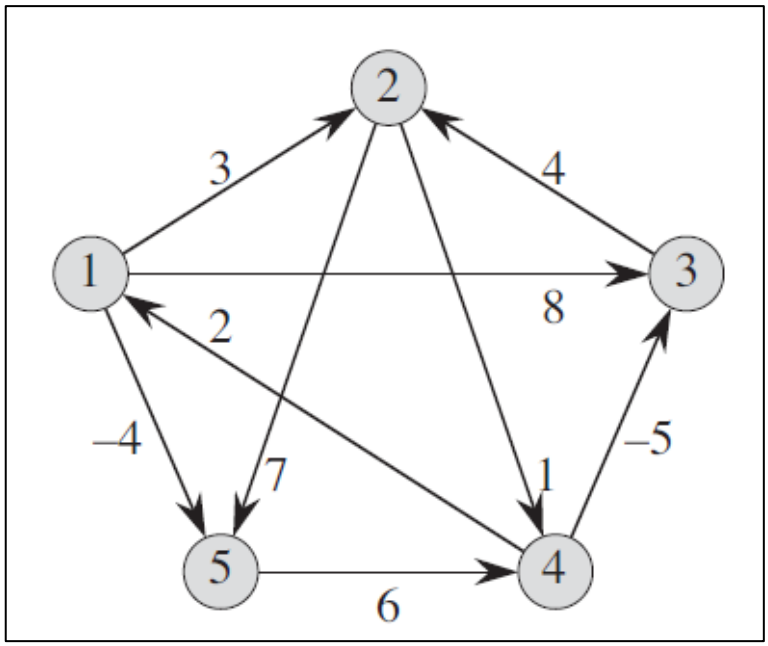
Bir graf üzerindeki tüm düğümler arası en kısa yolları hesaplamak için tek-kaynaklı en kısa yol algoritmalarını her düğüm için çalıştırmak gerekir. Floyd-Warshall algoritması ise iki düğüm arasındaki en kısa yolu bulmak yerine ara düğümlere odaklanarak, iki düğüm arasındaki en kısa yolun hangi ara düğümden geçtiğini hesaplar. Böylece iki düğüm arasındaki en kısa yolu döngüsel bir şekilde bulur. Döngüsel hesaplama için iki temel matris kullanılır. Bu matrisler maliyet (D) ve yol (π) matrisleridir (Sen, 2013: 90-91) (Şekil 12).

Maliyet matrisi başlangıç ağırlık matrislerinden (W) aşağıdaki gibi hesaplanır. Grafın ağırlık matrisi her düğümden diğerlerine olan bağlantı ağırlıklarından oluşur. Komşu olmayan düğümler için ağırlık değeri ∞ (sonsuz) kabul edilir (Cormen ve diğerleri, 2013: 695).

$$D_{ij}^{k} = \begin{cases} W_{ij}, k = 0 \\ Min(D_{ij}^{k-1}, D_{ik}^{k-1} + D_{kj}^{k-1}) \end{cases} \quad (1.1)$$

n: Düğüm sayısı; k: 1..n-1; i, j: 1.. n

Şekil 11: Floyd-Warshall Algoritması İçin Örnek Graf



Kaynak: Cormenve diğerleri, 2013: 691.

Şekil 11'deki Örnek graf için ağırlık matrisi (W)

$$\begin{matrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{matrix}$$

Yol matrisi ise aşağıdaki şekilde hesaplanır.

$$\pi_{i,j}^{0} = \begin{cases} Boş, i = j \; veya \; W_{ij} = \infty \\ i, i \neq j \; ve \; W_{ij} < \infty \end{cases} \tag{1.2}$$

$$\pi_{i,j}^{k} = \begin{cases} \pi_{i,j}^{k-1}, D_{ij}^{k-1} \leq D_{ik}^{k-1} + D_{kj}^{k-1} \\ \pi_{k,j}^{k-1}, D_{ij}^{k-1} > D_{ik}^{k-1} + D_{kj}^{k-1} \end{cases} \tag{1.3}$$

Kaynak: Cormen ve diğerleri, 2013: 697.

Şekil 12: Floyd-Warshall Algoritması Örneği için Maliyet ve Yol Matrisleri

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Kaynak: Cormen ve diğerleri 2013: 696.

Floyd-Warshall algoritması ile iki düğüm arasındaki en kısa yolu ve bu yolun maliyetini hesaplamak için aşağıdaki adımlar izlenir.

i düğümünden j düğümüne en kısa yolu hesaplamak için; yol matrisinin i. satırından i değeri bulunur. i değerinin bulunduğu sütun numarası(k) için k=j ise işlem sonlandırılır. Maliyet fonksiyonunun D_{ij} elemanı bu yolun maliyetini verir.

i değerinin bulunduğu sütün numarası (k) için k≠j ise i düğümünden j düğümüne en kısa yol k düğümünden geçer. D_{ij} elemanı toplam maliyete eklenerek birinci adıma geri dönülür. i düğümü yeni başlangıç düğümü(k) olarak atanarak işlemlere devam edilir.

1.5.4. Johnson Algoritması

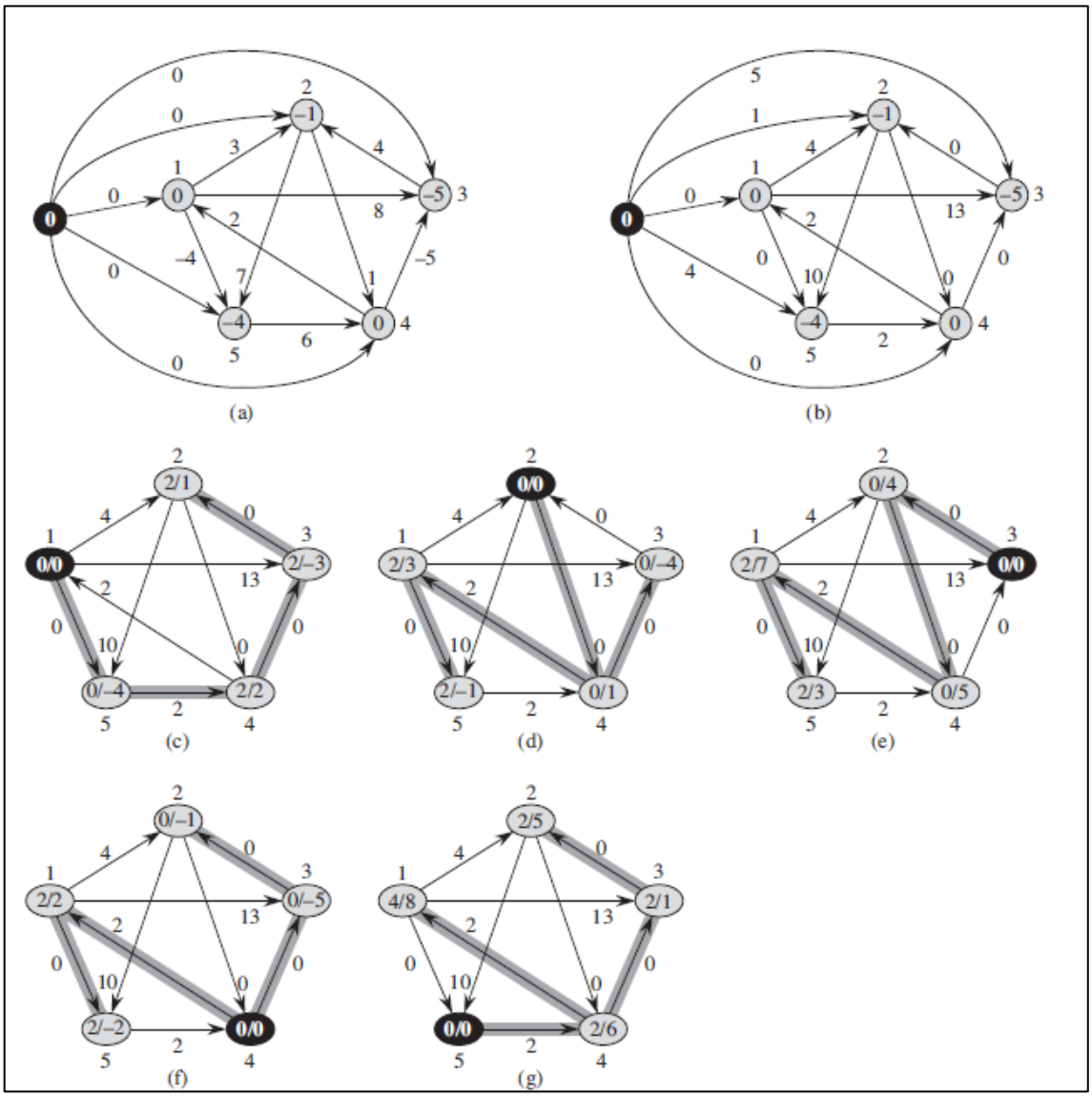
Johnson algoritması geniş graflarda Floyd-Warshall algoritmasına göre daha hızlı sonuç üretir. Algoritma grafa yeni bir düğüm ekleyerek bu düğümden tüm graf düğümlerine yeni kenarlar oluşturur. Kenar bağlantılarının yönü, yeni düğümden grafın diğer düğümlerine doğrudur ve tüm ağırlıklar başlangıçta sıfırdır (Şekil 13-a, Joyner ve diğerleri, 2013: 135).

Yeni düğüm ve bağlantılarla genişletilen graf üzerinde eklenen düğümden diğer tüm düğümlere en kısa yollar Bellman-Ford algoritması ile hesaplanır (Şekil 13 (a)). Johnson, bu yeni grafın negatif döngü içermesi durumunun ancak ve ancak orijinal grafın tüm düğüm çiftlerinden en az birinin negatif döngü içermesi durumda mümkün olacağını göstermiştir. Böylece orijinal grafın tüm düğüm çiftleri için negatif döngü içerip içermediğini Bellman-Ford algoritmasını bir kez çalıştırarak tespit edebilmek mümkün olmuştur. Normal koşullarda bu işlem için Bellman-Ford algoritmasının her düğüm çifti için uygulanması gerekmektedir (Cormen ve diğerleri, 2009: 700).

Algoritmanın ikinci aşamasında grafın tüm kenar ağırlıkları $\hat{w}_{ij} = w_{ij} + d(i) - d(j)$ formülü ile yeniden hesaplanır (Bkz. Şekil 13-b). Burada w_{ij} i düğümünden j düğümüne kenar ağırlığı; d(i) ise bir önceki adımda hesaplanan yeni düğümden i düğümüne en kısa yol maliyetidir. Böylece graf hiçbiri negatif olmayan yeni ağırlıklarla yeniden ağırlıklandırılmış olmaktadır. Johnson bu yeni ağılıklar ile hesaplanacak en kısa yolların orijinal ağırlıklarla hesaplanan yollarla aynı olacağını göstermiştir. Böylece algoritmanın geri kalan adımları için Dijkstra algoritmasını uygulamak mümkün olmaktadır (Joyner ve diğerleri, 2013: 135).

Algoritmanın son aşamasında tüm düğüm çiftleri için Dijkstra algoritması ile en kısa yollar hesaplanır (Şekil 13 c, d, e, f, g). Ancak en kısa yol maliyetleri bir önceki adımda uygulanan dönüşüm tersine çevrilerek ($w_{ij} = \hat{w}_{ij} - d(i) + d(j)$) hesaplanır (Joyner ve diğerleri, 2013: 136).

Şekil 13: Johnson Algoritması Adımları



Kaynak: Cormen ve diğerleri, 2013: 703.

1.5.5. A* Algoritması

A* algoritması, Dijsktra algoritmasının gelişmiş bir türevi olarak 1968 yılında Peter Hart ve diğerleri (1968) tarafından geliştirilmiş sezgisel bir arama algoritmasıdır. Graf tarama ve yol bulma problemlerinde yaygın olarak kullanılmakta ve Dijkstra algoritmasına göre daha iyi bir performans sağlamaktadır.

Algoritma, arama yapılacak ara düğümlerin önceliklerini belirlemek için sezgisel bir f(x) fonksiyonu kullanır. f(x) fonksiyonu iki alt fonksiyonun toplamı olarak hesaplanır. Alt fonksiyonlardan biri başlangıç düğümünden x ara düğümüne olan toplam mesafeyi temsil eden g(x) fonksiyonudur. Diğer fonksiyon(h(x)) ise ara düğümden hedef düğüme mümkün olan en kısa mesafe kestirimini hesaplar. Örneğin rotalama problemlerinde h(x), iki düğüm arasındaki doğrusal (kuş uçuşu) yol olarak alınabilir. Böylece arama işlemi f(x) fonksiyon değeri en küçük olan düğümden devam eder.

A* algoritması Dijkstra algoritmasından farklı olarak derinlik öncelikli bir algoritmadır. Etiketleme, başlangıç düğümüne komşu olan tüm düğümler yerine, her defasında sezgisel fonksiyona göre seçilen bir sonraki düğüme doğru ilerleyerek yapılır. Böylece kaynak düğüme ilk ulaşılan yolun en kısa yola yakın bir toplam maliyetinin olması hedeflenir. Bu işlem, algoritmanın daha az işlemle en kısa yolu bulmasını sağlayarak diğer yöntemlere göre daha yüksek bir performans sergilemesini sağlar.

Rios ve Chaimowicz (2009), A* algoritmasının paralelleştirilmiş çift yönlü uygulamasını geliştirerek algoritmanın çalışma süresini anlamlı düzeyde azaltmışlardır.

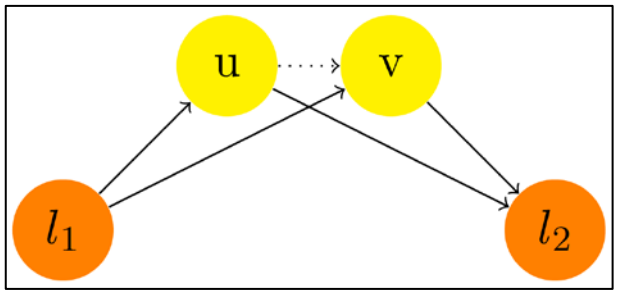
Nosrati ve diğerleri (2012) star (*) algoritmaları olarak bilinen A* algoritmasının türevlerini ele alarak bunların algoritma akışlarını göstermişlerdir.

1.5.6. ALT Algoritması

A* algoritması, referans noktaları ve üçgen eşitsizliği temeline dayanan bu yöntem, Goldberg ve Harrelson (2004) tarafından geliştirilmiştir. Yöntem ağ üzerinde önceden belirlenen referans noktaları üzerinden, A* algoritmasındaki sezgisel fonksiyonun hesaplanmasını esas alır. Çalışmada referans noktalarının en basit haliyle rastlantısal olarak

seçilebileceği belirtilmiş ve buna ek olarak farklı referans noktası seçim önerileri geliştirilmiştir. Algoritma, tüm düğümlerin referans noktalarının tümüne önceden hesaplanmış uzaklıkları kullanılarak çalıştırılır. Üçgen eşitsizliğinden yararlanan algoritma, iki düğüm arasındaki mesafenin düğümlerin referans noktalarına ayrı ayrı mesafelerinin farkından büyük olacağı gerçeğinden yararlanır. Bu gerçekten hareketle düğümler arası mesafe için bir alt sınır belirlenmiş olur. Algoritma etiketlemeye alt sınırı en küçük olan düğümden başlayarak en kısa yola hızlı bir şekilde ulaşmayı hedefler (Goldberg ve Werneck, 2005). Şekil 14, ALT algoritmasındaki referans noktalarını göstermektedir.

Şekil 14: ALT Algoritması Referans Noktaları



Kaynak: Fuhs 2010: 2,

Zhao ve diğerleri (2008) zamana bağımlı ağlarda ALT algoritmasını ön işlemlerle hızlandırmışlardır.

Nannicini ve diğerleri (2010) ise algoritmayı zamana bağımlı ağlara çift yönlü olarak uyarlamıştır.

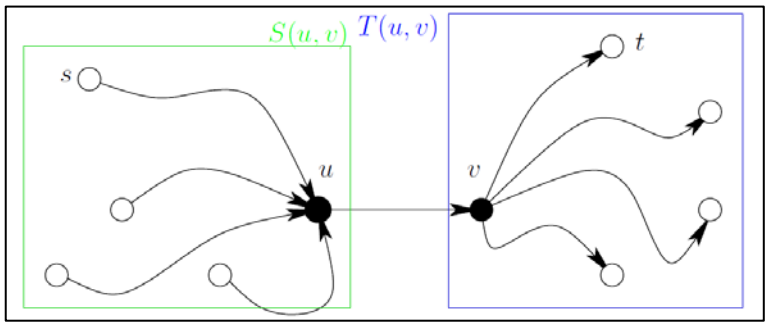
1.5.7. Geometrik Kalıplar Algoritması

Geometrik Kalıplar Algoritması (GKA), Almanya ulusal demiryolu ağı için zaman parametreli olarak Schulz ve diğerleri (2000) tarafından geliştirilmiştir. Wagner ve diğerleri (2003) ise algoritmayı tüm ağırlıklı ağlar için genelleştirmişlerdir.

GKA, grafın her kenarı için, en kısa yol çözümünün o kenardan başladığı düğümleri içeren bir geometrik şekil (Şekil 15) oluşturarak önceden hesaplanmış bu veri yapısı üzerinden Dijkstra algoritması ile çözüm üretir. Böylece algoritmanın çözüme

ulaşamayacak düğümlere sapması engellenerek çözümün hesaplanma süresi iyileştirilmiş olur.

Şekil 15: GKA Örnek Kalıpları



Kaynak: Wanger ve diğerleri 2003: 2.

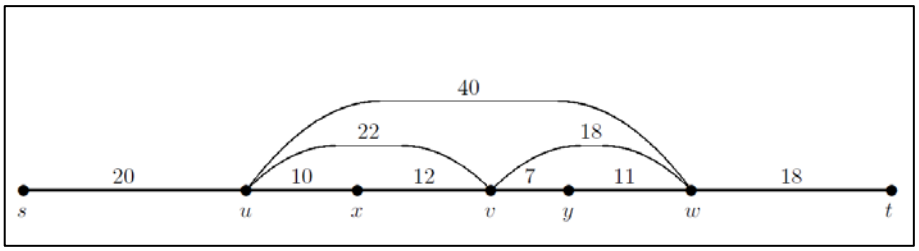
Algoritma daha sonra gerçek zamanlı trafik durumuna göre dinamik olarak çalışacak şekilde düzenlenmiş ve ön işlem adımının hızlandırılması sağlanmıştır (Wagner ve diğerleri, 2005).

1.5.8. Reach Algoritması

Algoritma Gutman (2004) tarafından geliştirilmiştir. Yöntem grafın her düğümü için önceden hesaplanan reach ölçüsüne göre çalışır. Grafın bir düğümü (v) için reach ölçüsü şöyle hesaplanır: Herhangi iki düğüm arasında v düğümünden geçen tüm yollar için başlangıç düğümünden v düğümüne ve v düğümünden varış düğümüne toplam maliyeti küçük olan değerlerin en büyüğü v düğümünün reach ölçüsü olarak belirlenir. Şekil 16'de v düğümünün u-başlangıç ve u-varış düğümleri için reach ölçüsü 18'dir.

Derinlik öncelikli arama temelinde çalışan algoritma her adımda, toplam maliyeti ilgili düğümün reach ölçüsünden küçük olan veya hedef düğüme öklit uzaklığı reach ölçüsünden küçük olan düğüm ile devam ederek en kısa yola ulaşmaya çalışır. Reach algoritması A* algoritması ile birleştirilerek en kısa yol çözümüne daha hızlı ulaşabilmektedir (Gutman, 2004).

Şekil 16: Reach Ölçüsü



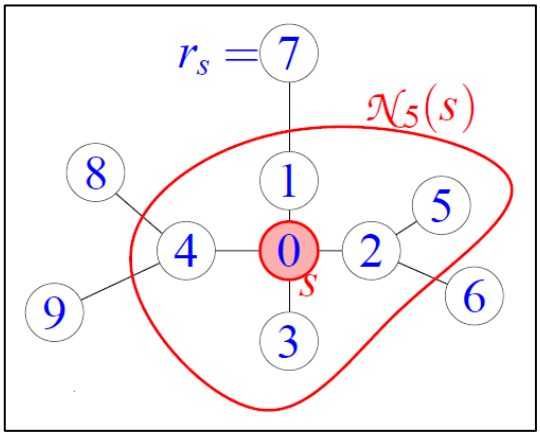
Kaynak: Goldberg ve diğerleri, 2007a: 7.

1.5.9. Anayol Hiyerarşileri Algoritması

Anayol Hiyerarşileri Algoritması (AHA), Sanders ve Schultes (2005) tarafından geliştirilmiştir. Algoritma, uygun olarak tanımlanmış yerel arama ve anayol ağı kavramları temel alınarak işletilir.

Yerel arama işlemi, problemin başlangıç düğümü s'ye komşu olan belli H sayıdaki düğümler ($N_H(s)$) arasında gerçekleştirilir (Şekil 17) (Sanders ve Schultes, 2005: 569).

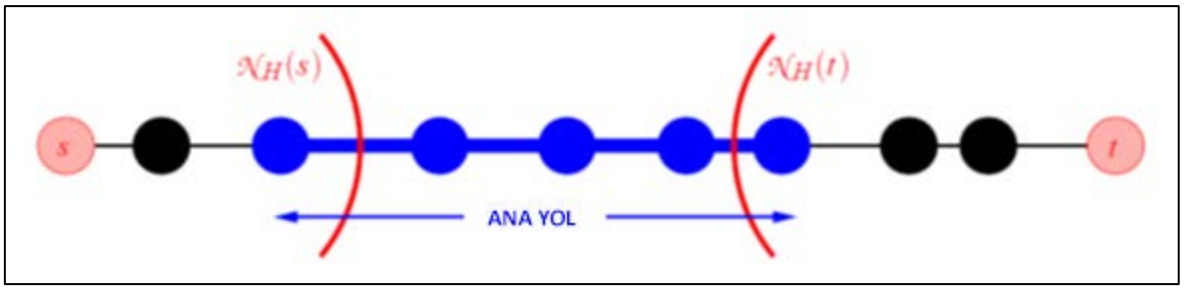
Şekil 17: Anayol Hiyerarşileri: H = 5 için Örnek Gösterim



Kaynak: Schultes, 2005: 12

Ana yol ağı ise bu başlangıç düğümüne H komşulukta olmayan düğümlerden oluşur (Şekil 18) (Sanders ve Schultes, 2005: 569).

Şekil 18: Anayol Hiyerarşileri: Anayol Ağı



Kaynak: Schultes 2005: 13.

Buraya kadar bütün düğümler en az bir başka düğümün H komşuluğu içinde yer alacağından, anayol ağı tüm düğümleri içerecektir. Bu noktadan sonra ağı daraltmak için her yerel arama ağına ait ara düğümler (Şekil 18'deki siyah düğümler gibi) ağdan çıkarılarak ikinci seviye anayol ağı oluşturulur. Ağda birbirine H sayısından fazla komşulukta düğüm kalmayıncaya kadar bu işlem devam ettirilerek ana yol ağının seviyeleri oluşturulur. Böylece hiyerarşik yapıda anayol ağları elde edilir. En kısa yol sorgulamaları en alt seviyedeki ana yol ağından bir üst seviyeye doğru ilerletilerek iki düğüm arasındaki en kısa yol çözümüne ulaşılır (Sanders ve Schultes, 2005: 572-573).

Schultes ve Sanders 2007 yılında algoritmanın dinamik olarak değişen ağlarda da efektif bir şekilde çalışmasını sağlayacak geliştirmeler ile önişlem süresini 2 dakikadan daha kısa süreye düşürmüşlerdir. Yeni algoritmanın sorgu süresinin ise 2 ile 40 milisaniye arasında olduğu görülmüştür.

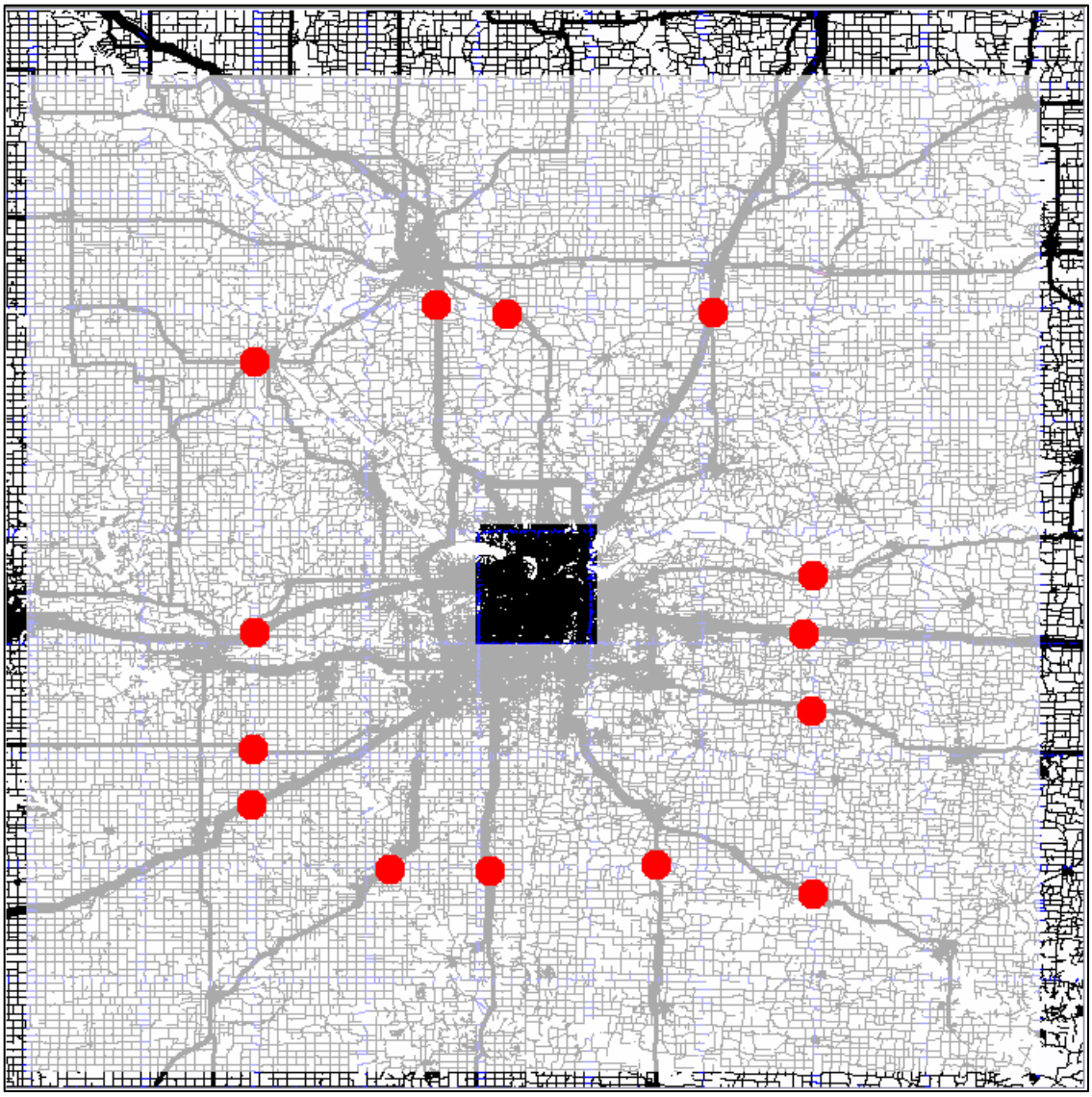
1.5.10. Geçiş Düğümleri Algoritması

AHA temeline dayanan Geçiş Düğümleri Algoritması (GDA) da graf üzerinde önceden hesaplama ile elde edilmiş veriler ile arama algoritmasının hızlandırılması esas alınmaktadır.

GDA'da graf, birbirlerine belli bir uzak olmayan öklid mesafesinde (D) olan düğümlerden oluşan alt ağlara bölünür. Bir alt ağın her düğümünden diğer tüm düğümlere en kısa yollar hesaplanarak, en kısa yollardan en az biri üzerinde bulunan düğümler

belirlenir. Bu düğümlerden D mesafesi içinde kalan son düğümler geçiş düğümü olarak belirlenir (Şekil 19).

Şekil 19: GDA Geçiş Düğümleri



Kaynak: Bast ve diğerleri, 2006: 2.

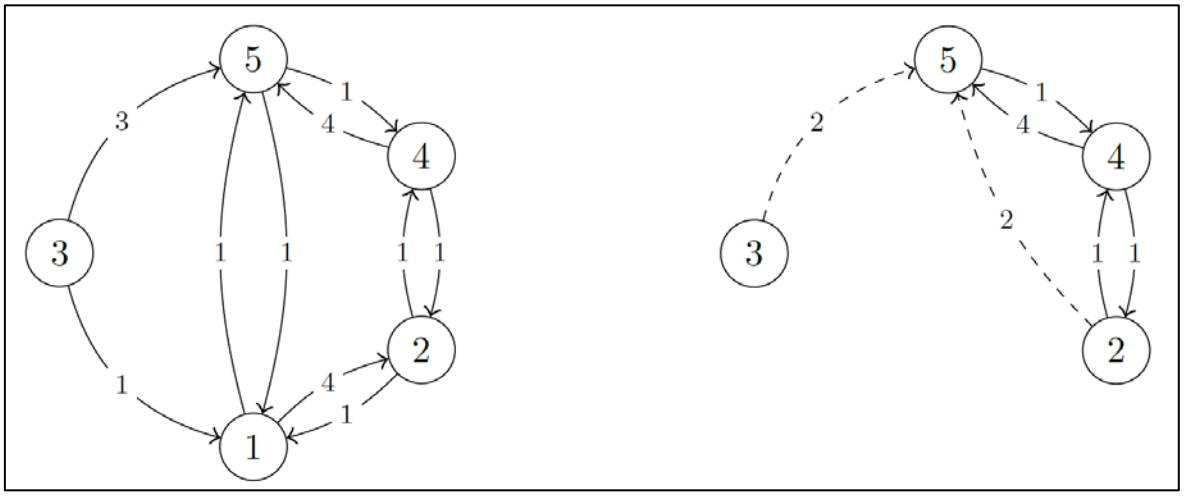
Geçiş düğümleri hesaplandıktan sonra tüm düğümlerden geçiş düğümlerine, geçiş düğümleri arası ve tüm geçiş düğümlerinden tüm düğümlere en kısa yol mesafeleri hesaplanarak bir tabloya kaydedilir. Arama işlemi bu tablo üzerinden gerçekleştirilir.

1.5.11. Daralma Hiyerarşileri Algoritması

Geisberger (2008) tarafından geliştirilen Daralma Hiyerarşileri Algoritması (DHA), grafın hiyerarşik olarak daraltılması esasına göre çalışır ve AHA'nın iyileştirilmiş bir türevidir.

Algoritmada grafın tüm düğümleri artan bir önem sırasına göre numaralandırılır. Düğümler bu sıraya göre grafın tüm en kısa yollarını etkilemeyecek şekilde graftan çıkarılarak bir hiyerarşi oluşturulur. Çıkarılan düğüm bir en kısa yol üzerinde ise kopan en kısa yol çözümleri için grafa yeni "kısayol-shortcut" bağlantıları eklenir (Şekil 20-21). Böylece graf üzerindeki en kısa yollar bu daraltma işleminden etkilenmezler (Geisberger ve diğerleri, 2008: 320).

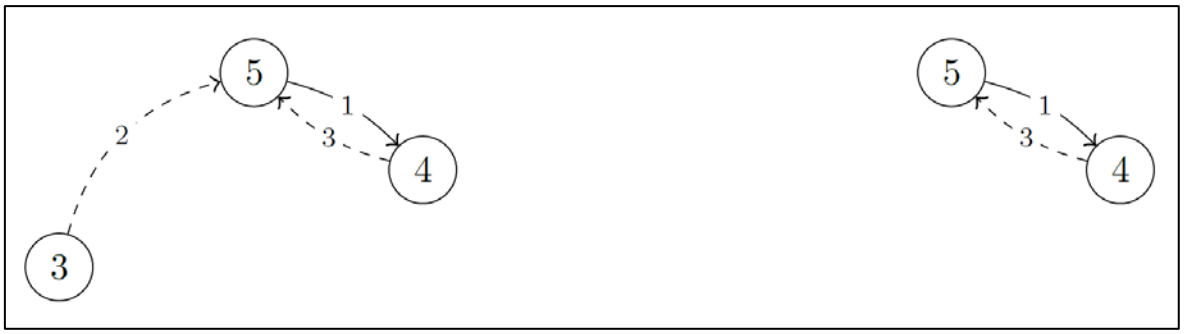
Şekil 20: DHA 1. Seviye Daraltma İşlemi



Kaynak: Columbus, 2012: 14.

Bir düğümün önem numarasının hesaplanması için faklı yaklaşımlar geliştirilmiş olmakla birlikte en basit anlamda bu değer, kenar farkı olarak adlandırılan yönteme göre düğüm çıkarıldığında oluşan kısayol sayısından düğümün bağlantı sayısı çıkarılarak hesaplanır (Geisberger, 2008: 10).

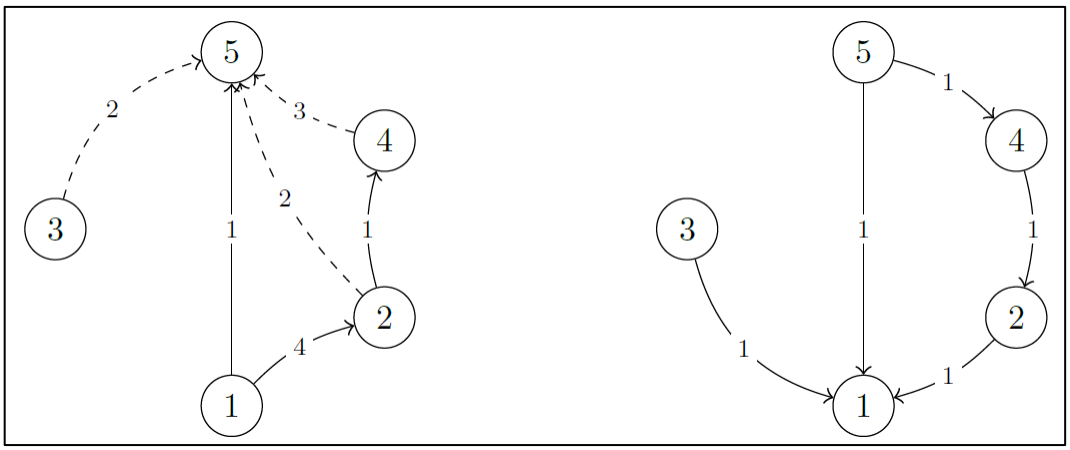
Şekil 21: DHA 2. ve 3. Seviye Daraltma İşlemi



Kaynak: Columbus, 2012: 15.

En kısa yol çözümü ise yukarıda anlatılan yönteme göre daraltılmış ağ üzerinden önceden hesaplanarak oluşturulan aşağı ve yukarı arama grafları ile Dijkstra algoritmasının iki yönlü bir uyarlaması ile hesaplanmaktadır. Aşağı arama grafı, kendinden sonraki düğüm önem seviyesi daha küçük olan; yukarı arama grafı ise kendinden sonraki düğüm önem seviyesi daha büyük olan düğümler arası bağlantıları içeren iki graftır (Şekil 22). İki yönlü Dijkstra algoritmasının ileri yönlü araması yukarı arama grafı üzerinde; geri yönlü araması ise aşağı arama grafı üzerinde koşturulur. Bir düğüm çifti arasında en kısa yol, bu iki aramanın önem sırası maksimum olan ortak bir düğümde kesişmesi ile belirlenir.

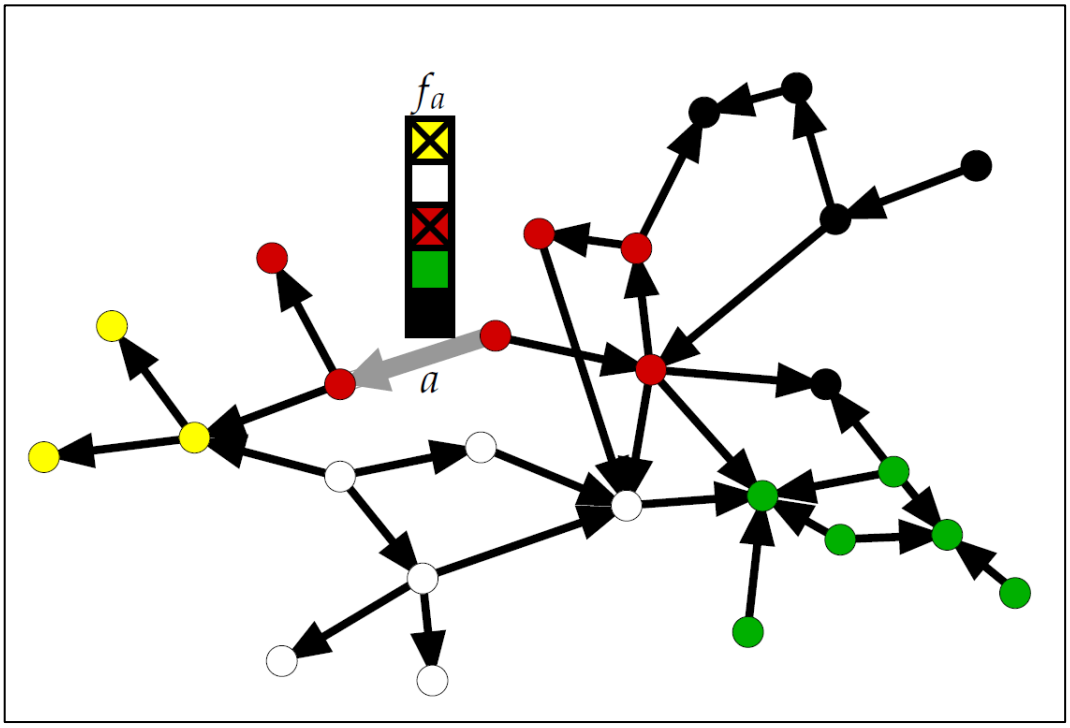
Şekil 22: DHA – Yukarı ve Aşağı Arama Grafları



Kaynak: Columbus, 2012: 20.

1.5.12. Kenar Bayrakları Algoritması

Şekil 23: KBA Kenar Bayrakları



Kaynak: Köhler ve diğerleri, 2006: 6.

Kenar Bayrakları Algoritması (KBA), önceden hesaplanmış kenar bayraklarına dayanan bir algoritmadır. Alt graflara bölünmüş grafın her alt grafındaki bağlantılar diğer alt graflara en kısa yol üzerinde bulunup bulunmadıklarına göre işaretlenir (Şekil 23). Böylece arama algoritması varış düğümünün bulunduğu alt grafa en kısa yol üzerinde bulunan bağlantılar üzerinden kolayca çalıştırılabilir.

Köhler ve diğerleri (2006), KBA'yı hızlandırmak için farklı önişlem adımı içeren birkaç iyileştirme önermişlerdir. Önerilen önişlem adımlarının klasik yaklaşıma göre daha az zaman aldığı ve sorgu süresini de önemli ölçüde azalttığı kaydedilmiştir.

1.5.13. Karma En Kısa Yol Algoritma Çalışmaları ve Yöntemlerin Karşılaştırılması

Cherkassky ve diğerleri (1996) Dijkstra ve Bellman-Ford algoritmalarını farklı öncelik kuyruğu yöntemleriyle uygulamış ve yöntemlerin performanslarını karşılaştırmıştır.

Schultes (2008) AHA'yı ALT ile geliştirerek karma algoritmayı statik ve dinamik ağlara uyarlamıştır. Ayrıca algoritmanın çok düğümden çok düğüme (many-to-many) çalışacak bir türevini geliştirmiş ve GDA'nın bazı özel durumlarında bu algoritmanın kullanılabileceğini söylemiştir.

Goldberg ve diğerleri (2007a) A* ve Reach algoritmalarını kullanarak en kısa yol sorgu süresini iyileştiren bir karma yöntem geliştirmişlerdir (REAL). Algoritma Kuzey Amerika yol ağındaki rastlantısal düğüm çiftleri ile test edilmiş ve ortalama sorgu süresinin 4 milisaniyeden az olduğu kaydedilmiştir.

Goldberg ve diğerleri (2007b) daha önce geliştirdikleri karma algoritma REAL'de ALT algoritmasını kullanarak yalnızca yüksek reach ölçüsüne sahip düğümler için referans noktaları belirlemek suretiyle iyileştirme yapmışlardır. Yaklaşık 20 milyon düğüm içeren ağlar üzerinde test ettikleri algoritma, bir-iki saatlik ön işlem adımından sonra rastgele seçilen düğüm çiftleri arasındaki en kısa yolları ortalama 1 milisaniye gibi çok kısa bir sürede hesaplayabilmektedir.

Arz ve diğerleri (2013) DHA'yı GDA ile birleştirerek hem önişlem hızı DHA'ya göre daha yüksek olan hem de sorgulama süresi diğer yöntemlere göre iyileştirilmiş bir yaklaşım geliştirmişlerdir. Yöntem ayrıca önişlem verisinin saklanması için daha az alana ihtiyaç duymaktadır.

Bauer ve Delling (2009) çoğu en kısa yol algoritmalarının çift yönlü arama tekniği ile çalışmasını eleştirerek geriye doğru aramanın mümkün olamayacağı zamana bağımlı ağlarda kullanılabilecek bir karma algoritma geliştirmişlerdir. Algoritma AHA kısayol kavramı ve KBA yöntemlerine dayanmaktır (SHARC). Çalışmada geliştirilen yöntem Avrupa ve Amerika yol ağları üzerinden test edilerek diğer yöntemlerle performans

karşılaştırmaları yapılmıştır. Ayrıca zamana bağımlı problemlerde ağın daraltılmasından kaynaklanan hatalı sonuçlar açısından yöntemler karşılaştırıldığında yeni yaklaşımın daha az hata oranı ile diğer yöntemlere yakın bir performans sergilediği görülmüştür.

Bauer ve diğerleri (2010) son on yılda geliştirilen hızlandırma tekniklerinin önişlem adımlarını ele alarak yöntemlerin karmaşıklık seviyelerini incelemiş ve tüm yöntemlerin NP-Hard sınıfında olduğunu göstermiştir.

Kalpana ve Thambidurai (2010) ALT ve GKA yöntemlerini karşılaştırarak bu algoritmaları hızlandırma teknikleri ile birleştirerek performanslarını artırmaya çalışmışlardır. Çalışmada iki algoritmanın birleştirilmiş uygulamasının daha iyi performans sergilediği görülmüştür.

Sanders ve Schultes (2007) GDA'nın genel bir çerçevesini çizerek algoritmayı AHA ile karma bir yaklaşımla hızlandırmışlardır. Ayrıca paralel programlama teknikleri ile algoritmanın önişlem hızlarının artırılabileceği gösterilmiştir.

Wu ve diğerleri (2012) son yıllarda geliştirilen en kısa yol algoritmalarını mekânsal-uyum ve düğüm-önemi temelli olmak üzere ikiye ayırarak farklı senaryolar içeren ağ yapılarında hangi yöntemin tercih edilmesi gerektiği konusunda öneriler getirmişlerdir.

Bast ve diğerleri (2014) son yıllarda geliştirilen en kısa yol algoritmaları ve hızlandırma tekniklerini ele alarak kapsamlı bir çalışma yapmışlardır. Neredeyse tüm türev yöntemlerin incelendiği çalışma literatürdeki en kapsamlı inceleme ve karşılaştırma çalışmalarından biridir.

1.6. K En Kısa Yol Algoritmaları

Klasik en kısa yol algoritmaları, iki düğüm arasındaki en kısa (en az maliyetli) yolu bulmak üzere tasarlanmıştır. Ancak bazı durumlarda en kısa tek bir yol hesaplaması yerine en kısa ilk k yolun ya da k'yıncı en kısa yolun hesaplanması da önemli olabilmektedir. K en kısa yol (KEKY) algoritmaları EKY algoritmalarının genelleştirilmiş halidir. Diğer bir deyişle EKY algoritmaları KEKY algoritmalarının özel (k=1) bir şeklidir.

KEKY algoritmaları döngülü ve döngüsüz olacak şekilde iki ana temel üzerinde geliştirilmiştir. Döngüsüz en kısa yol problemi, bir düğüm üzerinden bir defa geçme kısıtıyla oluşturulmuş özel bir problem türüdür.

KEKY algoritmaları üzerine 1957 yılından beri birçok çalışma yapılmıştır. Yen (1971) tarafından yayınlanan çalışma, literatürde döngüsüz KEKY algoritması üzerine yapılmış en temel çalışma olarak kabul edilir. Algoritma, iki temel liste üzerinden işletilir. Birinci liste en kısa yolları tutarken ikinci liste, algoritmanın bir sonraki en kısa yolu hesaplamak için kullanacağı aday kısa yol çözümleri barındırır (Yen, 1971: 713-714).

Chen ve Chou (1999), Floyd Algoritmasının genelleştirilmiş halini, K en kısa yol problemine uyarlayarak yeni algoritmayı Yen (1971) algoritması ile karşılaştırmışlardır. Çalışmada yeni algoritmanın daha fazla bellek alanına gereksinim duymasına karşın Yen algoritmasına göre daha yüksek performans sağladığı gözlenmiştir.

Jianfu Li (2012), Yen (1971) algoritmasını A* algoritması ile birleştirerek karma Yen* algoritmasını geliştirmiş ve algoritmanın performansını artırmıştır.

Döngülere izin verilen durumlarda ise Eppstein (1991: 4) tarafından geliştirilen algoritma en kısa yol üzerinde olamayan bağlantıların ikili yığınlara (binaryheaps) işlenmesi ve böylece sorgulama süresinin iyileştirilmesine dayanmaktadır.

Aljazzar ve Leue (2011) k en kısa yol için A* algoritmasının probleme uyarlanması ile sezgisel bir yöntem (K*) geliştirmişlerdir. Algoritma Newyork City ve Doğu Amerika yol ağları üzerinde test edilerek Eppstein'nin (1997) k en kısa yol algoritması ile karşılaştırılmış ve daha yüksek performans gösterdiği kaydedilmiştir.

1.7. Toplu Ulaşım Ağlarında En Kısa Yol Problemi

G = (V, E) şeklinde gösterilen bir toplu ulaşım ağında (TUA) V, ağda bulunan durakların kümesi olarak tanımlanır. Bazı tanımlamalarda önemli noktalar (POI-Point-of-Interest) da V kümesine dâhil edilebilmektedir. EKY probleminin temel çözüm yaklaşımları paralel bağlantı içermeyen ağlar için geliştirildiklerinden, bu algoritmalar TUA'na uyarlanırken oluşturulacak grafların bu varsayıma uygun olarak tasarlanmaları

gerekmektedir (Meng ve diğerleri, 1999: 3). Çünkü TUA'ında iki durak arasında birden fazla hatla seyahat etmek mümkün olduğundan ilkel haliyle graf paralel bağlantılar içerecektir. Ayrıca TUA'ında seyahatler zamana bağlı olarak planlandığı için problem geleneksel EKY probleminden faklı olarak bir zaman parametresi içermektedir.

TUA'ında hatlar arası yürüme bağlantıları da önemli bir yere sahiptir. Özellikle farklı ulaşım seçenekleri (metro, otobüs vb.) arasındaki aktarımlarda duraklar farklı konumlarda olacağından duraklar arası yürüme bağlantıları problemin en kısa yol çözümünü etkileyecektir.

Yürüme bağlantılarının yanında TUA'da bir yolculuk çözümünün aktarım sayısı da çözümün kalitesini önemli derecede etkileyen bir faktördür. Bu sebeple çoğu durumda en düşük maliyetli çözümün aynı zamanda en az aktarım gerektiren çözüm olması gerekecek ve böylece problem çok amaçlı hale gelecektir.

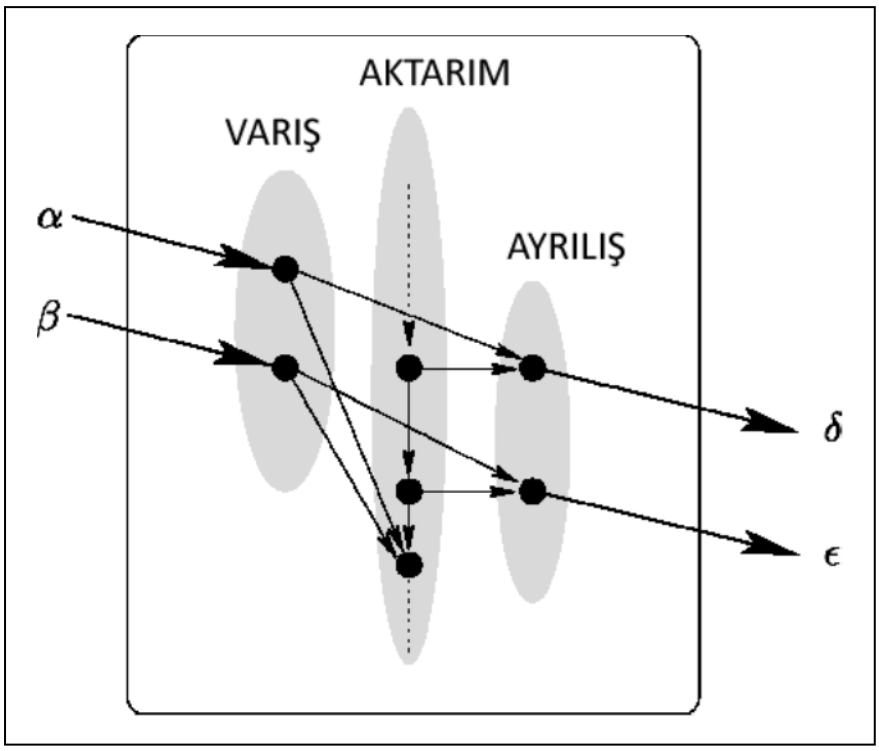
Aktarım sayısı ve toplam yürüme mesafeleriyle birlikte çözüm kalitesini etkileyen diğer bir önemli faktör de yolculuğun bir sonraki adımındaki hat için durakta bekleme süreleridir. Örneğin aynı yolculuk maliyetine sahip iki çözümden daha az bekleme süresi içeren çözümün kalitesinin çoğu durumda daha yüksek olacağı açıktır.

Zaman parametresi ve paralel bağlantılar göz önüne alınarak literatürde TUA için iki farklı yaklaşım geliştirilmiş ve ağ için oluşturulacak graf bu yöntemlere göre yeniden tasarlanmıştır.

1.7.1. Zaman Genişletme Yaklaşımı

Zaman genişletme yaklaşımında (ZGY) düğüm kavramı, bir durakta belli bir zamandaki (kalkış veya varış) olayına karşılık gelir. Böylece TUA'ndaki tüm hatlar için hattın bir duraktan kalkış ve durağa varış olayı ağda bir düğüm olarak gösterilir (Müller-Hanneman ve diğerleri, 2006: 3).

Şekil 25: ZGY Örnek Gösterimi



Kaynak: Pyrga ve diğerleri, 2008: 10.

Aktarım durağı olarak belirlenen bir durakta, bir önceki hat varış düğümü ile bir sonraki hat kalkış düğümü arasında oluşturulacak bağlantılarda ağırlık genellikle zamanlara göre oluşturulur. Bu tür bağlantılar oluşturulurken hareket düğümünün zaman parametresinin varış düğümü zaman parametresinden büyük olması gerekir (Delling, 2009: 18).

Şekil 25 de çok küçük bir kısmı gösterilen ZGY'ında düğüm sayısının artışına bağlı olarak ağ önemli ölçüde büyümektedir.

1.7.2. Zamana Bağımlı Yaklaşım

Bu yaklaşımda graf geleneksel yaklaşımda olduğu gibi duraklar üzerinden tanımlanır. Ancak paralel bağlantıları engellemek amacıyla bir duraktan geçen her hat için ayrı bir düğüm tanımlanır. Zaman bağımlı yaklaşımda (ZBY) bağlantı ağırlıkları ise

algoritmanın çalışması sırasında çözüme giren düğüm dikkate alınarak dinamik olarak belirlenir (Müller-Hanneman ve diğerleri, 2006: 3).

ZBY'ın çoğu çalışmada ZGY'na göre daha yüksek performans sergilediği görülmüştür (Müller-Hanneman ve diğerleri, 2006: 3; Pyrga ve diğerleri, 2008: 38; Delling, 2009: 18). Ayrıca arama algoritmasını hızlandıracak önişlem adımlarının oluşturulduğu birçok hızlandırma tekniğinde ZBY'ın uygulanma kolaylığına dikkat çekilmiştir (Delling, 2009: 63-64).

1.7.3. Toplu Ulaşım Ağlarında Yolculuk Planlama Problemi Optimizasyon Kriterleri

TUA'larında EKY problemi çözülürken amaç fonksiyonu genellikle hedef durağa varış zamanına oluşturulur. Yani optimizasyon probleminin amacı başlangıç durağından varış durağına en erken ulaşan çözümü bulmaktır. Bu basitçe bir maliyet minimizasyonu problemidir.

Varış zamanı kriteri, çoğu araştırmacı tarafından geliştirilen çözüm yöntemlerinde ve akademik çalışmalarda kullanılan en temel optimizasyon kriteri olmakla beraber 1.7'de bahsedilen hususlardan dolayı problem, birçok çalışmada çok kriterli olarak ele alınmıştır.

Toplu ulaşım ağlarında yolculuk planlama probleminin ele alındığı çalışmalarda en sık kullanılan diğer optimizasyon kriterleri aktarım sayısı, ücret ve yolculuğun güvenilirliği (trafik vb. etkilere göre hesaplanan çözümün uygulanabilirlik olasılığı) kriterleridir.

Ayrıca birçok araştırmacı, problemin bazı durumlarda belli bir başlangıç zamanına göre değil; bir zaman aralığına (bir gün içindeki tüm seyahat seçenekleri gibi) göre çözülmesi gerektiğini belirtmişlerdir. Özellikle şehirlerarası seyahatlerde yolcular, yolculuklarını belli bir hareket saatine göre değil; belli bir hareket gününe göre planlamaktadırlar. Bu tür seyahat sorgulamaları literatürde Profil ya da Aralık Sorgulama (Profile – Range Query) olarak adlandırılmaktadır.

1.7.4. Toplu Ulaşım Ağları En Kısa Yol Çalışmaları

EKY problemi, literatürde yaklaşık 50 yıl gibi kısa bir geçmişe sahip olmasına rağmen, probleme özellikle Dijkstra algoritmasını temel alan birçok çözüm yaklaşımı geliştirilmiştir. Bilgisayar teknolojisinin ve buna bağlı olarak bilgisayarların hesaplama kapasite ve performanslarının da zaman içindeki hızlı gelişimi ile EKY problemi çözüm yöntemleri, çok büyük graflar için bile optimum çözümleri mikro saniyeler ile ölçülebilen sürelerde hesaplayabilmektedir (Bast ve diğerleri, 2014: 17).

Başlangıçta çoğu araştırmacı ve akademisyen, TUA'nda da EKY problemini çözerken, klasik yöntemleri kullanmışlardır. Bu anlamda ilk çalışmalar Dijkstra algoritmasının zaman parametreli olarak çözüm üretmesini hedefleyen sürümlerinden oluşmaktadır. Kısaca Tarifeli (scheduled) Ağ olarak adlandırılabilecek bir TUA'nın zaman genişletmeli veya zaman bağımlı olarak ele alınması durumunda, ağın katlanarak büyümesi sebebiyle klasik yaklaşımların optimum çözümleri yeterince etkin bir performansta üretemediği görülmüştür. Bu sebeple son birkaç yılda araştırmacılar, problemi graf yapısından kurtarıp daha basit veri yapıları ile ele almıştır (Delling ve diğerleri, 2013) (Dibbelt ve diğerleri, 2013). Böylece yeni algoritmaların çözüm hesaplama performanslarının artırılması hedeflenmiştir. Bu yeni bakış açısı problemin trafik, hava koşulları gibi olasılıklı (probabilistic) parametrelere bağlı değişim gösteren faktörlerden etkilenmesi sebebiyle önişlem adımı içeren yöntemlerin gerçek uygulamalarda kullanılabilir çözümler üretemeyeceği görüşüne dayanır. Bu çerçevede geliştirilen yöntemler, ağı basit veri yapılarıyla ele alan ve dinamik programlama temelinde dayanan yöntemlerdir.

Aşağıda TUA'da yolculuk planlama problemini ele alan temel çalışmalar kronolojik sırada verilmiştir.

Meng ve diğerleri (1999), Singapur TUA üzerinde uyguladıkları yöntemle problemi çok modlu (metro, otobüs v.b.) ve çok kriterli (en az seyahat mesafesi, en düşük ücret, en erken varış zamanı gibi) ele alarak bir çözüm yaklaşımı geliştirmiştir.

Schulz ve diğerleri (2000), çalışmalarında, optimizasyon kriteri olarak sadece seyahat süresini ele almış ve Dijkstra algoritmasını farklı hızlandırma teknikleriyle birleştirerek

ortalama sorgu süresinin azaltılmasını amaçlamışlardır. Çalışmada varış zamanının önceden bilinememesine bağlı olarak zaman parametreli problemde çift yönlü arama yaklaşımının kullanılamayacağı ancak varış zamanı için belirlenebilecek bir üst sınır değeri ile çift yönlü arama yapmanın mümkün olacağı belirtilmiştir. Graf yapısının ZGY'na göre oluşturulduğu çalışmada grafın daraltılması ve buna bağlı olarak sorgu süresinin azaltılması için çeşitli yaklaşımlar önerilmiştir.

Brodal ve Jacob (2003) yaptıkları çalışmada, TUA'yı başlangıç durağından itibaren her aktarımda erişilebilen duraklara göre katmanlara ayırmışlardır. Daha sonra bu katmanları aktarım yapılabilen duraklar arasında bağlantılar oluşturarak birbirine bağlamış ve her bir katman üzerinde Dijkstra algoritmasını kullanarak en erken varış zamanı kriterine göre yolculuk çözümleri üretmişlerdir. Literatürde Katmanlı Dijkstra (Layered Dijkstra - LD) olarak adlandırılan algoritma, aktarım sayısı ve varış zamanı kriterlerine göre optimum çözümler üreterek problemi çok kriterli olarak ele alan etkin bir yöntemdir.

Brandes ve diğerleri (2004), Almanya Ulusal Demiryolu Ağı ve Tren seferleri için en az yolculuk süresini hedef alan bir yolculuk planlama yaklaşımı geliştirmişlerdir. Çalışmada bir hareket durağından varış durağına en kısa sürede gidebilmek için varış durağına ulaşabilen ve en erken kalkan hattın seçilmesi önerilmiştir.

Wu ve Hartley (2004), KSP algoritmaları ile Nottingam City TUA'sında k adet en kısa yolu hesaplamak için KSP probleminin klasik yaklaşımı olan yol çıkarma işlemini önermişlerdir. Bu yaklaşıma göre bir durak çifti arasındaki en kısa yol hesaplandıktan sonra bulunan yol graftan çıkarılarak tekrar arama yapılır. Böylece ikinci en kısa yol hesaplanmış olur. Ancak çözümün birden fazla çözüm adımından oluştuğu (aktarımlı) durumlarda, en kısa yolun tüm adımlarını graftan çıkarmak, bu adımlardan birinin sonraki kısa yol çözümlerinin bir parçası olma ihtimali göz önüne alındığında uygun bir yaklaşım olmayabilir.

Barrett ve diğerleri (2006), TUA'nın bağlantılarını düğümler arası mümkün olan ulaşım seçeneklerine göre etiketleyerek bir çözüm yaklaşımı geliştirmişler ve bu etiketleme işlemi için farklı yöntemler önermişlerdir. Daha sonra önişlem adımında etiketlere göre uyarladıkları Dijkstra algoritmasının hızlandırma teknikleri ile geliştirilmiş versiyonlarıyla

en kısa yol çözümlerini aramışlar ve kullandıkları yöntemlerin sonuçlarını karşılaştırmışlardır.

Bielli ve diğerleri (2006), TUA'nı nesne yönelimli (object-oriented) olarak modelleyerek hiyerarşik yaklaşımla ağ üzerindeki kaynak-hedef düğüm çiftleri için en kısa yolları hesaplamışlardır. Kullanıcının belirleyeceği aktarım sayısına göre çalışan bir uygulama ile yöntemi farklı boyuttaki ağlar üzerinde test ederek sonuçları karşılaştırmışlardır. Ayrıca yöntemin gerçek zamanlı trafik durumuna uyarlanabileceğini belirtmişlerdir.

Müller-Hannemann ve diğerleri (2006), TUA'nı ZGY ve ZBY'a göre modelleyerek tek kriterli ve çok kriterli problemler için en kısa yol çözümleri hesaplamışlardır. ZGY ile oluşturdukları yöntemin problemi en az aktarım kriterine göre de çözebileceğini belirmişlerdir. Yöntemlerin testinde ZBY yönteminin açık bir şekilde daha yüksek performansla çözüm ürettiği görülmektedir. Ayrıca elde ettikleri test sonuçlarına göre çok kriterli problemin sorgu süresinin 10 kat daha yüksek olduğu görülmüştür.

Sanders ve Schultes (2006), son yıllarda geliştirilen hızlandırma teknikleri ile sorgu süresi milyon kat azalan Dijkstra algoritmasının TUA'na uyarlanması üzerinde yaptıkları çalışmada yöntemlerin özellikle ağırlıkların dinamik olarak değiştiği ağlarda nasıl kullanılacağı üzerinde durmuşlardır.

Wagner ve Willhalm (2007), son birkaç yılda önişlemlerle hızlandırılan Dijkstra algoritmasını TUA'na uyarlamış ve hızlandırma algoritmalarının birleşimlerinden oluşan yeni yaklaşımların sonuçlarını test etmişlerdir. Birçok hızlandırma tekniği amaç yönelimli (goal-directed) ve çift yönlü arama teknikleri ile birleştirilerek, sorgu işlemlerinin dikkate değer bir şekilde hızlandırılabileceğini söylemişlerdir. Ancak hızlandırma tekniği seçiminin yüksek oranda problemin (kabul edilebilir önişlem süresi, kullanılabilir bellek alanı v.b. gibi) kendi kısıtlarına bağlı olacağını belirtmişlerdir.

Batz ve diğerleri (2008), DHA'nı ZBY'a göre değiştirip algoritmayı bağlantıların zamana bağlı olarak ağırlıklandırıldığı ağlar için genelleştirmişlerdir. Yeni yaklaşımın çift yönlü arama yönteminin kullanılabileceği ilk zaman parametreli hiyerarşik hızlandırma tekniği olduğunu söylemişlerdir.

Delling (2008), AHA ve KBA algoritmalarının birleştirilmiş bir versiyonu SHARC algoritmasının zaman parametreli olarak ZBY'na göre uyarlamıştır. Delling bu yeni uyarlamanın kıtasal boyuttaki zaman parametreli TUA'nda mutlak en kısa yolu hesaplayabildiğini göstererek yöntemin bu tür problemler için ilk etkin yöntem olduğunu söylemiştir.

Koszelew (2008), TUA'yı en kısa seyahat süresi kriterine göre çözüm üreten iki yeni yaklaşım geliştirmiş ve bu çözüm yaklaşımlarını sorgu süresi ve çözüm kalitesi açılarından karşılaştırmıştır. Çalışmada standart genişlik öncelikli arama algoritması ile oluşturulmuş iki boyutlu aktarım matrisleri ve iki durak arasındaki en kısa mesafeyi gösteren uzaklık matrisi kullanılmıştır. Aktarım matrisleri bir önişlem adımı ile bir kez hesaplanmakta ve bir kaynak-hedef düğüm çifti arasında k adet aktarımlı bir çözüm olup olmadığını (0-1) göstermektedir. Birinci yöntem en kısa yolu aktarım matrisleri ve uzaklık matrisini kullanarak hesaplarken ikinci yöntemde uzaklık matrisi ile birlikte düğüm etiketleme tekniği kullanılmıştır. Test sonuçlarına göre birinci yöntemin genel olarak daha hızlı çalıştığı söylenmiştir.

Pyrga ve diğerleri (2008) ZGY ve ZBY ile modelledikleri TUA'nda iki yaklaşımın performanslarını karşılaştırmışlardır. Çalışma soncunda ZBY'ın açık bir şekilde daha yüksek performansla çalıştığı ancak karmaşık ağları modellemede ZGY'nın daha güçlü olduğu söylenmiştir. Ayrıca çalışmada Dijkstra algoritması çok etiketli (Multicriteria-Label) olarak uygulanmıştır. Böylece çözümler aktarım sayısı ve varış zamanına göre optimize edilerek Dijkstra algoritmasının çok kriterli bir versiyonu elde edilmiştir (Multicriteria Label Setting - MLS).

Booth ve diğerleri (2009), TUA'nın ilişkisel veri yapılarıyla gösterimi üzerine yaptıkları çalışmada çok modlu ve çok kriterli problemler için bu ilişkisel veri yapısına dayanan bir sorgulama dili önermişlerdir.

Delling (2009), daha önce önermiş olduğu zaman parametreli SHARC yöntemini çok kriterli probleme uyarlamıştır. Çalışmada ayrıca DHA ve ALT yöntemlerinin birlikte kullanıldığı CALT ve DHA ve KBA yöntemlerinin birlikte kullanıldığı CHASE yöntemleri

ile hem yerel arama hem de toplam çalışma performansları bakımından önceki yöntemlere göre daha güçlü karma yaklaşımlar önerilmiştir.

Delling ve Wagner (2009), son yıllarda geliştirilen hızlandırma tekniklerinin temel unsurlarını ele alarak bu yöntemlerin zaman parametreli problemlerde ürettikleri çözümlerin doğruluğunu garantilemek için geliştirilmeleri gerektiğini göstermişlerdir. Bu bağlamda etkin birer hızlandırma tekniği olarak ALT, SHARC ve DHA yöntemlerini ele almışlardır. Çalışmada ayrıca en hızlı çözümün her zaman en iyi çözüm olmayacağının altı çizilerek problemin çok kriterli olarak çözümlenmesi gerektiği vurgulanmıştır.

Martinek ve Zemlicka (2009), TUA'da en kısa yol probleminin hızlandırma tekniklerini Prag TUA için mobil cihazlar üzerinde uygulayarak sonuçları tartışmışlardır. Hızlandırma tekniklerinin yüksek bellek ihtiyacına bağlı olarak yöntemlerin mobil cihazlarda istenilen performansta çalışmadığı için yöntemlerin yeni geliştirmelere bellek gereksinimlerinin azaltılması gerektiği ifade edilmiştir.

Berger ve diğerleri (2010), zaman parametreli ağlarda çok kriterli en kısa yol problemi için tamamen dinamik iki yeni hızlandırma tekniği önermişlerdir. SUBITO adını verdikleri birinci teknikte ağ ağırlıkları tamamen çalışma zamanında atanabilmektedir. İkinci teknik ise KBA'nın yeni bir uyarlaması olan k-flags yöntemidir. Çalışmada önerilen yöntemlerin farklı algoritma teknikleri ile birleştirilmiş versiyonları test edilmiş ve sonuçlar tartışılmıştır.

Fan ve Mumford (2010), problemin zorluk derecesinin yüksekliğine işaret ederek çözüm için meta-sezgisel yöntemlerin uygun olacağını belirtmiş; tavlama benzetimi ve tepe tırmanma meta-sezgisel yöntemleriyle çözümler üretmişlerdir. Bu çalışmada da problemde yolculuk mesafesiyle birlikte aktarım sayısının da minimize edilmesi gerektiğinin altı çizilmiş ve test sonuçlarında tavlama benzetimi algoritmasının tepe tırmanma algoritmasına göre daha yüksek performansla çözüm ürettiği görülmüştür.

Jariyasunant ve diğerleri (2010), hatlar arası optimal aktarım noktalarını önceden hesapladıkları çözüm yöntemiyle, problemin k adet en kısa yol çözümlerini hesaplamış ve en kötü durumda sorgu süresinin 3sn'den az sürdüğü görülmüştür. Ayrıca optimal aktarım

noktalarının hesaplandığı önişlem adımının genellikle 1 dakikadan daha az zaman aldığı belirlenmiştir.

Kumari ve Geethanjali (2010), TUA üzerinde en kısa yol problemi çözüm yöntemleri üzerine yapılan çalışmaları inceleyerek bir literatür incelemesi yapmıştır. Standart en kısa yol ve k en kısa yol yöntemlerinin statik ve dinamik ağlardaki uygulamalarının incelendiği çalışmada rastlantısal kaynak-hedef düğümleri üzerinden elde edilen bulguların gerçek uygulamalarda çoğu kez elde edilemediği ifade edilmiştir.

Liu (2010), TUA'ında EKY bulma problemini, yolculukta kullanılacak ulaşım modlarının önceden belirlenmiş bir sıraya bağlı olarak veya uygun sıralamanın çözüm esnasında belirleneceği iki alt problem üzerinden ele alınmıştır. Çalışmada özellikle modlar arası aktarım noktalarının belirlenmesi problemine odaklanarak TUA'nda çok modlu problem için çözüm üreten iki farklı algoritma önerilmiştir. Önerilen algoritmalar Münih TUA üzerinde test edilerek sonuçları karşılaştırmıştır.

Merrifield (2010), A* algoritmasının farklı türevlerini Chicago TUA'ya uygulayarak yaptığı çalışmada algoritmaların farklı özellikteki kaynak-hedef düğüm çiftleri arasında gerçekleştirilen en kısa yol aramalarındaki sonuçlarını karşılaştırmıştır. Çalışmada algoritma performansları geleneksel Dijkstra algoritmasının performansına göre bağıl performanslar şeklinde gösterilmiş ve ağın 2 boyutlu ikili ağaç yapısı (2d-binary tree) ile alt ağlara bölünmesine dayalı olarak geliştirilen algoritma genel anlamda daha yüksek performans göstermiştir.

Antsfeld ve Walsh (2012a), problemi yürüme mesafeleri, aktarım süreleri, çok amaçlılık gibi açılardan ele alarak TUA'nda optimum seyahat çözümünü bulan bir algoritma geliştirmişlerdir. Buna ek olarak önişlem adımının süresini ve veri saklama alanı ihtiyacını azaltan birkaç hızlandırma tekniği önermişlerdir. Sidney TUA üzerinde uyguladıkları yöntemin önişlem ve sorgulama süresi açısından etkin bir yöntem olduğunu söylemişlerdir.

Antsfeld ve Walsh (2012b), daha önce geliştirdikleri algoritmayı mevcut hızlandırma teknikleriyle birleştirerek yeniden ele almış ve iyileştirmişlerdir. Yeni yaklaşımın Sidney ve New South Wales TUA üzerinde uygulanması ile elde edilen

sonuçlara göre yöntemin bellek ihtiyacının ve önişlem zamanının anlamlı bir şekilde azaldığı görülmüştür.

Li ve diğerleri (2012), literatür çalışmalarında çoğu yöntemi TUA üzerindeki anlık değişimleri hesaba katmadan çözüm üretmeleri açısından eleştirerek hatların dinamik olarak hesaplanan varış zamanı kestirimlerine göre düzenlenmiş bir yaklaşım önermişlerdir.

Zhang ve diğerleri (2012), ulaşım ağını özel (yürüme, bisiklet ve otomobil) ve genel (toplu ulaşım hatları) olarak iki alt ağa ayırarak problemi, bu alt ağların aktarım bağlantıları ile oluşturulmuş ağ üzerinden süper-ağ olarak adlandırdıkları teknik ile çözmüşlerdir. Tekniğin Eindhoven TUA üzerinde uygulama sonuçlarını tartışarak çift-yönlü arama ve sezgisel tekniklerin sorgu performansını önemli ölçüde artırdığını söylemişlerdir.

Delling ve diğerleri (2009) TUA verilerini tek boyutlu diziler şeklinde işleyerek yeni bir veri yapısı önermişlerdir. Bu yeni veri yapısı sayesinde optimizasyon işleminde Dijkstra gibi graf algoritmalarının performansını olumsuz etkileyen öncelik sırası kuyruk işlemlerine ihtiyaç duyulmadığından, sorgu sürelerinde önemli derecede iyileşme sağlanmıştır. Çalışmada ayrıca yeni veri yapısına uygun dinamik programlama temeline dayanan yeni bir algoritma (RAPTOR) önerilmiştir. Problemin varış zamanı ve aktarım sayısına göre en uygun çözümlerini hesaplayan algoritma, günümüzde bu alada geliştirilen en yüksek performanslı ve en çok kullanılan algoritmalar arasındadır.

Artigues ve diğerleri (2013), TUA en kısa yol çözümlerini, seyahat süresi ve aktarım sayısını minimize edecek şekilde deterministik olmayan ağ ağırlıkları üzerinden hesaplayabilen yaklaşım önermişlerdir. Topolojik etiketleme tekniğine dayanan bu yaklaşımla çift-yönlü ve sezgisel yöntemlerle birleştirilerek çözümler üretilmiş ve yöntemlerin performanslarını karşılaştırılmıştır.

Dibbelt ve diğerleri (2013), Delling ve diğerleri (2009) tarafından geliştirilen RAPTOR algoritmasına benzer bir şekilde, TUA zaman çizelgesi üzerinden çözüm üreten yeni yaklaşım geliştirmiştir. Bağlantı Tarama Algoritması (Connection Scan Algorithm - CSA) olarak adlandırdıkları algoritmada çözüm hesaplamaları, RAPTOR algoritmasından

farklı olarak duraklar üzerinden değil; bağlantılar üzerinden yapılmaktadır. Bağlantıları başlangıç durağından erişilebilirliğine göre etiketleyen algoritma, böylece varış zamanı kriterinin yanı sıra aktarım sayısını da optimize ederek problemi RAPTOR gibi çok kriterli olarak çözmektedir. Strasser ve Wagner (2014) algoritmayı ülkeler arası ölçekte ve profil sorgulamalarda hızlı sonuç üretecek şekilde geliştirerek yeni algoritmayı Hızlandırılmış Bağlantı Tarama Algoritması (Accelerated Connection Scan Algorithm - ACSA) olarak adlandırmışlardır.

Huguet ve diğerleri (2013), çok modlu ağlarda yolculuğun bir adımının bisiklet veya otomobil ile yapılması önerildiğinde, dönüş yolunun aracın park edildiği noktadan geçmesi gerekliliğinden dolayı TUA'nda yolculuk planlama problemini iki yönlü (gidiş-dönüş) olarak ele almışlardır. Çalışmada dönüş yolunu dikkate almadan yapılacak planlamanın gidiş yönü için optimum olmasına rağmen dönüş yolu için optimumdan uzak olabileceğine dikkat çekilmiştir. İki yönlü bir planlama için çift-yönlü arama tabanlı etkin bir algoritma önerilmiş ve yöntem Paris şehrinin de içinde yer aldığı Fransa'nın Ile-de-France bölgesi TUA üzerinde test edilmiştir.

Liebig ve diğerleri (2014) Open Trip Planner[4] alt yapısını kullanarak Irlanda'nın Dublin şehri TUA üzerinde anlık trafik verisini dikkate alarak yolculuk planlaması yapan bir yöntem önermişledir.

Bast ve Storandt (2014), durak çiftleri arasında zaman bağımsız olarak kullanılabilir tüm güzergâhları bir örüntü (Pattern) olarak ele almış, hat ve durak bazında bulanık (fuzzy) olarak hesapladıkları örüntü benzerliklerine göre grupladıkları güzergahlar üzerinden çözümler üretmişlerdir (Transfer Patterns - T. Patterns). Önişlem adımlı bu yaklaşım, önişlem zamanının yüksek olmasına karşın, çok kriterli optimizasyonda çok büyük boyuttaki TUA'da optimum çözümü literatürdeki en düşük sorgu süresinde hesaplayabilmektedir.

[4] http://www.opentripplanner.org/

İKİNCİ BÖLÜM

2. EN AZ AKTARIM ODAKLI ÇÖZÜM YAKLAŞIMI

Toplu taşıma ağlarında aktarım, iki nokta arasında seyahat ederken değiştirilen araç sayısı olarak tanımlanmaktadır. Aktarım, iki farklı ulaşım modu (metrodan otobüse gibi) arasında olabileceği gibi aynı moda ait iki vasıta arasında (A1 otobüs hattından A5 otobüs hattına gibi) da olabilir.

2.1. Aktarım Sayısının Çözümün Kalitesine Etkisi

Toplu taşıma ağları üzerinden planlanan bir yolculuğun kalitesi, bu yolculuğun gerektirdiği aktarım sayısı ile doğrudan ilişkilidir. Bu ilişkinin psikolojik ya da alışkanlıklara bağlı yönleri olduğu gibi bazı teknik yönleri de vardır. Psikolojik faktörlerin incelenmesi bu çalışmanın yapıldığı alandan çok daha farklı bilim dallarının konusu olduğu için çalışmada bu faktörlere yer verilmemiştir. Aktarım sayısı ve çözüm kalitesi ilişkisi teknik yönden ise dört alt başlık altında incelenebilir.

Aktarım sayısı – Maliyet ilişkisi: Bazı toplu taşıma işletmeleri belli hat ve seferler için indirimler uygulasalar da genel anlamda bir yolculuktaki her aktarım yeni bir araç kullanımı anlamına geleceği için yolculuk maliyetini artıracaktır.

Aktarım sayısı – Zamanlama ilişkisi: Toplu ulaşım ağlarında bir yolculuk planlanırken toplu taşıma vasıtasının belirli bir duraktan hareket saati tahmini olarak hesaplanır. Her aktarım yeni bir tahmini gerektireceğinden, bu durum yolculuğun planlanan zamanlama ile yapılamaması riskini artıracaktır.

Aktarım sayısının bekleme süresine etkisi: Bir yolculukta aktarım sayısı arttıkça, yolcuların bir sonraki hat için durakta bekleme sürelerinin de artış eğiliminde olacağı açıktır. Bu sebeple planlamada, aktarım sayısının toplam yolculuk süresini olumsuz yönde etkileyeceği söylenebilir.

Aktarım sayısının yürüme mesafesine etkisi: Çoğu toplu taşıma ağında, farklı ulaşım modlarına ait duraklar aynı noktada bulunmamaktadır. Dolayısı ile ulaşım modları arasında yapılacak aktarımlar, yolcuların sonraki hatta binebilmeleri için belli bir mesafe yürümelerini gerektirecektir. Bu durum sadece modlar arası aktarımlarda değil çoğu zaman aynı moddaki vasıtalar arasındaki aktarımda da söz konusu olabilmektedir.

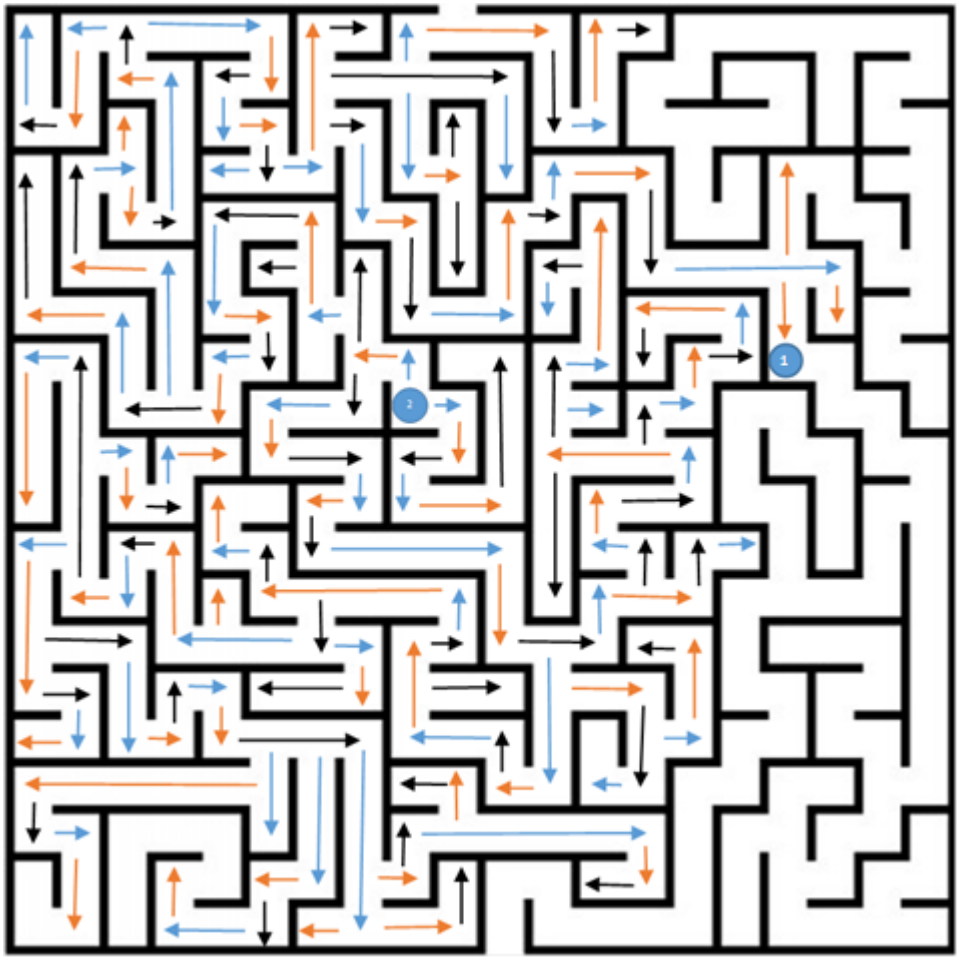
Yukarıda bahsedilen sebeplerle, toplu taşıma ağları üzerinde yüksek kaliteli yolculuk çözümleri üretebilmek için aktarım sayısının optimize edilmesi birinci derecede öneme sahiptir. Bu amaçla bu çalışmada, toplu taşıma ağları üzerinde iki nokta arasında bir yolculuk çözümü üretilirken, bu iki nokta arasında en az aktarım gerektirecek çözümlere odaklanan bir yaklaşım geliştirilmiştir. Bununla birlikte, toplam yolculuk süresi ve/veya toplam maliyeti mümkün olan en az aktarımlı çözümden daha düşük iken mümkün olan en az aktarımdan yalnızca bir fazla aktarım gerektiren çözümler de optimum çözüm olarak kabul edilmektedir. Ayrıca çözüm algoritmasında arama uzayı maksimum 5 aktarımla sınırlandırılmıştır.

2.2. İki Nokta Arasında Mümkün Olan En Az Aktarımlı Yolun Bulunması İçin Ters Akım Yöntemi

Ters akım yöntemi, bir toplu taşıma ağında, iki nokta arasındaki mümkün olan en az aktarımlı yolu bulmayı amaçlamaktadır.

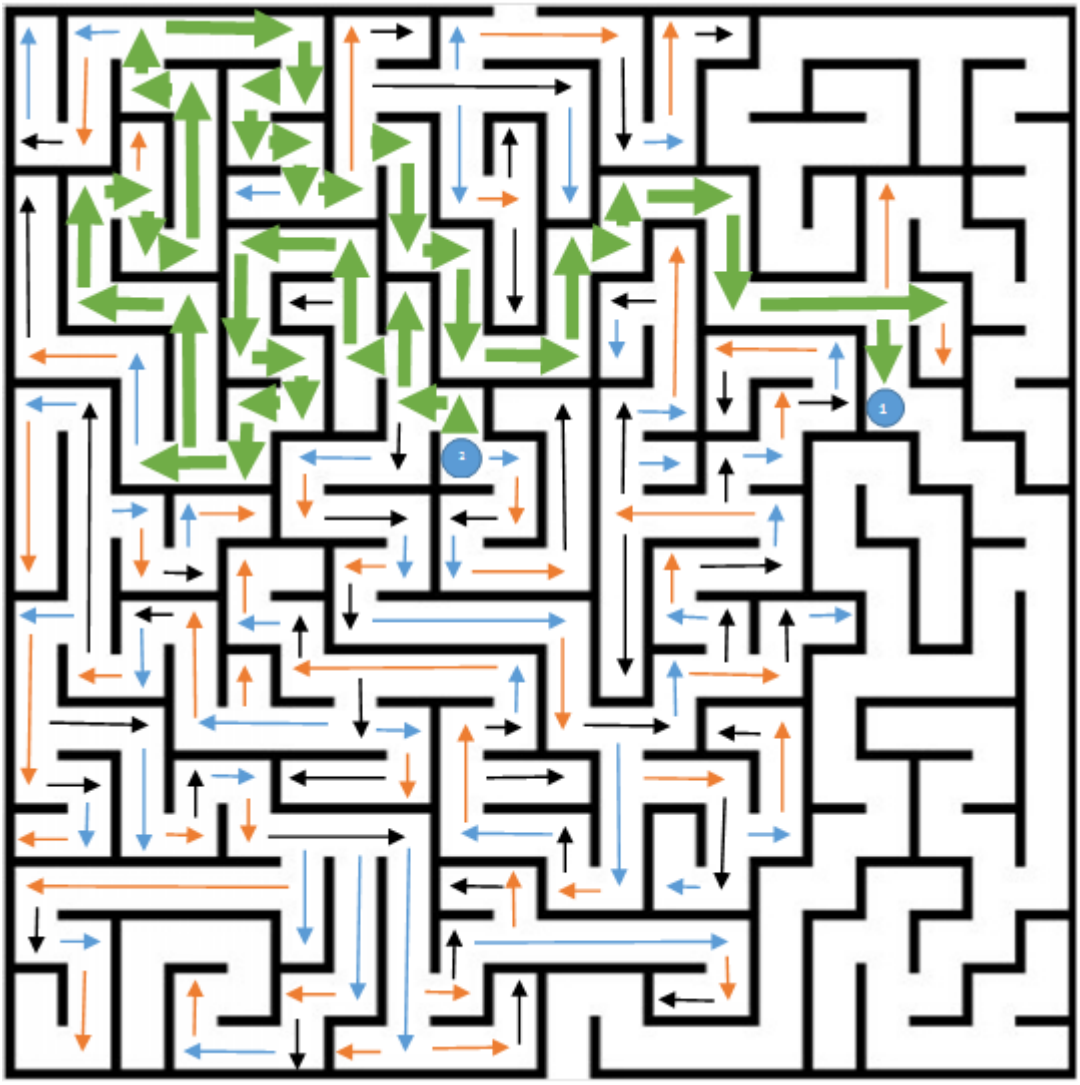
Ters akım algoritması Şekil 24'deki labirent çözüm algoritmasından esinlenilerek geliştirilmiştir. Labirentte belirlenen iki nokta arasında yol bulmak için varış noktasından başlanarak mümkün olan tüm yollar, her adımda bir sonraki dönemeç ya da yol ayrımına kadar olmak üzere adım adım oklarla işaretlenmiştir. Her adımda işaret oklarından açık olan (yola devam edebilen) uçlar farklı bir renkte okla devam ettirilerek alternatif yolların eş zamanlı olarak ilerletilmesi sağlanmıştır (Genişlik öncelikli arama algoritması). Algoritmada her açık uç devam edemeyeceği bir noktaya (labiret çıkmazı) veya başlangıç noktasına ulaşıncaya kadar devam ettirilmiştir. Burada ana amaç varış noktasından başlangıç noktasına ulaşabilen yolların ters oklarla işaretlenmesidir. Böylece başlangıç noktasından harekete başlanarak ters yönde oklar takip edildiğinde doğrudan varış noktasına ulaşmak mümkün olmaktadır (Şekil 25).

Şekil 24: Ters Akım Algoritmasının Labirent Çözümü Adımları



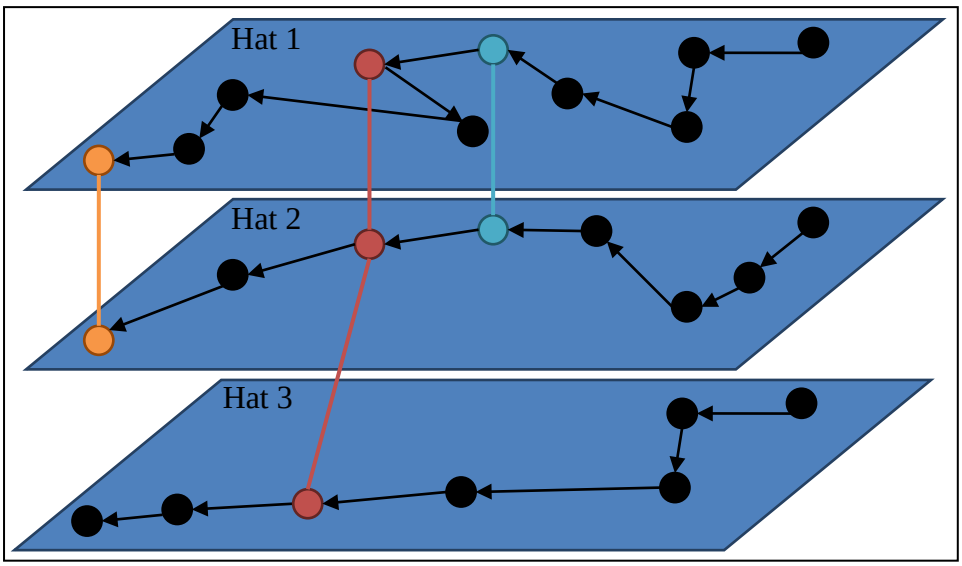
Yukarıda bahsedilen labirent çözümü, toplu taşıma ağlarına uygulanırken bir toplu taşıma hattının durakları labirentteki dönemeçler gibi kullanılmıştır. Bir duraktan başka hatlar geçiyorsa bu durak bir yol ayrımı olarak kabul edilmiş ve algoritma bu hatlar üzerinden ilerletilmiştir. Labirentin tek bir düzleme sahip olmasından dolayı labirent problemi tek boyutludur. Toplu taşıma ağlarında ise iki durak arasında birden fazla hat ile yolculuk mümkün olduğundan problem çok boyutlu hale gelmektedir. Bu sebeple her bir hat ayrı bir düzlem olarak düşünülmüştür. Ortak duraklardan geçen 3 hat için örnek bir model Şekil 26'daki gibi oluşturulabilir. Modelde hatların ortak durakları aynı renkle gösterilmiştir. Turuncu ve mavi düğümler arasında iki hat üzerinden seyahat etmek mümkündür. Yeşil düğümler ise labirentteki yol ayrımlarını temsil etmektedir. Turuncu ve mavi düğümlerin de birer yol ayrımı olduğu modelde görülmektedir.

Şekil 25: Ters Akım Algoritmasının Labirent Çözümü



Modele göre birinci hattın ilk düğümünden son düğümüne doğrudan Hat 1 üzerinden ulaşmak mümkünken turuncu ve mavi düğümler kullanılarak Hat 1 ve Hat 2 üzerinden aktarımlı ulaşmak da mümkündür. Model üzerinden benzer birçok senaryo üretilebilir. Gerçek toplu taşıma ağlarında da bu tür senaryolarla sıklıkla karşılaşılabilmektedir.

Şekil 26: Örnek Bir Toplu Taşıma Ağı Modeli



Ters akım algoritması ile toplu ulaşım ağlarında iki düğüm arasında optimum yolculuk çözümü aranırken, hedef düğüme ulaşılabilen tüm düğümler, bu düğümden hedef düğüme mümkün olan en az aktarım sayısı ile etiketlenirler. Binlerce hattı ve on binlerce durağı bulunan büyük ağlarda ise bu algoritmayı koşturmak için uygun bir veri yapısına ihtiyaç duyulmaktadır.

Bu çalışmada, ters akım algoritmasını ve yöntemin diğer adımlarını uygulamak üzere hatlar ve hatların geçtiği durakların işlenebilmesi için Tablo 1'deki veri yapısı geliştirilmiştir. Bu veri yapısında bir hattın gidiş ve dönüş güzergâhları ayrı birer hat olarak düşünülmüş ve veriler, veri tabanına bu şekilde işlenmiştir.

Ters akım algoritması toplu taşıma ağlarına uygulanırken, yalnız aynı duraktan geçen hatlar arasında aktarıma izin verildiğinde, özellikle modlar arası aktarımlar hesaplanamamaktadır. Bunun nedeni farklı ulaşım modlarına ait durakların farklı noktalara kurulu olmasıdır. Dolayısı ile farklı ulaşım moduna ait hatların ortak duraktan geçmeleri olanaksızdır (Metro hattı ile otobüs hattı gibi). Bu sebeple ters akım algoritmasının çalışmanın konusu olan probleme uygulanmasında, birbirine yürüme mesafesinde olan duraklar arasında sanal bağlantılar tanımlama yoluna gidilmiştir. Böylece ortak durakları olmasa da bir hattın duraklarına yürüme mesafesindeki duraklardan geçen hatlar da bu

sanal bağlantılarla birbirlerine bağlanmış ve bu tür hatlar arasında aktarım imkânı sağlanmıştır.

Tablo 1: Hat Güzergâhları için Veri Yapısı

Alan Adı	**Açıklama**
Hat No	Her hat ve bu hattın her güzergahı için birbirinden farklı hat numarası.
Durak Sıra No	Her hat güzergahı için, hattın geçtiği durağın sıra numarası.
Durak No	Toplu ulaşım ağında bulunan tüm duraklar için tekil olacak şekilde oluşturulmuş durak numarası. Aynı duraktan geçen hatlar için Durak No alanı aynı olurken Durak Sıra No alanı bir birinden faklı olabilir.
Mesafe	Durağın hattın başlangıç durağına mesafesi. Bu alan duraklar arası mesafelerin toplamı şeklinde hesaplanmaktadır ve çözüm algoritmasının zamanlama adımında kullanılmaktadır.

Sanal bağlantıların algoritmaya tümüyle dâhil edilmesi zaten karmaşık olan ağ yapısını daha da karmaşık hale getirerek problemin zorluk derecesini artırmaktadır. Bu sebeple ters akım algoritması uygulanırken sanal bağlantılar ağa adım adım dâhil edilmektedir. Bu işlem için yeni bir veri yapısı oluşturularak birbirine yürüme mesafesinde olan durak çiftleri bu veri yapısına önceden işlenmiştir. Veri yapısı Tablo 2'deki gibidir.

Tablo 2: Yürüme Mesafesindeki Durak Çiftleri için Veri Yapısı

Alan Adı	**Açıklama**
Durak Kodu 1	Birbirine yürüme mesafesindeki durakların durak kodları
Durak Kodu 2	
Mesafe	İki durak arasındaki mesafe

2.2.1. Ters Akım Algoritması İşleyiş Adımları

1. Adım:

Genel Aktarım Sayısı (GAS) değişkenine 1 ata.

2. Adım:

Varış durağını ve bu durağa yürüme mesafesinde olan durakları Varış Durakları (VD) listesine ekle

3. Adım: Döngü Başlangıcı

VD listesindeki duraklardan geçen hatları listele (VDGH)

4. Adım: Etiketleme

VDGH listesindeki tüm hatların VD listesindeki duraklarını –durak hattın ilk durağı değilse- Transfer Noktası (TN etiketi) olarak etiketle. Aynı durakların Aktarım Sayısı (AS) etiketini GAS olarak ata.

5. Adım: Etiketleme

Adım 3'te etiketlenen durakların durak numarasından küçük olan etiketlenmemiş durakların AS etiketini GAS + 1 olarak ata.

6. Adım:

VD ve VDGH listelerini temizle.

7. Adım:

AS etiketi GAS + 1 olan durakları ve bu duraklara yürüme mesafesindeki durakları VI listesine ekle.

8. Adım:

GAS değişkenini bir artır.

Başlangıç durağına AS etiketi atanmışsa işlemi sonlandır. Aksi halde 3. Adıma geri dön.

GAS değeri döngü içinde 7. Adımda artırıldığı için algoritma etiketlere atanan en büyük GAS değerinden daima bir fazla değerle sonlanır. GAS değerinin etiketlemede kullanılan en büyük değeri (EBGAS = GAS-1) ise yolculukta kullanılacak durak sayısına eşittir. Dolayısı ile algoritma sonucunda iki durak arasındaki yolculukta kullanılacak minimum hat sayısı EBGAS-1; minimum aktarım sayısı ise EBGAS-2 ile elde edilir.

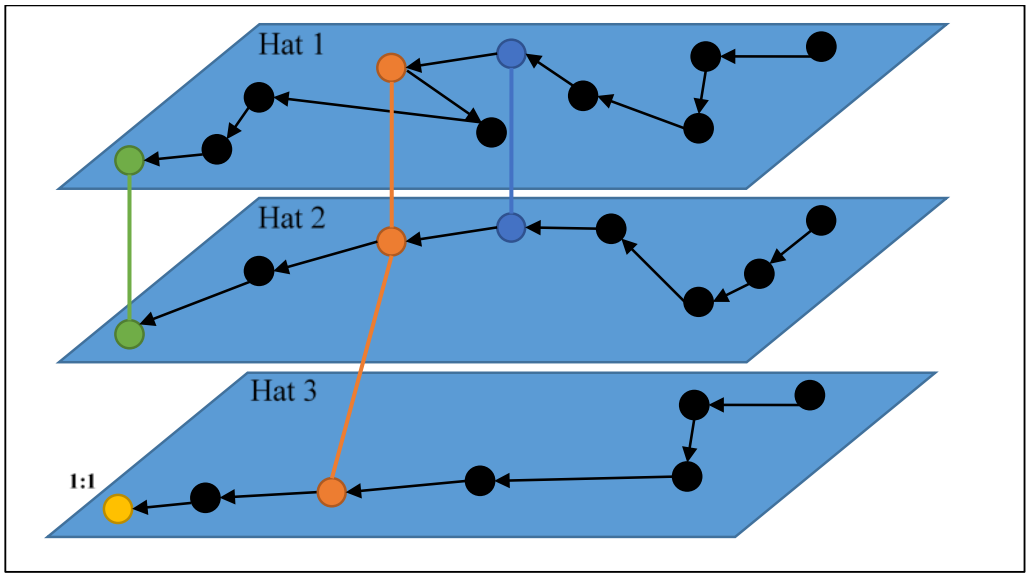
Hattın ilk durağının transfer noktası olarak etiketlenmemesinin sebebi, ilk durakta daha hat kullanılmadan başka bir hatta aktarım yapmanın anlamsız olmasıdır. Bu konu ileride daha ayrıntılı olarak ele alınmaktadır.

2.2.2. Ters Akım Algoritmasının Yürüme Bağlantıları Olmayan Örnek Bir Model Üzerinde İşleyişi

Ters akım algoritmasının örnek model olarak verilen ağ yapısı üzerindeki işleyişi adım adım aşağıdaki gibi gerçekleşir (Şekil 30). Bu örnek, Hat1'in ilk durağından Hat3'ün son durağına planlanan bir yolculuk için tasarlanmıştır ve yürüme mesafeleri göz ardı edilmiştir.

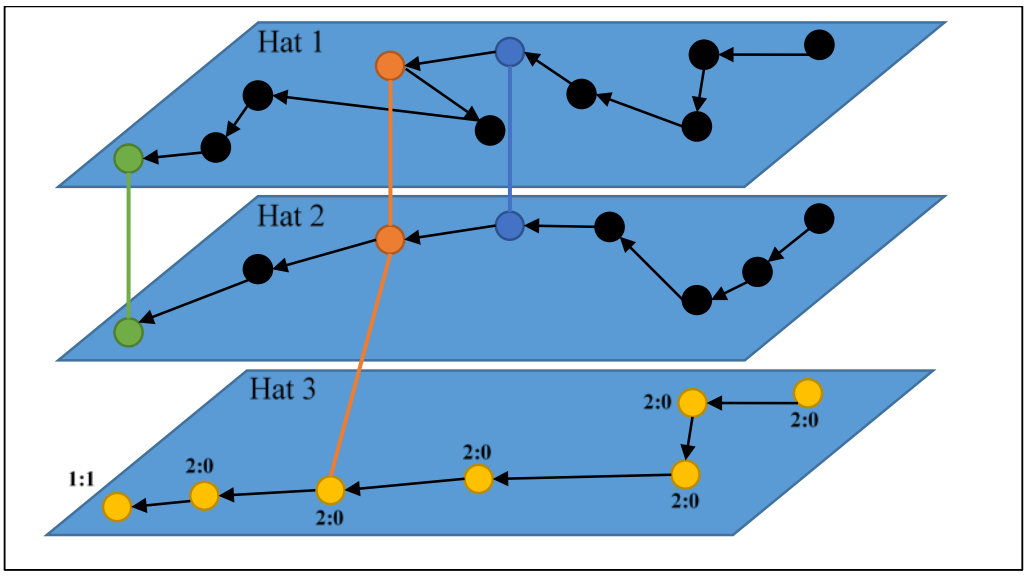
Şekil 27: Yürüme Bağlantısı Olmayan Örnek Modelde Ters Akım Algoritması

1. Döngü: Varış Duraklarının TN (Transfer Noktası) ve AS Etiketlerinin Atanması



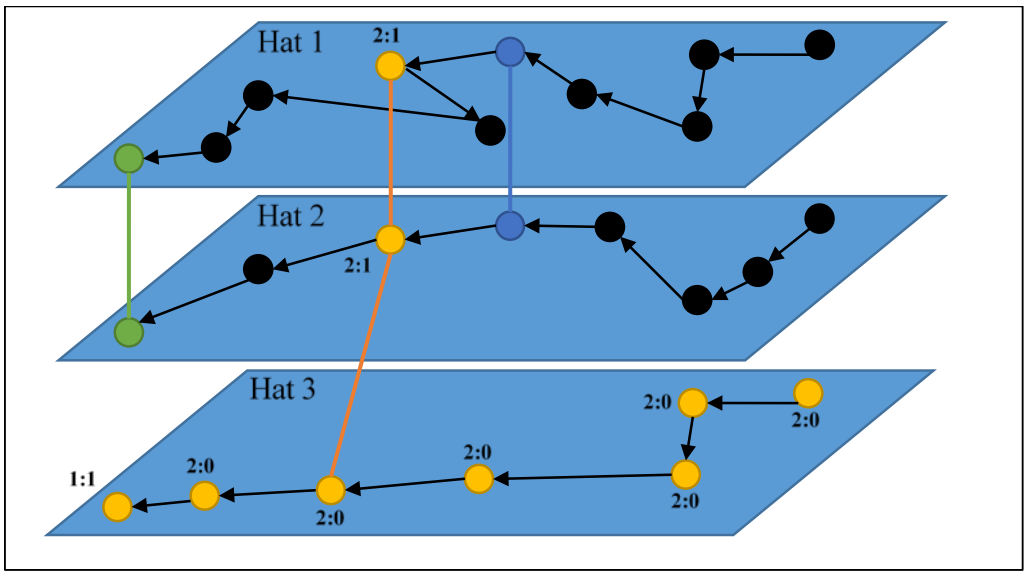
Not: İlk rakam AS ikinci rakam TN etiketini göstermektedir. Etiketlenen duraklar sarı renkle gösterilmiştir.

Şekil 28: Yürüme Bağlantısı Olmayan Örnek Modelde Ters Akım Algoritması 1. Döngü: Varış Durağından Geriye Doğru Etiketleme



Nihai durumda modele bakıldığında EBGAS değerinin 3 olduğu görülmektedir. Bu da örnek uygulama için iki durak arasında en az 2 hat ve bir aktarımla yolculuk yapılabileceği anlamına gelir. Algoritma işletimi ile elde edilen etiketler üzerinden iki durak arasındaki en az aktarımlı çözüm(ler)e ulaşmak ise oldukça kolaydır. Bunun için başlangıç durağının AS etiketine bakılır. Aynı hat üzerindeki AS etiketi küçük ve TN etiketi 1 olan duraklar aranır. Şekil 30'da bu durak tek ve etiketi 2:1 olan duraktır. Böylece aktarım durağı aradaki duraklara bakmaya gerek kalmadan kolayca bulunabilir. Sonraki aşamada ise aktarım durağından geçen ve durak AS etiketi 2 olan hatlarla çözüm devam eder. Şekil 30'da Hat 2 ve Hat 3 için aktarım durağı AS etiketi 2'dir. Ancak Hat 2'de ilgili duraktan sonra etiketlenen bir durak olmadığı için Hat 2 çözümde yer almaz. Hat 3 için ise aynı yol izlenerek varış durağına ulaşılır. Bu yöntem önerilen çözüm algoritmasının ikinci ayağını oluşturduğu için ilerleyen bölümlerde yöntem üzerinde daha ayrıntılı olarak durulmaktadır.

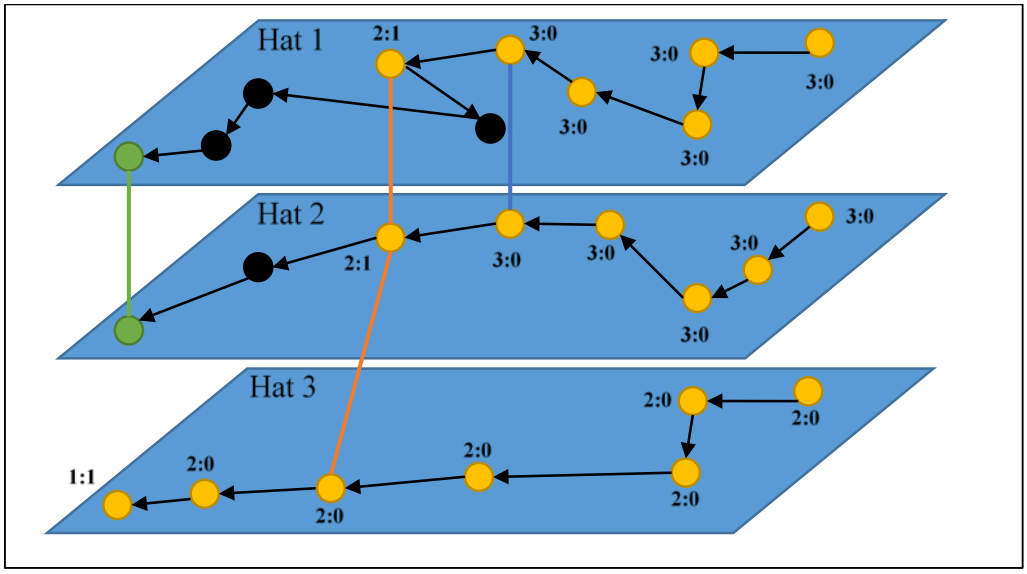
Şekil 29: 1.Yürüme Bağlantısı Olmayan Örnek Modelde Ters Akım Algoritması 2. Döngü: Varış Duraklarının TN ve AS Etiketlerinin Atanması



Ters akım algoritmasının en az aktarım odaklı çalıştığının bir göstergesi verilen örnekte açık bir şekilde ortaya çıkmaktadır. Mavi çizgi ile bağlanmış Hat 1 ve Hat 2 durakları incelendiğinde, örnekteki başlangıç ve varış durakları arasında bu durak üzerinden de yolculuk yapılabileceği görülmektedir. Ancak bu çözüm 3 hat ve iki aktarım gerektirecektir. Algoritma, etiketlenen düğümler üzerinde güncelleme yapmamaya yönelik geliştirildiği için 2 adımda çözüme ulaşılamaması ve döngünün devam etmesi durumunda bile bu hat duraklarının etiket değerleri değişmeyeceği için Hat 2 örnek durak çifti için hiçbir zaman çözüme giremeyecektir.

Ters akım algoritması toplu ulaşım ağındaki hat ve durakları, varış durağından başlayarak etiketlediği için varış durağına ulaşılamayacak durakları ve hatları çözüm dışı bırakmaktadır. Böylece çözüm algoritmasının ikinci aşamasındaki arama uzayı anlamlı bir ölçüde daraltmakta ve algoritmanın çok daha hızlı çözümler üretebilmesine olanak sağlamaktadır. Ayrıca algoritmanın yaptığı etiketleme ile ikinci aşamada nihai çözümleri üretecek algoritmanın verilen durak çiftleri arasındaki en az aktarımlı tüm yolları kolayca bulabilmesi sağlanmıştır. Bu yönleriyle ters akım algoritmasının bir önyükleme (bootstrap) adımı olduğu söylenebilir.

Şekil 30: Yürüme Bağlantısı Olmayan Örnek Modelde Ters Akım Algoritması 2. Döngü: Varış Durağından Geriye Doğru Etiketleme



Ters akım algoritması tek-kaynak en kısa yol algoritmalarında olduğu gibi bir başlangıç düğümünden belirlenen bir maksimum aktarım sayısına kadar erişilebilen tüm düğümleri etiketler. Algoritmanın tüm düğümler etiketleninceye kadar durdurulmadan çalışması halinde, ağdaki tüm düğümlerin AS-2 değerleri, kaynak düğümden etiketlenen düğüme an az kaç aktarımla ulaşılabileceğini gösterecektir.

2.2.3. Ters Akım Algoritmasının Yürüme Bağlantısı Olmayan Örnek Model İçin Hat Güzergâhı Veri Yapısı Üzerinden Uygulaması

Yukarıda bahsedilen örneğin Hat Güzergâhı veri yapısı ile uygulanması için gerekli olan güzergâh veri seti Tablo 3'deki gibi olmalıdır. Ters akım algoritmasında mesafe alanı kullanılmadığından tablodan bu alan çıkartılmıştır. Ayrıca algoritmanın ihtiyaç duyduğu etiket alanları da tabloya alan olarak eklenmiş ve etiket alanlarının başlangıç değerleri sıfır olarak atanmıştır. Algoritmanın her adımında yapılan etiketleme, sırasıyla Tablo 4 ve Tablo 5'de gösterilmiştir.

Tablo 3: Yürüme Bağlantısı Olmayan Örnek Model için Hat Güzergâhı Tablosu

Hat No	Durak Sıra No	Durak No	AS Etiketi	TN Etiketi
1	1	1*	0	0
1	2	2	0	0
1	3	3	0	0
1	4	4	0	0
1	5	5	0	0
1	6	6	0	0
1	7	7	0	0
1	8	8	0	0
1	9	9	0	0
1	10	10	0	0
2	1	11	0	0
2	2	12	0	0
2	3	13	0	0
2	4	14	0	0
2	5	5	0	0
2	6	6	0	0
2	7	15	0	0
2	8	10	0	0
3	1	16	0	0
3	2	17	0	0
3	3	18	0	0
3	4	19	0	0
3	5	6	0	0
3	6	20	0	0
3	7	21*	0	0

Not: Tabloda hatların ortak durakları aynı durak numarası ile belirtilmiş ve ilgili satırlar aynı renkle gösterilmiştir. Ayrıca başlangıç durağı ve varış durağı * ile işaretlenmiştir.

Tablo 4:Yürüme Bağlantısı Olmayan Örnek Modelde Ters Akım Algoritması

1. Döngü: Varış Durağı Etiketleme ve Geriye Doğru Yayılma

Hat No	Durak Sıra No	Durak No	AS Etiketi	TN Etiketi
1	1	1*	0	0
1	2	2	0	0
1	3	3	0	0
1	4	4	0	0
1	5	5	0	0
1	6	6	0	0
1	7	7	0	0
1	8	8	0	0
1	9	9	0	0
1	10	10	0	0
2	1	11	0	0
2	2	12	0	0
2	3	13	0	0
2	4	14	0	0
2	5	5	0	0
2	6	6	0	0
2	7	15	0	0
2	8	10	0	0
3	1	16	2	0
3	2	17	2	0
3	3	18	2	0
3	4	19	2	0
3	5	6	2	0
3	6	20	2	0
3	7	21*	1	1

Not: Tabloda etiket atamaları sarı renkle gösterilmiştir.

Tablo 5: Yürüme Bağlantısı Olmayan Örnek Modelde Ters Akım Algoritması 2. Döngü 1. Adım: Etiketlenen duraklardan geçen hatların aktarım duraklarının etiketlenmesi.

Hat No	Durak Sıra No	Durak No	AS Etiketi	TN Etiketi
1	1	1*	3	0
1	2	2	3	0
1	3	3	3	0
1	4	4	3	0
1	5	5	3	0
1	6	6	2	1
1	7	7	0	0
1	8	8	0	0
1	9	9	0	0
1	10	10	0	0
2	1	11	3	0
2	2	12	3	0
2	3	13	3	0
2	4	14	3	0
2	5	5	3	0
2	6	6	2	1
2	7	15	0	0
2	8	10	0	0
3	1	16	2	0
3	2	17	2	0
3	3	18	2	0
3	4	19	2	0
3	5	6	2	0
3	6	20	2	0
3	7	21*	1	1

Not: Bu adımda yapılan etiketlemeler koyu sarı olarak renklendirilmiştir.

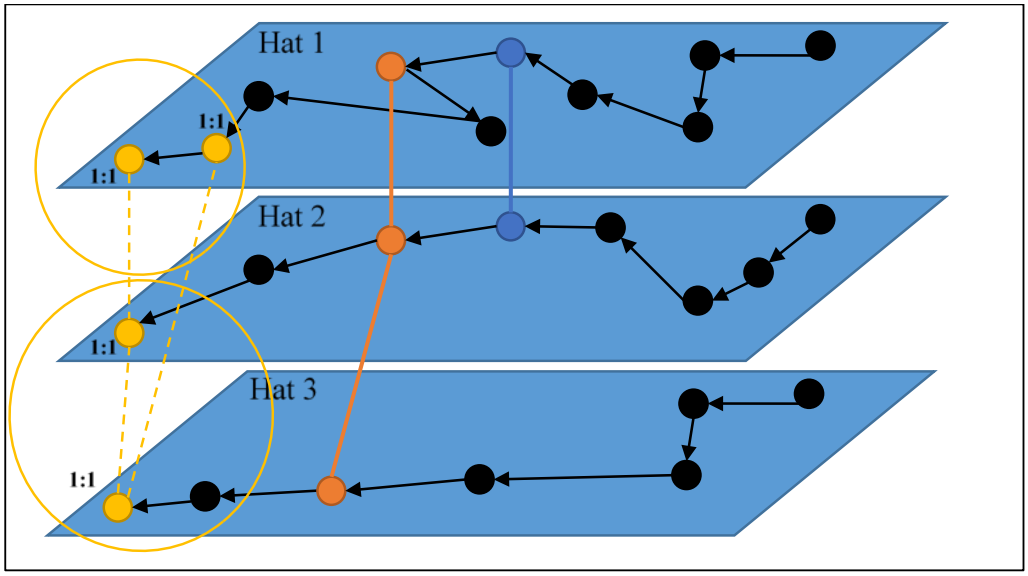
Tablo 5'den görüldüğü gibi başlangıç durağı 2. Döngü sonunda etiketlenerek algoritma sonlandırılmıştır. Tabloda etiketlenmeyen satırlar çözüme giremeyeceği için listeden çıkarılarak ikici aşamadaki arama uzayı daraltılabilir. Veritabanı işlemlerinde bir tablo üzerindeki belli bir koşula uyan satıra erişim süresi tablonun satır sayısıyla doğrudan ilişkili olduğu için bu işlem ikinci aşama işlemlerini hissedilir derecede hızlandıracaktır. Örnekte kullanılan model ağ yapısı çok küçük olduğundan etiketlenmeyen satır sayısı oldukça azdır. Gerçek toplu taşıma ağlarında ise güzergâh tablosunun boyutu çok daha büyüktür. Örneğin Bursa toplu taşıma ağında güzergâh tablosu yaklaşık 16 bin satır; İstanbul için ise yaklaşık 58 bin satırdır.

2.2.4. Ters Akım Algoritmasının Yürüme Bağlantıları Olan Örnek Bir Model Üzerindeki İşleyişi

Bu örnekte kullanılan model bir önceki örnek modelle aynı olmakla birlikte bu defa işleyiş, yürüme bağlantıları da hesaplanarak uygulanmaktadır. Yürüme bağlantıları, yalnız aktarım noktası olarak etiketlenen duraklar için hesaplanmaktadır. Aktarım durağı ile aynı hat üzerinde bulunan ve üzerinden başka bir hat geçmeyen yürüme mesafesindeki duraklara, yürüme bağlantısı eklenmemelidir. Çünkü aksi durum gereksiz yürüme adımlarının oluşmasına neden olur. Modellerde yürüme mesafesindeki duraklar sarı renkli çerçeve içinde, yürüme bağlantıları ise kesikli sarı çizgilerle gösterilmiştir (Şekil 31).

Örnek modelde Hat2 ve Hat3'ün son duraklarının birbirlerine yürüme mesafesinde olduğu varsayılmıştır. Hat2 ve Hat1'in son durakları aynı olduğuna göre bu duraktan Hat1'e de erişilebilir. Ayrıca Hat1'in sondan bir önceki durağının da Hat3'ün son durağına yürüme mesafesinde olduğu varsayılmıştır.

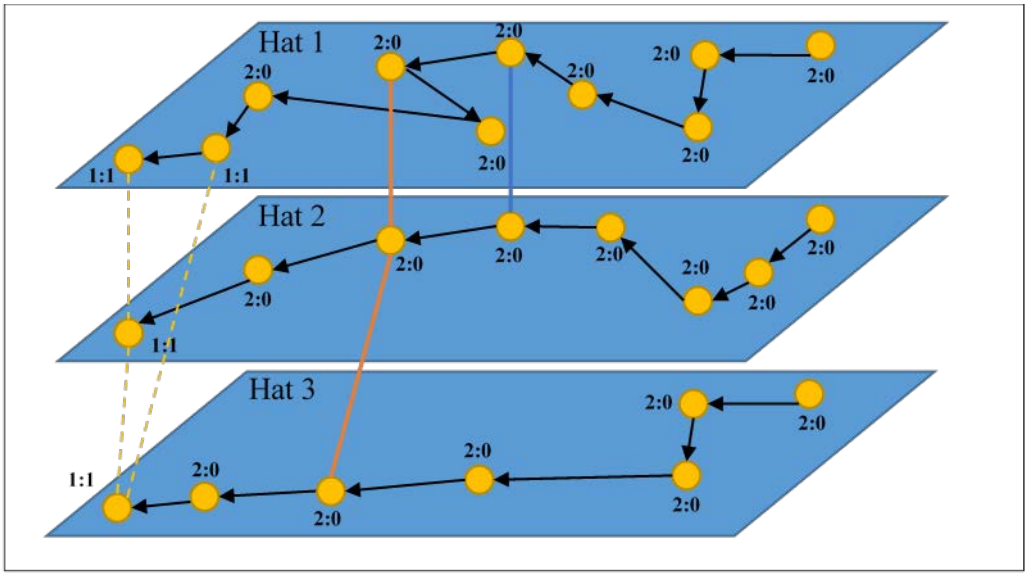
Şekil 31: Yürüme Bağlantılı Örnek Modelde Ters Akım Algoritması 1. Döngü: Varış Duraklarının ve Bu Duraklara Yürüme Mesafesindeki Durakların Etiketlenmesi



Not: İlk rakam AS ikinci rakam TN etiketini göstermektedir. Etiketlenen duraklar sarı renkle gösterilmiştir.

Şekil 32'deki modelin son durumunda görüldüğü gibi algoritmanın 1. Döngüsünde ağın tamamı etiketlenmiştir. Bu durum, yürüme bağlantılarıyla birlikte ağın her durağından varış durağına ulaşılabilmesinden kaynaklanmaktadır. Algoritma sonunda çözüme girebilen tek hat, Hat1'dir. Hat1'in aktarım durağı olarak etiketlenen iki durağında da varış durağına yürünebileceği için toplamda iki çözüm söz konusudur. Her iki çözüm de yürüme bağlantısı ile aktarımsız olarak varış durağına ulaşabilmiştir.

Şekil 32: Yürüme Bağlantılı Örnek Modelde Ters Akım Algoritması 1. Döngü: Varış Durağından Geriye Doğru Etiketleme



Algoritma sonucunda etiketlenmiş Hat Güzergâhları tablosu Tablo 6'daki gibi olmaktadır. Tabloda hatların ortak durakları aynı durak numarası ile belirtilmiş ve ilgili satırlar aynı renkle gösterilmiştir. Ayrıca başlangıç durağı ve varış durağı * ile işaretlenmiştir.

2.2.5. Ters Akım Algoritmasının Bir Fazla Aktarım Uyarlaması

En az aktarım yaklaşımı, genel anlamda problemin doğasına uygun olmakla birlikte aşağıdaki sözü edilen durumlarda, üretilen çözümlerin düşük kalitede olmasına, hatta zaman zaman problem kısıtlarına bağlı olarak çözüm üretilememesine neden olabilmektedir.

En Az Aktarım Yaklaşımının Uygun Olmadığı Durumlar;

Parabolik Güzergâhlı Hatlar ve Toplam Seyahat Süresi: Şekil 33'teki gibi bir toplu ulaşım ağında, A noktasından B noktasına Hat1 üzerinden aktarımsız seyahat edebilmek mümkün iken, Hat1 ve Hat2 ile aktarımlı seyahat seçeneğinin toplam mesafesinin daha kısa olacağı görülmektedir. Hatların zaman çizelgelerine de bağlı olmakla birlikte bu tür durumlarda bazen aktarımlı seçeneğin toplam seyahat süresi daha düşük olmaktadır.

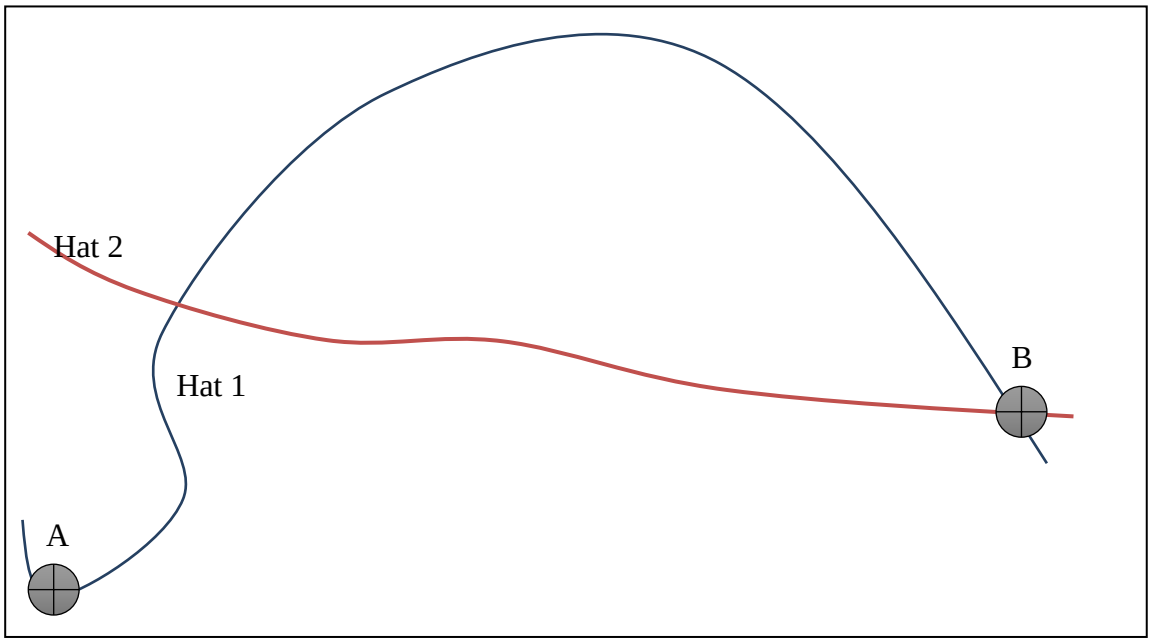
Tablo 6: Yürüme Bağlantılı Örnek Modelde Ters Akım Algoritması: Etiketlenmiş Hat Güzergâhları Tablosu

Hat No	Durak Sıra No	Durak No	AS Etiketi	TN Etiketi
1	1	1*	2	0
1	2	2	2	0
1	3	3	2	0
1	4	4	2	0
1	5	5	2	0
1	6	6	2	0
1	7	7	2	0
1	8	8	2	0
1	9	9	1	1
1	10	10	1	1
2	1	11	2	0
2	2	12	2	0
2	3	13	2	0
2	4	14	2	0
2	5	5	2	0
2	6	6	2	0
2	7	15	2	0
2	8	10	1	1
3	1	16	2	0
3	2	17	2	0
3	3	18	2	0
3	4	19	2	0
3	5	6	2	0
3	6	20	2	0
3	7	21*	1	1

Sefer Sıklığı Düşük Olan Hatlar: Toplu taşıma ağlarında bazı hatlar, mesai saati başlangıcı ve bitişi gibi yalnızca günün belli saatlerinde çalışmak üzere düşük sefer sıklığında çalışmaktadırlar. Bu saatler dışındaki bir zaman dilimi için planlanacak bir yolculuğun, sefer sıklığı düşük olan bir hat güzergâhı üzerinden geçmesi muhtemeldir.

Ters akım algoritmasında hatların hareket saatleri göz ardı edildiğinden (en az aktarım odaklı çözüm yaklaşımının bir sonucu olarak) sefer sıklığı düşük hatlar çözüme girebilmektedirler. Bu tür ön çözümler, optimizasyon algoritmasının ikinci aşamasındaki en fazla bekleme süresi kısıtına takılarak algoritmanın hiç çözüm üretememesine ya da kabul edilebilir olmayan (birkaç saatlik bekleme süreleri gibi) çözümler üretmesine sebep olmaktadır.

Şekil 33: Parabolik Güzergâhlı Hat Örneği



Aktarım İndirimleri ve Toplam Maliyet: Çoğu toplu ulaşım işletmesi, belli hatların (metro hatları gibi) kullanılırlığını artırmak amacıyla bu hatlara yapılan aktarmalarda ücret indirimleri uygulamaktadır. Uygulanan bu indirimler nadiren de olsa bazı durumlarda, daha çok aktarım içeren çözümün toplam maliyetinin en az aktarım içeren çözümlerin tamamı ya da birkaçının toplam maliyetinden daha düşük olmasına sebep olabilmektedir.

Yukarıda bahsi geçen sebeplerden dolayı uygulamada kullanılan ters akım algoritması, iki durak arasında mümkün olan en az aktarım sayısından bir fazla aktarım gerektirecek seçeneklerin de etiketlemesini sağlayacak şekilde değiştirilmiştir. Aslında bu değişiklik için algoritma döngüsünü, başlangıç durağının etiketlenmesinden sonra bir kez daha çalıştırmak yeterli olabilir. Ancak bu ek döngü yalnızca başlangıç durağına ulaşamayan hatlar için uygulanmalıdır. Çünkü başlangıç durağına ulaşan hatların

etiketlenmeye devam edilmesi durumunda, algoritmanın etiket güncellememeye dayalı olması sebebiyle, etiketlenecek yeni durakların, başlangıç durağından önceki duraklar olacağı açıktır. Bu etiketleme işleminin ise çözüme bir katkı sağlaması söz konusu değildir. Aksine etiketlenecek durak sayısı artacağından algoritmanın ikici aşamasında elenecek satır sayısı azalacaktır.

2.3. Zaman Parametresiz Problemde Çözüm Adımlarının Oluşturulması

Ters akım algoritması ile etiketlenen ağ üzerinden çözümlerin bulunması işlemine daha önce kısaca değinilmişti. Bu bölümde çözüm adımlarının oluşturulmasını sağlayacak algoritma zaman parametresi ve yürüme mesafeleriyle birlikte daha geniş bir şekilde ele alınmaktadır.

Çözüm adımlarının, zamanlama ve yürüme mesafeleriyle birlikte nasıl hesaplanacağına geçmeden önce, bu işlemle oluşturulacak çözüm adımlarının saklanması için gerekli olan veri yapısının oluşturulması gerekmektedir. Bu çalışmada çözüm adımları için Tablo 7'deki veri yapısı kullanılmıştır.

Veri yapısındaki her satır, toplam çözümün muhtemel bir adımını göstermektedir. Adım numarası 1 olan satırlar, başlangıç durağından yada bu durağa yürüme mesafesindeki bir duraktan geçen hatları göstermektedir. Ters akım algoritması, algoritmada etiketlenen bu hatlardan varış durağına doğrudan ya da aktarmalı olarak ulaşılabileceğini garanti etmektedir. Çünkü varış durağına ulaşılamayan hatlar algoritma tarafından zaten elenmiş olmaktadır.

Tablo 7: Zaman Parametresiz Çözüm Adımları İçin Veri Yapısı

Alan Adı	Açıklama
Hat No	Her hat ve bu hattın her güzergâhı için bir birinden farklı hat numarasıdır.
Aktarım Sayısı	Çözüm adımının dâhil olacağı çözümün toplam aktarım sayısıdır.
Adım Numarası	Çözüm adımının çözümdeki sıra numarasıdır.
Yolculuk Başlangıç Durağı	Bu adım için yolculuğun başlayacağı durak numarasıdır. Bu durak hat üzerinde olmayabilir. Yürüme mesafesindeki durak aktarımları için kullanılmaktadır.
Hat Hareket Durağına Yürüme Mesafesi	Yolculuk Başlangıç Durağı ile Hat Hareket Durağı arasındaki mesafedir.
Hat Hareket Durağı	Yolcunun hatta bineceği durak numarasıdır.
Hat Hareket Durağı Sıra Numarası	Yolcunun hatta bineceği durağın hat güzergâhındaki sıra numarasıdır.
Hat Varış (Aktarım) Durağı	Yolcunun hattan ineceği durak numarasıdır.
Hat Varış Durağı Sıra Numarası	Yolcunun hattan ineceği durağın hat güzergâhındaki sıra numarasıdır.

Çözüm adımlarının elde ediliş yöntemi anlatılırken ters akım algoritması tarafından etiketlenmeyen satırların etiketlenmiş hat güzergâh tablosundan silindiği varsayılmaktadır. Yöntem, genişlik öncelikli arama algoritmalarına benzer bir şekilde her adımda aynı aktarım seviyesindeki çözüm adımlarını araştırmaktadır.

Etiketlenen hat güzergâhları tablosu üzerinden başlangıç çözüm adımlarını elde etmek için öncelikle başlangıç durağından veya bu durağa yürüme mesafesindeki duraklardan geçen hatlar ele almaktadır. Başlangıç çözüm adımları bu hatların ilgili durağı ve hat üzerinde aktarım noktası olarak işaretlenen duraklardan oluşan çiftler ile oluşturulur. Eğer hat doğrudan başlangıç durağından geçiyor ise çözüm adımının Çözüm Adımları tablosundaki Yolculuk Başlangıç Durağı ve Hat Hareket Durağı alanları birbirine eşit olacak ve başlangıç durağı durak numarası olarak atanacaktır. Böyle bir durumda Hat Hareket Durağına Yürüme Mesafesi alanının değeri de elbette 0 (sıfır) olacaktır. Hattın doğrudan başlangıç durağından değil de ona yürüme mesafesinde olan başka bir duraktan geçmesi durumunda ise bu çözüm adımının Yolculuk Başlangıç Durağı alanı başlangıç durağının durak numarası olarak atanacak ve Hat Hareket Durağı alanı ise yolcunun ilgili hatta bineceği başlangıç durağına yürüme mesafesinde olan durağı gösterecektir.

Kural 1: Hat Varış Durağının Seçimi

Teorem 1: Monoton Azalmayan AS Etiketleri: *Ters akım algoritması tarafından etiketlenmiş bir hat durağının AS etiketi kendinden önce gelen ve aktarım noktası olmayan durakların AS etiketinden küçük olamaz. Ayrıca aktarım noktası olarak etiketlenen durakların AS etiket değerleri de kendinden önceki durakların AS etiket değerinden en fazla bir eksik olabilir.*

$$AS(D_{ij}) \leq AS(D_{ik})\ j: 1..n-1,\ k: j+1..n \text{ ve } TN((D_{ik}) \neq 1 \qquad (2.1)$$

D_{ij}: i'yinci hattın j'yinci. durağı,

n: i'yinci hattın etiketlenen durak sayısı

AS(D): D durağının AS etiket değeri

TN(D): D durağının aktarım noktası etiketi

İspat: *Algoritma bir aktarım noktasından geriye doğru hattın ilk durağına kadar etiketleme yaptığı ve etiketler üzerinde güncelleme yapılmadığı için hattın aktarım hattı olarak seçildiği adımda yapılan etiketleme, algoritma sonuna kadar sabit kalacaktır. Aynı hattın ikinci kez aktarım hattı olarak seçilmesi durumunda ise etiketleme yalnızca bir önce seçilen aktarım noktasından sonraki duraklar için uygulanacaktır. Etiketlemede kullanılan GAS değişkeninin değeri döngü boyunca arttığından yeni etiket değerlerinin önceki değerlerden küçük olması mümkün değildir.*

Hat Varış Durağı seçilirken ters akım algoritmasının Teorem1'deki özelliğinden yararlanılarak aynı hat üzerinde durak sıra numarası Hat Hareket Durağından büyük olan ve AS etiket değeri Hat Hareket Durağı etiket değerinden küçük olan durak seçilir. Böylece başlangıç çözüm adımları oluşturulmuş olur.

Kural 2: Aktarım Hatlarının Seçimi

Teorem 2: Açık Uçlu Dalların Elenmesi*: Aktarımlı bir yolculukta, nihai varış durağına henüz ulaşmamış her çözüm adımının Kural1'e göre seçilmiş varış durağından aktarım yapılabilecek en az bir hat vardır.*

İspat*: Ters akım algoritması geriye doğru yayıldığı için önceki adımlarda kullanılabilecek hatların etiketlemesi ancak ve ancak bu hatta geçiş yapılabilen en az bir, sonraki adım hattının etiketlenmiş olması ile mümkündür.*

MinAS: En az aktarım sayısı

VD_i: i'yinci çözüm adımında kullanılan hattın varış durağı

H_i: i'yinci çözüm adımında kullanılan hat

H_{in}: i'yinci hattın durakları olmak üzere;

MinAS > 1 iken

$$\exists H_i \mid VD_j \in H_{jn} \Rightarrow VD_j \in H_{in},\ j < i < MinAS \quad (2.2)$$

Teorem 3: Aktarım Hatlarının Nihai Varış Durağına Ulaşma Garantisi*: Teorem2'nin bir sonucu olarak olası her çözüm adımı, doğrudan ya da aktarım ile nihai varış durağına ulaşır.*

İspat*: Etiketlenen her durağa nihai varış durağından geriye yayılarak ulaşılmıştır.*

Teorem 4: *Teorem2 ve Teorem3, bir fazla aktarımlı çözümler aranırken etiketlenen hatlar için de geçerlidir.*

Aktarım hatlarının seçimi için ise benzer bir yol izlenir. İlk olarak önceki adımda Hat Varış Durağı olarak seçilen duraklardan ya da bu duraklara yürüme mesafesinde bulunan duraklardan geçen ve AS etiket değeri önceki adımda seçilen Hat Varış Durağı AS etiketine eşit olan hatlar ikinci adım hatları olarak seçilirler. Teorem2'de bu koşulları

sağlayan en az bir hattın varlığı gösterilmiştir. Ayrıca Teorem3'e göre ters akım algoritması, aktarım hatlarının nihai varış durağına ulaşabilmelerini garanti eder. Bu aşamadaki çözüm adımlarının Yolculuk Başlangıç Durağı Alanı bir önceki adımın Hat Varış Durağı olarak atanır. İlgili hat bir önceki adımda seçilen Hat Varış Durağından geçiyorsa, çözüm adımının Hat Hareket Durağı alanı da aynı değere atanır. Aksi durumda Hat Hareket Durağının değeri önceki adımda seçilen Hat Varış Durağına yürüme mesafesindeki duraklar olarak belirlenir. Hat Hareket Durağı yürüme mesafesindeki bir durak ise Hat Hareket Durağına Yürüme Mesafesi alanının değeri bu yürüme mesafesi olur,aksi halde bu alan sıfır olur. Hat Varış Durağı olarak ise durak numarası Hat Hareket Durağı Sıra Numarası alanından büyük olan ve AS etiket değeri bu durağın etiket değerinden küçük olan duraklar seçilir.

Kural 3: Durdurma Kriteri

Eğer seçilen Hat Varış Durakları ya da bu duraklara yürüme mesafesindeki duraklardan biri nihai varış durağı ise çözüm adımlarının tamamı elde edilmiş olur. En az aktarım sayısından bir fazla aktarımlı çözümler için döngü bir kez daha tekrar ettirilerek işlem tamamlanır.

2.3.1. Zaman Parametresiz Problemde Çözüm Adımlarının Hesaplanma Algoritması

1. Adım:
Boş bir Çözüm Adımları Listesi (ÇAL) oluştur.

2. Adım: Başlangıç Durakları Listesinin (BDL) oluşturulması
Başlangıç durağı ile BDL'yi oluştur.

3. Adım: Hareket Durakları Listesinin (HDL) oluşturulması (Döngü Başlangıcı)
BDL duraklarından ve bu duraklara yürüme mesafesindeki duraklardan geçen hatların ilgili durak satırlarını Hat Güzergâh tablosundan seçerek HDL'yi oluştur.

4. Adım: Varış (Aktarma) Durak Listesinin (VDL) oluşturulması

HDL'deki her satır için aynı hat üzerinde bulanan ve Durak Sıra Numarası alanı ilgili HDL satırındaki durak sıra numarası alanından büyük olan ve yine aynı hat üzerindeki AS etiket değeri küçük olan durakları eşleştirerek VDL'ni oluştur.

5. Adım: ÇAL'ın güncellenmesi

HDL ve VDL listeleri üzerinden hat numarası aynı olan satırların kartezyen çarpımları ile çözüm adımlarını oluşturarak ÇAL'a ekle.

6. Adım: BDL'nin güncellenmesi

VDL durakları ile tekrarsız bir yeni BDL oluştur.

7. Adım: Durdurma Kriteri ve Döngü Sonu

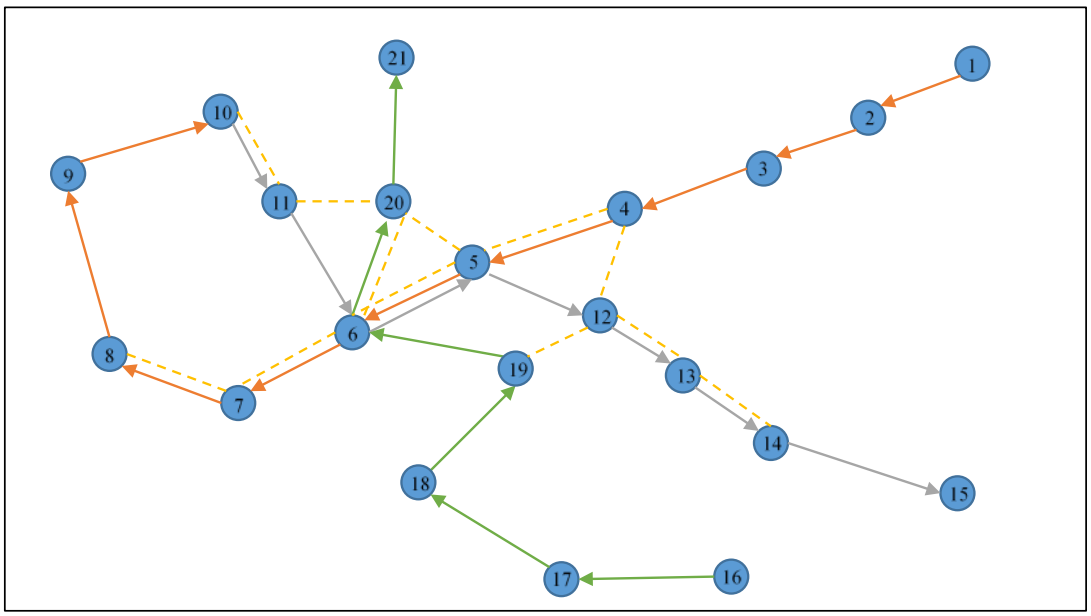
Varış durağı BDL'de varsa döngüyü sonlandır. Aksi durumda 3. Adıma dön.

Algoritmanın 5. Adımında bir durak üzerinde aynı AS etiket değerli birden fazla aktarım durağı olması halinde, bir Hareket Durağının birden çok Varış Durağı ile eşleşmesi söz konusu olacağından aynı hat üzerinden birden fazla çözüm adımı oluşacaktır.

2.3.2. Zaman Parametresiz Problemde Çözüm Adımlarının Örnek Bir Model Üzerinden Hesaplanması

Şekil 34'te gösterilen modelde her hat güzergâhı ayrı renkte oklarla gösterilmiştir. Yürüme bağlantıları ise kesikli sarı çizgilerle oluşturulmuştur. Örnek modelin Hat Güzergâhları ve Yürüme Mesafesindeki Durak Çiftleri sırasıyla Tablo 8 ve Tablo 9'daki gibidir. Başlangıç durak numarası 8 ve varış durak numarası ise 21 olarak belirlenmiştir.

Şekil 34: Zaman Parametresiz Problem için Örnek Ağ Modeli



Hat güzergâhları tablosunda (Tablo 8), algoritma döngülerinde etiketlenen satırlar döngü sırasına göre sarı, mavi ve yeşil renkte gösterilmiştir. Ayrıca her döngünün aktarım noktası etiketlemesi yapılan birinci adımında etiketlenen satırlar koyu renktedir. Etiket Açıklaması alanında ise aktarım noktası olarak etiketlenen satırların hangi kurala göre etiketlendiği açıklanmıştır. Yürünebilir açıklamasına sahip olan satırlar, bu durağın bir önceki adımda etiketlenen duraklardan hangisi ile yürüme mesafesinde olduğunu göstermektedir.

Hatların ortak duraklara sahip olmasına rağmen tüm aktarım noktalarının yürüme bağlantıları üzerinden etiketlenmesinin sebebi, durakların bir önce etiketlenen hattın ortak durağından önce yürüme mesafesindeki başka bir durakla bağlanmasıdır. Bu durum tamamen tesadüfi olarak ortaya çıkmıştır. Gerçek uygulamalarda hat durakları ortak durak oldukları için de etiketlenebilir. Tablonun mesafe alanı daha önce de ifade edildiği gibi hattın durakları arasındaki mesafelerin toplamları şeklinde oluşturulurken duraklar arası mesafe ise 500 ortalamalı 50 standart sapmalı normal dağılıma sahip rastlantısal olarak üretilen değerlerle belirlenmiştir. Yürüme mesafesindeki durak çiftleri tablosu da üretilen bu rastlantısal değerlere bağlı kalınarak oluşturulmuştur. Buna ilaveten yine rastlantısal olarak seçilen ve aynı hat üzerinde olmayan durak çiftleri yürüme mesafesinde kabul edilerek tabloya eklenmiştir.

Tablo 8: Örnek Ağ Modelinin Etiketlenmiş Hat Güzergâh Tablosu

Hat No	Durak Sıra No	Durak No	Mesafe	AS Etiketi	TN Etiketi	Etiket Açıklaması
1	1	1	0	3	0	
1	2	2	540	3	0	
1	3	3	1080	3	0	
1	4	4	1630	3	0	
1	5	5	2020	2	1	20. duraktan yürünebilir
1	6	6	2460	2	1	20. duraktan yürünebilir
1	7	7	2920	2	1	6. duraktan yürünebilir.
1	8	8	3340	4	0	
1	9	9	3890	4	0	
1	10	10	4390	3	1	11. duraktan yürünebilir.
2	1	10	0	3	0	
2	2	11	470	2	1	20. duraktan yürünebilir
2	3	6	1090	2	1	20. duraktan yürünebilir
2	4	5	1530	2	1	20. duraktan yürünebilir
2	5	12	2030	2	1	19. duraktan yürünebilir.
2	6	13	2470			
2	7	14	2920			
2	8	15	3530			
3	1	16	0	2	0	
3	2	17	500	2	0	
3	3	18	1060	2	0	
3	4	19	1590	2	0	
3	5	6	2110	2	0	
3	6	20	2560	2	0	
3	7	21	3100	1	1	

Başlangıç çözüm adımları oluşturulurken, yolculuğun başlangıç durağından ya da bu durağa yürüme mesafesindeki duraklardan geçen hatlar seçilmektedir. Etiketlenen hat güzergâhları tablosunda bu koşula uyan sadece bir satır vardır. Dolayısı ile başlangıç çözüm adımları sadece 1 numaralı hattı içermektedir. Başlangıç çözümüne giren bir hat

olduğu ve bu hat üzerinde yalnızca bir aktarım durağı olduğu için başlangıç çözümü tek olmaktadır. Çözüm Tablo 10'da gösterilmektedir.

Tablo 9: Örnek Ağ Modelinde Yürüme Mesafesindeki Durak Çiftleri Tablosu

Durak No 1	Durak No 2	Mesafe (m)
4	5	390
4	12	430
5	6	440
5	20	430
6	7	460
6	20	450
7	8	420
10	11	470
11	20	380
12	13	440
12	19	490
13	14	450

Not: Maksimum yürüme mesafesi 500 m olarak alınmıştır.

Tablo 10'da aktarım sayısı alanı EBGAS değeri 2 olarak hesaplanmıştır. Etiketlemiş hat güzergâhları tablosunun EBGAS değeri 4'tür. Yolculuk başlangıç durağı ve hat hareket durağı aynı olduğu için yürüme mesafesi 0 olarak atanmıştır.

Tablo 10: Örnek Problemin Başlangıç Adımı Çözümleri

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10

İkinci adım çözümlerinin oluşturulması için başlangıç çözümünün hat varış durağından ya da bu durağa yürüme mesafesinde olan duraklarından geçen hatlar

belirlenir. 11 numaralı durak başlangıç çözümünün hat varış durağı olan 10 numaralı durağa yürüme mesafesinde olduğu için bu duraklardan geçen ve AS etiketi başlangıç çözümünün hat varış durağı AS etiketine eşit olan hatlara bakılır. Tablo 8'de koşula uyan tek satır vardır. 2 numaralı hat 11 numaralı duraktan geçtiği halde bu satırın AS etiketi başlangıç çözümünün hat varış durağı AS etiketine eşit olmadığı için bu satır işleme katılmaz. Çünkü bu durak bir aktarım durağıdır. Teorem 1'de bir durağın AS etiketinin kendinden önce gelen durakların AS etiketinden küçük olması durumunda bu durağın bir aktarım durağı olacağı gösterilmişti. Zaten bir aktarım durağındayken, başka bir aktarım durağını yolculuk başlangıç durağı olarak seçmenin algoritmanın çözüm üretememesine sebep olma riski vardır. Örneğin hat üzerinde yalnızca bir tane aktarım durağı varsa, bu durağın yolculuk başlangıç durağı olarak seçilmesi durumunda Kural 1'e uygun bir varış durağı seçmek mümkün olmayacaktır. Kaldı ki algoritma bu duruma izin verse bile, bu yolculuk adımı Teorem 1'e göre aynı AS etiketine sahip iki durak arasında gerçekleşecektir. Dolayısı ile bu adım varış durağına yaklaşamayan gereksiz bir adım olacaktır. Bu durum ters akım algoritmasında bir hattın başlangıç durağının transfer noktası olarak işaretlenmeyişinin bir başka nedenidir. Böylece hat üzerinde en az bir adet transfer noktası olmayan durağın kalması garanti edilmektedir.

Ayrıca bu kural, anlamsız çözüm adımlarının oluşmasını engelleyerek algoritmanın gereksiz çalışmasına mani olmaktadır. Mevcut durumda ikinci adım çözüm sayısı 4 iken, aksi durumda bu sayının 7'ye çıkacağı etiketlenmiş hat güzergâhları tablosundan (Tablo 8) açıkça görülebilir.

Böylece ikinci adım çözümlerinin tek yolculuk başlangıç alanı, hat üzerindeki 4 aktarım durağıyla Kural 1'e göre eşleştirilerek 4 adet ikinci adım çözümü elde edilir. Çözümler, Tablo 11'de verilmektedir.

Tablo 11: Örnek Problemin 2. Adım Çözümleri

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
2	2	2	10	0	10	1	11	2
2	2	2	10	0	10	1	6	3
2	2	2	10	0	10	1	5	4
2	2	2	10	0	10	1	12	5

Üçüncü adım çözümleri de benzer şekilde oluşturulur. Öncelikle 11, 6, 5 ve 12 numaralı duraklardan ya da bu duraklara yürüme mesafesinde olan duraklardan geçen hatlar bulunur. Yürüme mesafesindeki duraklar da düşünüldüğünde durak listesi şu şekilde olur: 4, 5, 6, 7, 10, 11, 12, 13, 19, 20. Tablo 8'de bu duraklardan geçen ve AS etiketi 2 olan tek hat 3 numaralı hattır. Bu hatta ait 3 satır (6, 19 ve 20 numaralı duraklar) kurala uygun olarak seçilebilir. Hat üzerinde yalnız bir aktarım durağı vardır. Böylece son adım çözümleri Tablo 12'deki gibi oluşur.

Tablo 12: Örnek Problemin 3. Adım Çözümleri

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
3	2	3	11	380	20	6	21	7
3	2	3	6	0	6	5	21	7
3	2	3	6	450	20	6	21	7
3	2	3	5	430	20	6	21	7
3	2	3	5	440	6	5	21	7
3	2	3	12	490	19	4	21	7

2.3.3. Zaman Parametresiz Problemde Çözüm Adımlarından Toplam Yolculuk Çözümlerinin Hesaplanması

Yukarıda verilen çözüm adımlarını hesaplayan algoritma, iki nokta arasındaki toplam yolculuk çözümlerinin mümkün olan tüm adımlarını hesaplayabilmektedir.

Kural 4: Zaman Parametresiz Çözüm Adımlarının Birleştirilmesi

Çözüm adımları birleştirilirken i' inci çözüm adımının Hat Varış Durağı, (i + 1)'inci çözüm adımının Yolculuk Başlangıç Durağı ile aynı olan çözüm adımları eşleştirilir. Teorem 2'ye göre (i+1)'inci çözüm adımlarının en az birinin Yolculuk Başlangıç Durağı, i'inci çözüm adımındaki en az bir seçeneğin Hat Varış durağı ile eşleşecektir.

i' inci adımdaki bir seçeneğin (i+1)'inci adımda birden fazla seçenekle eşleşmesi durumunda, i' inci adımları aynı (i+1)'inci adımları farklı olan iki çözüm ortaya çıkacaktır. Benzer şekilde (i+1)'inci adımdaki bir seçeneğin i' inci adımdaki birden fazla seçenekle eşleşmesi durumda da i' inci adımları faklı fakat (i+1)'inci adımları aynı olan iki farklı çözüm oluşacaktır.

Çözüm adımlarından toplam yolculuk çözümlerinin elde edilmesi kombinatoryal bir problemdir. Problemin çözüm uzayı, aktarım (adım) sayısına ve her bir adımda eşleşen seçenek sayısına bağlı olarak geometrik bir şekilde büyümektedir.

Yukarıdaki örnekte kullanılan Tablo 8'deki etiketlenmiş hat güzergâhlarından elde edilen çözüm adımları toplu olarak Tablo 13'de verilmektedir.

Tablo 13: Örnek Problemin Çözüm Adımı Seçenekleri

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	11	2
2	2	2	10	0	10	1	6	3
2	2	2	10	0	10	1	5	4
2	2	2	10	0	10	1	12	5
3	2	3	11	380	20	6	21	7
3	2	3	6	0	6	5	21	7
3	2	3	6	450	20	6	21	7
3	2	3	5	430	20	6	21	7
3	2	3	5	440	6	5	21	7
3	2	3	12	490	19	4	21	7

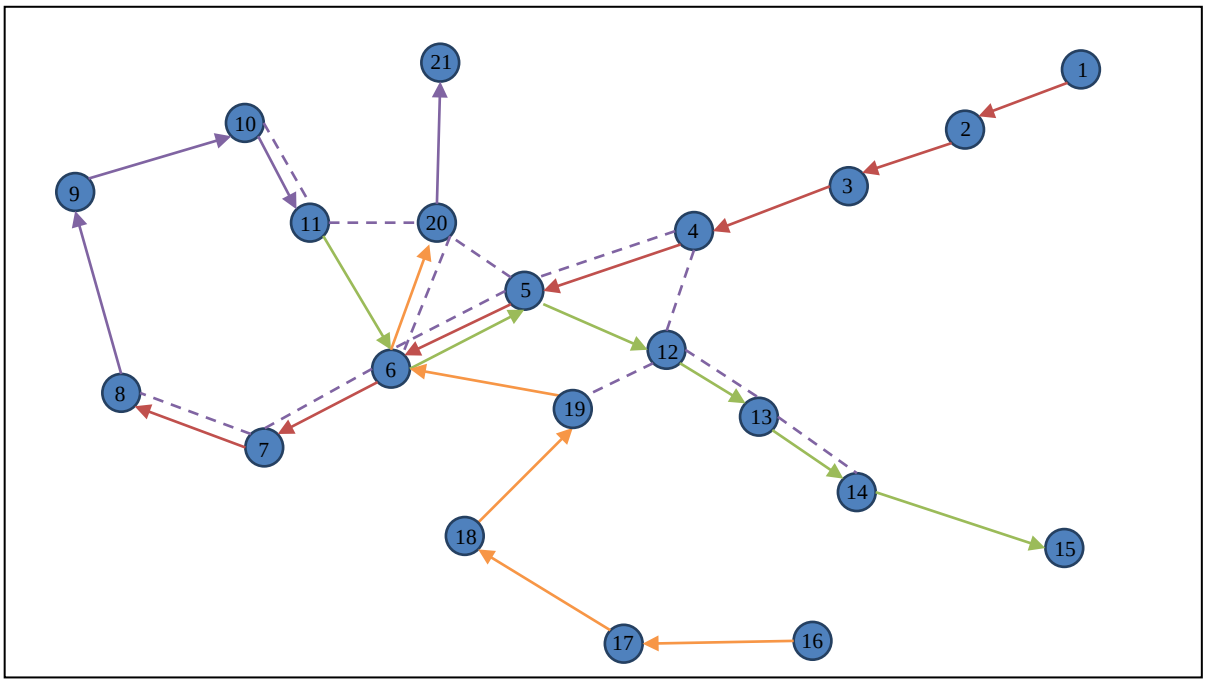
Örnek modelde başlangıç çözümü tek olduğu için toplam yolculuk çözümlerinin tümü aynı başlangıç çözümüne sahip olacaktır. Dolayısı ile ikinci adım çözümlerinin tümü başlangıç çözümünün hat varış durağından başlayacaktır. Tablo 13'de görüldüğü gibi ikinci adım çözümlerinin tümünde yolculuk başlangıç durağı, başlangıç çözümü adımının hat varış durağı olan 10 numaralı duraktır.

Tablo 13 üzerinden Kural 4'e göre oluşturulan toplam yolculuk çözümleri aşağıdaki Tablo 14-19'da verilmektedir. Ayrıca her çözüm Şekil 35-40'da model üzerinden sarı çizgilerle gösterilmiştir.

Tablo 14: Örnek Problem için 1. Çözüm

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	11	2
3	2	3	11	380	20	6	21	7

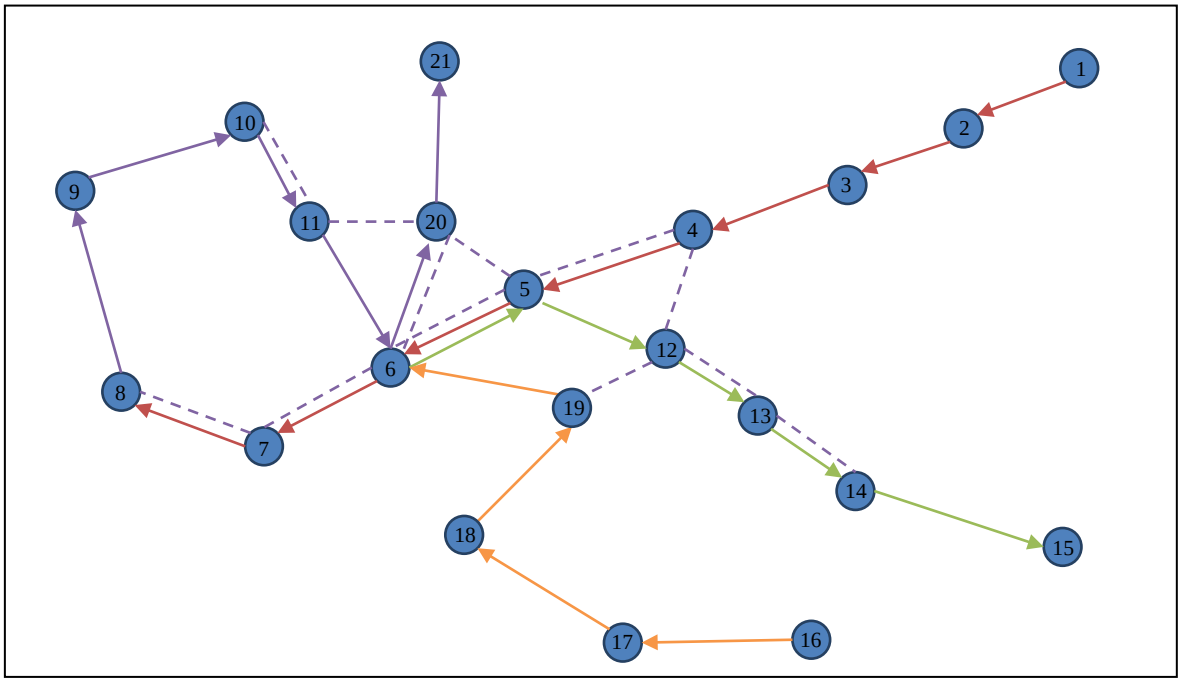
Şekil 35: Örnek Problem için 1. Çözümün Ağ Üzerinde Gösterimi



Tablo 15: Örnek Problem için 2. Çözüm

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	6	3
3	2	3	6	0	6	5	21	7

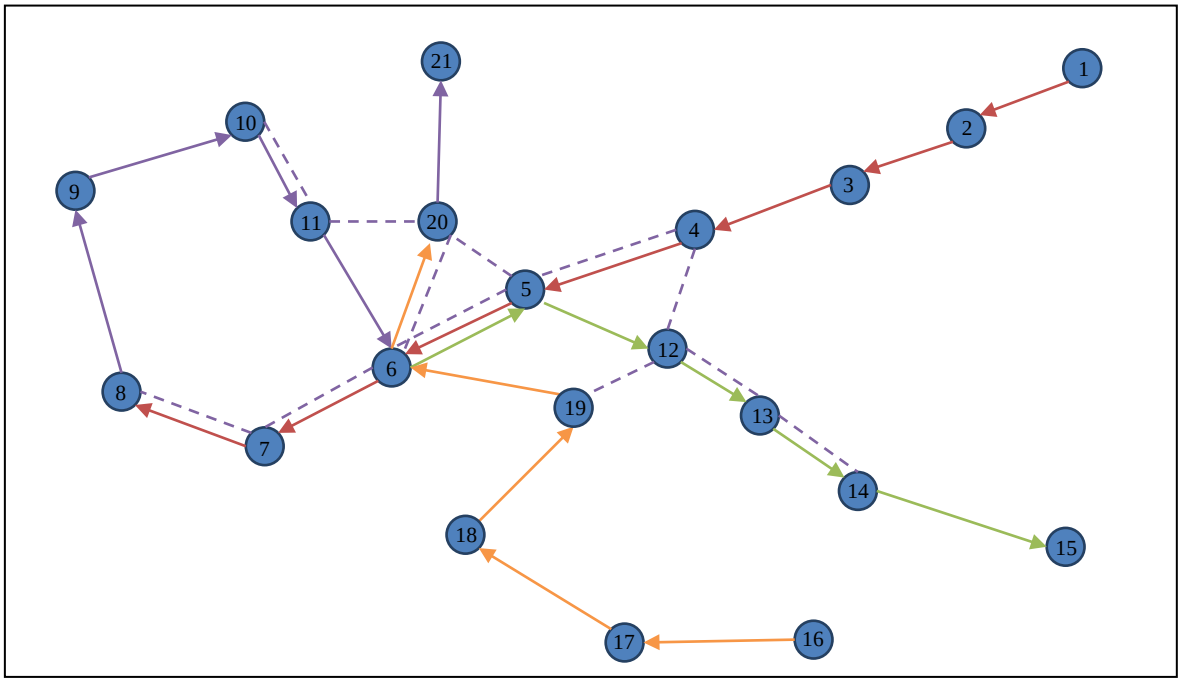
Şekil 36: Örnek Problem için 2. Çözümün Ağ Üzerinde Gösterimi



Tablo 16: Örnek Problem için 3. Çözüm

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	6	3
3	2	3	6	450	20	6	21	7

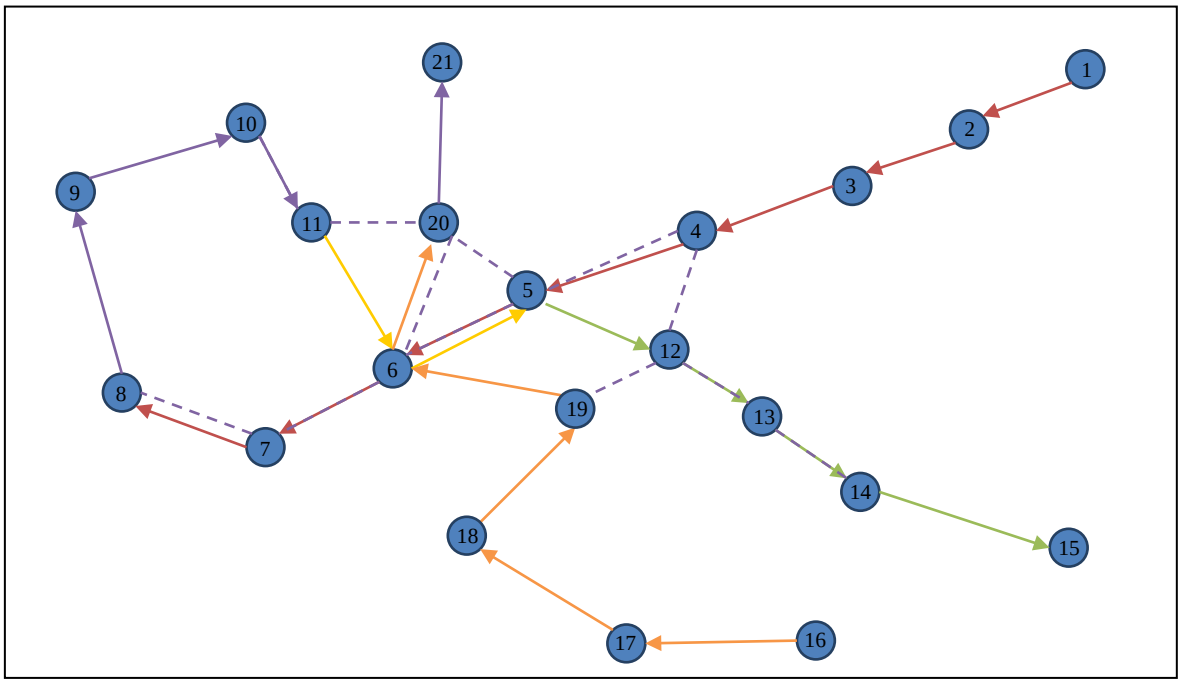
Şekil 37: Örnek Problem için 3. Çözümün Ağ Üzerinde Gösterimi



Tablo 17: Örnek Problem için 4. Çözüm

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	5	4
3	2	3	5	430	20	6	21	7

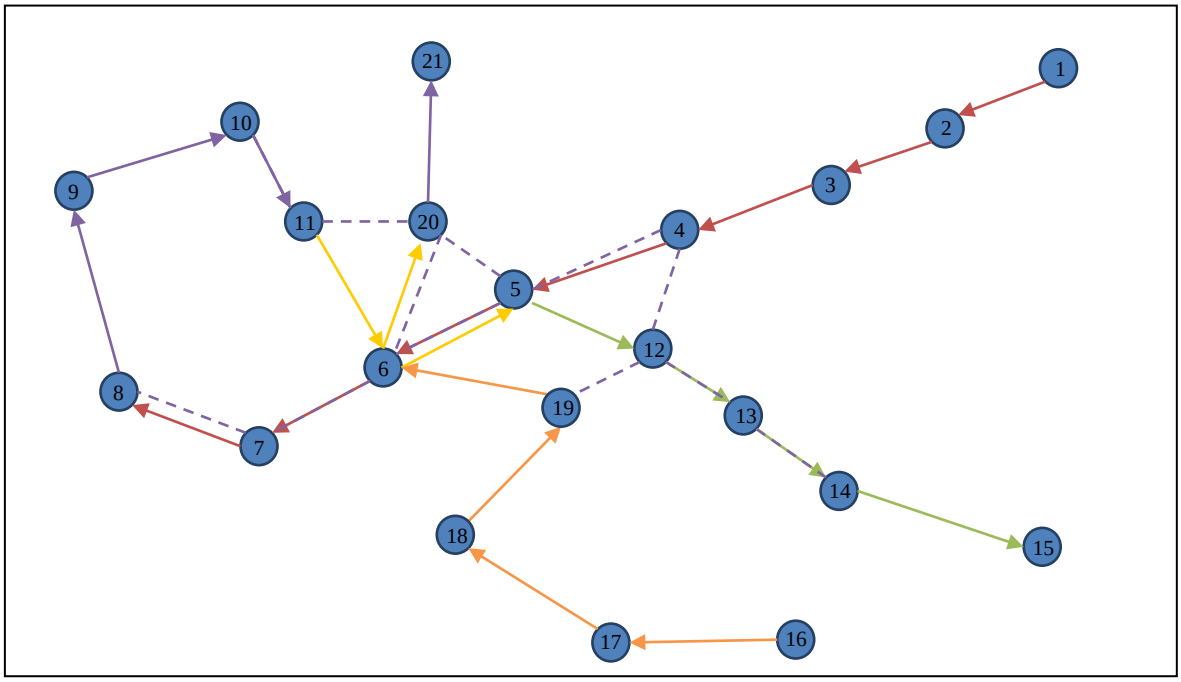
Şekil 38: Örnek Problem için 4. Çözümün Ağ Üzerinde Gösterimi



Tablo 18: Örnek Problem için 5. Çözüm

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi (m)	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	5	4
3	2	3	5	440	6	5	21	7

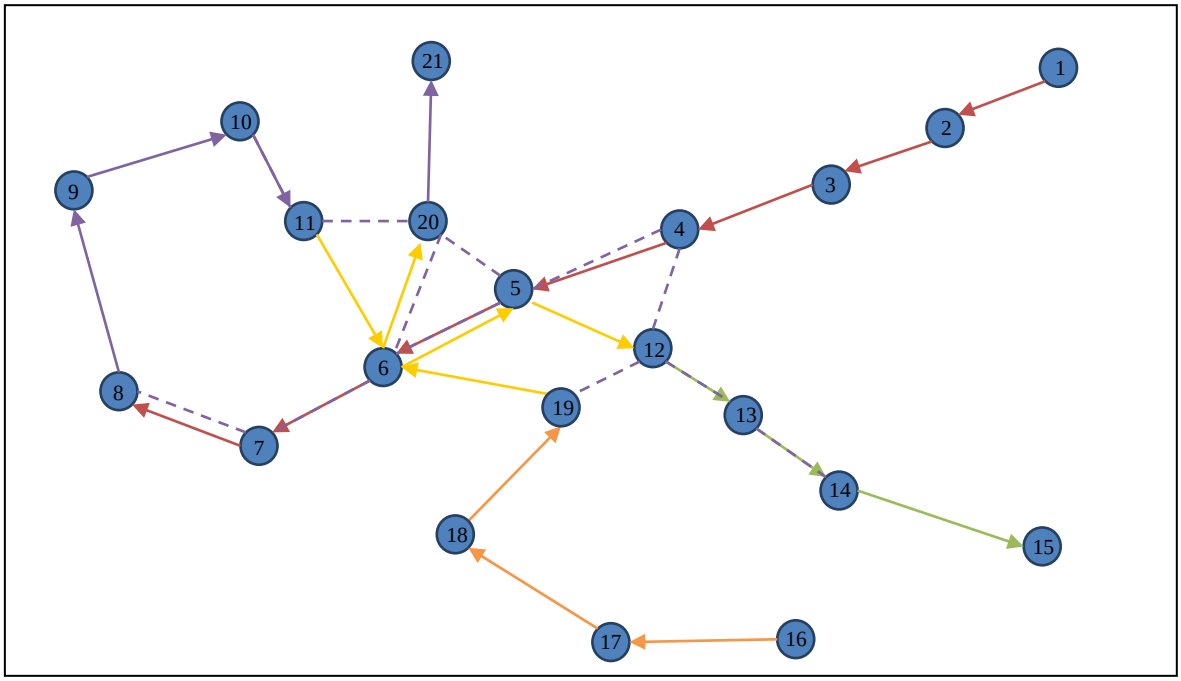
Şekil 39: Örnek Problem için 5. Çözümün Ağ Üzerinde Gösterimi



Tablo 19: Örnek Problem için 6. Çözüm

Hat No	Aktarım Sayısı	Adım Numarası	Yolculuk Başlangıç Durağı	Hat Hareket Durağına Yürüme Mesafesi	Hat Hareket Durağı	Hat Hareket Durağı Sıra No	Hat Varış Durağı	Hat Varış Durağı Sıra No
1	2	1	8	0	8	8	10	10
2	2	2	10	0	10	1	12	5
3	2	3	12	490	19	4	21	7

Şekil 40: Örnek Problem için 6. Çözümün Ağ Üzerinde Gösterimi



2.3.4. Zaman Parametreli Problemde Çözüm Adımlarının Zaman Alanlarının Hesaplanması

Çözüm adımlarının zaman alanlarını hesaplamaya geçmeden önce çözüm adımları veri yapısının bu alanları da kapsayacak şekilde güncellenmesi gerekmektedir. Yeni veri yapısı Tablo 20'deki gibi olmaktadır.

Bu veri yapısında süre alanı, Hat Hareket Durağı ve Hat Varış durağı arasındaki mesafenin (Mesafe Alanının) önceden belirlenmiş ortalama hıza bölünmesi ile hesaplanır. Bu çalışmada tüm hatlar için ortalama hız 40 km/s olarak alınmıştır.

Zaman parametreli yolculuk planlama probleminde başlangıç zaman parametresi, problem tanımında belirlenir. Önceki şekliyle bir başlangıç ve varış durağı çiftine göre üretilen çözümler, probleme zaman parametresinin eklenmesiyle, durak çifti değişkenlerine ek olarak başlangıç zamanı değişkenine göre hesaplanır. Başlangıç zamanı değişkeni gün ve saat bileşenlerinden oluşur. Bunun sebebi toplu taşıma ağı hatlarında seferlerin, gün ve saat bazında planlanıyor olmasıdır. Veri yapısındaki zaman alanlarının değerleri de bu başlangıç zamanı parametresine bağlı olarak üretilir.

Tablo 20: Zaman Alanları ile Güncellenmiş Çözüm Adımları Veri Yapısı

Alan Adı	Açıklama
Hat No	Her hat ve bu hattın her güzergâhı için bir birinden farklı hat numarası.
Aktarım Sayısı	Çözüm adımının dâhil olacağı çözümün toplam aktarım sayısı
Adım Numarası	Çözüm adımının çözümdeki sıra numarası
Yolculuk Başlangıç Durağı	Bu adım için yolculuğun başlayacağı durak numarası. Bu durak hat üzerinde olmayabilir. Yürüme mesafesindeki durak aktarımları için kullanılmaktadır.
Hat Hareket Durağına Yürüme Mesafesi	Yolculuk Başlangıç Durağı ile Hat Hareket Durağı arasındaki mesafe.
Hat Hareket Durağı	Yolcunun hatta bineceği durak numarasıdır.
Hat Hareket Durağı Sıra Numarası	Yolcunun hatta bineceği durağın hat güzergâhındaki sıra numarasıdır.
Hat Varış (Aktarım) Durağı	Yolcunun hattan ineceği durak numarasıdır.
Hat Varış Durağı Sıra Numarası	Yolcunun hattan ineceği durağın hat güzergâhındaki sıra numarasıdır.
Mesafe	Hat Hareket Durağı ile Hat Varış Durağı arasındaki toplam mesafedir.
Süre	Hat Hareket Durağı ile Hat Varış Durağı arasındaki toplam yolculuk süresidir.
Hat Hareket Durağına Varış Zamanı	Yolcunun başlangıç durağına varış zamanıdır.
Hat Hareket Zamanı	Hattın Hat Hareket Durağından ayrılış zamanıdır.
Hat Varış Zamanı	Hattın Hat Varış Durağına varış zamanıdır.

Kural 5: Başlangıç Çözüm Adımlarının Hat Hareket Durağına Varış Zamanının Hesaplanması

Başlangıç çözüm adımlarının Hat Hareket Durağına Varış Zamanı alanı, eğer ilgili çözüm adımının Hat Hareket Durağına Yürüme Mesafesi alanı sıfır ise başlangıç zamanı parametresinin değeri olarak atanır. Yürüme mesafesi sıfırdan farklı ise bu alan, başlangıç zamanına, yürüme süresi ilave edilerek hesaplanır. Bu çalışmada yürüme hızı 1m/sn olarak alınmıştır.

Kural 6: Sonraki Çözüm Adımları için Hat Hareket Durağına Varış Zamanının Hesaplanması

Hat Hareket Durağına Varış Zamanı alanı, başlangıç çözüm adımlarından sonraki çözüm adımları için hesaplanırken, başlangıç zamanı parametresi bir önceki çözüm adımının Hat Varış Zamanı olarak güncellenir. Önceki çözüm adımlarının hat varış

durağına ulaşma zamanları birbirinden farklı olabileceğinden bu çözüm adımlarındaki başlangıç zamanı parametresi, çözüm adımının yolculuk başlangıç durağına bağlı olarak, her çözüm adımı için farklı değerler alabilir. Bunun yanında önceki adımda aynı hat varış durağına farklı zamanlarda ulaşan çözüm adımları da olabilmektedir. Bu sebeple aynı seviyedeki çözüm adımları arasında, yolculuk başlangıç, hat hareket ve hat varış durakları aynı olan fakat zaman alanları birbirinden faklı olan çözüm adımı seçenekleri oluşabilmektedir. Hat Hareket Durağına Varış Zamanı alanının değeri hesaplanırken, başlangıç çözüm adımlarında olduğu gibi başlangıç zamanı parametresine yürüme sürelerinin eklenmesi gerektiği gözden kaçırılmamalıdır.

Kural 7: Hat Hareket Zamanının Hesaplanması

Hat Hareket Zamanı alanı, ilgili hattın zaman çizelgesine bakılarak belirlenir. Çoğu toplu ulaşım işletmesi, zaman çizelgelerinde sadece hatların başlangıç durağından hareket saatlerini ve hattın toplam süresini göstermektedir. Ancak, çözüm adımları hattın bir ara durağından başlayabileceğinden, zaman parametrelerinin hesaplanabilmesi için hatların güzergâhtaki her duraktan geçiş saatlerinin bilinmesi gerekir. Hatların zaman çizelgeleri için algoritmada ihtiyaç duyulan verileri tutacak veri yapısı Tablo 21'deki gibidir.

Tablo 21: Hat Zaman Çizelgesi Veri Yapısı

Alan Adı	**Açıklama**
Hat No	Her hat ve bu hattın her güzergâhı için bir birinden farklı hat numarasıdır.
Sefer No	Hattın sefer numarasıdır.
Durak Sıra No	Durağın hat güzergâhındaki sıra numarasıdır.
Durak No	Durak numarasıdır.
Gün	Seferin planlandığı günleri gösterir.
Saat	Seferin planlandığı saati gösterir.

Veri yapısında bir hattın haftalık zaman çizelgesindeki her hareket saati, ayrı bir sefer numarası ile numaralandırılmıştır. Bir hareket saatinde başlangıç durağından kalkan hattın, o sefer boyunca duraklardan geçiş saatleri ise aynı sefer numarası ile gösterilmiştir.

Gün alanı için ise bit temelli bir veri biçimi kullanılarak bir duraktan geçen aynı gün ve aynı saatteki seferlerin tek satırda ifade edilmesi sağlanmıştır. Böylece hat zaman çizelgesi tablosunun mümkün olan en az satır sayısına sahip olması hedeflenmiştir. Zaman alanları değerlerinin belirlenmesinde bu tabloya sıkça başvurulacağından tablonun az sayıda satır içermesi algoritmanın performansını anlamlı düzeyde artıracaktır. Gün alanının 7 bitten oluşan veri biçiminde sağdan itibaren sırasıyla her bit haftanın bir gününü temsil etmektedir (Tablo 22).

Tablo 22: Gün Alanı Veri Biçimi

Gün bitleri	**Pazar biti**	**Cumartesi biti**	**Cuma biti**	**Perşembe biti**	**Çarşamba biti**	**Salı biti**	**Pazartesi biti**
Onluk sistemdeki değeri	64	32	16	8	4	2	1

Tablo 22'de sunulan veri biçiminde haftanın farklı günlerinde aynı saatte planlanan bir seferin gün değeri, günlerin tabloda verilen onluk sistemdeki değerleri toplamına eşittir. Bu sayede, haftanın her günü aynı saatte planlanan bir seferin 7 ayrı satır yerine tek satırda ifade edilmesi mümkündür. Bu durumda ilgili satırın gün alanının değeri, ikili sayı sisteminde 1111111 değerinin onluk sistemdeki karşılığı olan 127 olacaktır.

Bir çözüm adımının hat hareket zamanı alanı, yukarıdaki veri yapıları kullanılarak şu şekilde belirlenir: İlgili çözüm adımında kullanılan hattın, problemin başlangıç zamanı parametresinin gün bileşenine bağlı olarak hat hareket durağı sıra numarasından geçiş zamanları hat zaman çizelgesi tablosundan bulunur. Bulunan satırlardan, çözüm adımının hat hareket durağına varış zamanından büyük veya eşit olan ilk satırın zaman alanı hat hareket zamanı olarak belirlenir.

Kural 8: Hat Varış Zamanının Hesaplanması

Hat varış zamanı alanı hat hareket zamanı alanına benzer şekilde hesaplanır. Ancak burada hat zaman çizelgesi tablosunda, hattın hat varış durağı sıra numarasından geçiş zamanına bakılırken, hat hareket zamanında kullanılan satır ile aynı sefer numaralı satırlar

sorgulanır. Böyle bir sorgu, hat zaman çizelgesi tablosunda yalnızca bir satırla eşleşecektir. Dolayısıyla eşleşen satırın zaman alanı hat varış zamanı olarak atanır.

Bazı hatlar aynı duraktan iki kez geçebilmektedir. Böyle bir durumda zaman çizelgesinde, hattın aynı sefer numarasıyla bu duraktan geçiş saati için iki satır olacaktır. Çözüm adımlarının hat hareket ve hat varış zamanı alanlarının durak numaraları yerine durak sıra numaraları üzerinden sorgulanma nedeni, hattın aynı seferine ilişkin geçiş saatlerinde durak sıra numaralarının tekil olmasıdır. Bu yaklaşım, çözüm adımının hareket ve varış zamanlarının hesaplanma sürelerini azaltarak algoritmanın toplam performansını olumlu yönde etkilemektedir.

2.3.5. Zaman Parametreli Çözüm Adımlarının Hesaplanma Algoritması Adımları

1. Adım:
Boş bir Çözüm Adımları Listesi (ÇAL) oluştur.

2. Adım: Başlangıç zamanı değişkeninin (BZD) ilk değer ataması
Problemin başlangıç zamanı parametresini BZD'ne ata.

3. Adım: Başlangıç Durakları Listesinin (BDL) BZD Alanı ile Oluşturulması
Başlangıç durağı ve BZD ile BDL'yi oluştur.

4. Adım: Hareket Durakları Listesinin (HDL) Hat Hareket Durağına Varış Zamanı Alanı (HHDVZ), Sefer Numarası Alanı (SN) ve Hat Hareket Zamanı (HHZ) Alanları ile Oluşturulması (Döngü Başlangıcı)
BDL duraklarından ve bu duraklara yürüme mesafesindeki duraklardan geçen hatların ilgili durak satırlarını Hat Güzergâh tablosundan seçerek HDL'yi oluştur.
HDL'nin HHDVZ alanını yürüme mesafesine bağlı olarak Kural 6'ya göre ata.

5. Adım: HDL'nin SN ve HHZ Alanının Hesaplanması
HDL'nin her satırı için Hat Zaman Çizelgesi tablosundan SN ve HHZ alanlarını Kural 7'ye göre belirle.

6. Adım: Varış (Aktarma) Durak Listesinin (VDL) oluşturulması

HDL'deki her satır için aynı hat üzerinde bulanan ve Durak Sıra Numarası alanı ilgili HDL satırındaki durak sıra numarası alanından büyük olan ve yine aynı hat üzerindeki AS etiket değeri küçük olan durakları eşleştirerek VDL'yi oluştur.

7. Adım: ÇAL'ın güncellenmesi

HDL ve VDL listeleri üzerinden hat numarası aynı olan satırların kartezyen çarpımları ile çözüm adımlarını oluşturarak ÇAL'a ekle.

8. Adım: ÇAL'ın Hat Varış Zamanı Alanının (HVZ) Hesaplanması

Listenin HVZ alanını, Kural 8'e göre belirle.

9. Adım: BDL'nin güncellenmesi

ÇAL'ın Hat Varış Durağı (HVD) ve HVZ alanları ile satırları tekrar etmeyen yeni BDL oluştur.

10. Adım: Durdurma Kriteri ve Döngü Sonu

Varış durağı BDL'de varsa döngüyü sonlandır. Aksi halde 3. Adıma dön.

2.3.6. Zaman Parametreli Problemde Toplam Yolculuk Çözümlerinin Çözüm Adımlarından Hesaplanması

Kural 9: Zaman Parametreli Çözüm Adımlarının Eşleştirilmesi

Zaman parametreli çözüm adımlarından toplam yolculuk çözümleri oluşturulurken Kural 4'den farklı olarak adım seçeneklerinin hat varış-yolculuk başlangıç durak çiftlerinin aynı olmasının yanında önceki adım hat varış zamanı alanının sonraki adım hat hareket durağına varış zamanı alanından küçük veya eşit olması gerekir.

2.3.7. Zaman Parametreli Problemde Aktarım Hatlarının Nihai Varış Durağına Ulaşma Garantisinin (Teorem 3) Geçerliliği

Zaman parametreli problemde Teorem 2, geçerliliğini sürdürmektedir. Ancak, Teorem 3 için durum biraz farklıdır. Teoremin geçerliliğini kaybettiğini söylemek

mümkün olmasa da zaman parametreli problemde bazı durumlarda, üretilen her çözüm adımından nihai varış durağına ulaşmak mümkün olmayabilir.

Bir çözüm adımının, bir sonraki adımda kullanılacak olan aktarım hattının hat hareket durağına, hattın ilgili duraktan son geçiş zamanından sonra ulaşması durumunda bu çözüm adımının hat hareket saati hesaplanamayacaktır. Çünkü Kural 7'de ilgili hattın o gün planlanan hareket saatlerine bakılmaktadır. Dolayısı ile Teorem 3 teorik olarak geçerliliğini sürdürse de makul olmayan çözümlerin üretilmesini engelleyecek şekilde tasarlanan Kural 7, uygulamada bu teoremin geçerliliğini ortadan kaldırmaktadır. Teorem 3'ün uygulamadaki geçerliliğinin nasıl artırılacağı ileride tartışılacaktır.

2.5. Yüksek Kaliteli Çözümlerin Seçilmesi: Optimizasyon

Toplu taşıma ağlarında, iki nokta arasındaki yolculuk çözümlerinin kalitesi birçok parametreye bağlı olarak hesaplanabilir. Bununla birlikte bir çözümün kalitesinin, onun kullanıcılar tarafından tercih edilirliğinin güçlü bir göstergesi olması gerekir. Bir çözümün kullanıcılar tarafından tercih edilirliğini etkileyen parametrelerden bazıları şunlardır; Çözümün toplam parasal maliyeti, toplam seyahat süresi, toplam yürüme mesafesi, bir durakta en fazla bekleme süresi, varış zamanı, aktarım sayısı, yolculukta kullanılacak ulaşım modları.

Çözümün tercih edilirliğini etkileyen başka parametreler de saymak şüphesiz mümkündür. Örneğin yolculukta kullanılan araçların ve bekleme duraklarının fiziki özellikleri, yolculukta kullanılan hatların güzergâhları (kıyı şeridi ya da trafik yoğunluğu düşük olan güzergâhlar v.b.) gibi parametreler de çözümün tercih edilirliğini etkileyecektir. Ancak bu tür parametrelere ait verilerin elde edilmesi oldukça zor ve zahmetlidir.

Bir çözümün kalitesi hesaplanırken, yukarıda sayılan parametrelerin çözüm kalitesini ne şekilde etkileyeceği de bilinmelidir. Ayrıca her parametrenin, kullanıcılar nezdinde farklı bir önem seviyesine sahip olabileceği de açıktır. Bu çalışmada geliştirilen algoritmanın olası tüm çözümleri hesaplayabilme yeteneği, çeşitli amaç fonksiyonları üzerinde optimizasyon gerçekleştirme imkanı vermektedir. Ancak burada çözüm kalitesi, hesaplama kolaylığı nedeniyle yolcuğun tamamlanma zamanı olarak ele alınmış ve optimizasyon için gerekli olan amaç fonksiyonu bu parametreye bağlı olarak

oluşturulmuştur. En iyi çözüm için amaç fonksiyonu Eşitlik (2.6) ile verilmektedir. Bu çalışmada K en kısa yol algoritmasında olduğu gibi en iyi k adet çözümün elde edilmesi amaçlanmaktadır. Bu nedenle en iyi k adet çözüm için amaç fonksiyonu da Eşitlik (2.7) ile verilmektedir.

En İyi Çözüm için Amaç Fonksiyonu:

$f = Ç_j \mid Min\{T_i, i: 1..n\} = T_j$ (2.6)

Burada;

$Ç_i$: i'yinci çözüm

T_i : i'yinci çözümün varış zamanı

n: Toplam çözüm sayısıdır.

En İyi k Çözüm için Amaç Fonksiyonu:

$$F = \begin{cases} \emptyset, m = 0 \\ f_m \mid Ç_j \mid Min(\{T_i, i: 1..n\} \backslash \{f_0..f_{m-1}\}) = T_j, m: 1..k \end{cases} \quad (2.7)$$

2.6. Çözüm Uzayının Daraltılması: Dinamik Programlama ve Sezgisel Yaklaşım

Çözüm uzayının çok büyük olduğu optimizasyon problemlerinde, mümkün olan tüm çözümleri hesaplayıp bunlar arasından en iyi çözümü seçmek, işlem maliyetinin yüksekliği sebebiyle tercih edilen bir yaklaşım değildir. Bunun yerine optimizasyon problemi alt adımlara bölünerek, her adımda optimum çözüme ulaşabilecek olan seçenekler belirlenip, problemin çözüm uzayı daraltılır. Bu yöntem literatürde dinamik programlama olarak adlandırılmaktadır.

Toplam çözümün, çözüm seçeneklerinin kombinasyonlarından oluştuğu kombinatoryal problemlerde dinamik programlama teknikleri, işlem maliyetini önemli

ölçüde azaltarak, optimizasyon algoritmasının çok daha hızlı bir şekilde çözüme ulaşmasına yardımcı olmaktadır.

Çözüm uzayının çok büyük olduğu ve optimum çözümün neye benzeyeceği hakkında önceden bilgi sahibi olmadığımız optimizasyon problemlerinde, optimum çözüme hızlı bir şekilde yaklaşan, ancak optimum çözümü garanti etmeyip optimuma yakın çözümler üreten diğer bir yaklaşım da sezgisel yaklaşımdır. Belli bir probleme yönelik olmayıp farklı optimizasyon problemlerini çözebilecek şekilde genelleştirilmiş sezgisel yaklaşımlar, literatürde meta-sezgisel yöntemler olarak anılmaktadır. Genetik algoritma, karınca kolonisi algoritması, tavlama benzetimi algoritmaları, bu kategorideki algoritmalardan bazılarıdır. Ayrıca bu tür optimizasyon problemlerinde, çözüm algoritmasını tamamen bir sezgisel yönteme dayandırmadan, optimizasyonu hızlandıracak sezgisel fonksiyon ya da kurallar oluşturmak suretiyle kesin çözümü garanti eden yaklaşımlar da kullanılmaktadır. En kısa yol problemi için geliştirilen A* ve ALT algoritmaları bu yaklaşım için örnek gösterilebilir.

Toplu taşıma ağlarında yolculuk planlama problemine dair çözüm uzayının, yolculukta kullanılacak aktarım sayısına bağlı olarak geometrik bir şekilde arttığı bilinmektedir. Probleme zaman parametresinin eklenmesi ile birlikte, bir çözüm adımında, durak alanları ve hat alanı aynı olan çözüm seçeneklerinin zamanlama farkından doğan türevleri, çözüm adımındaki seçenek sayısını artıracağı için çözüm uzayı çok daha büyük boyutlara ulaşacaktır.

Bu çalışmada toplu taşıma ağlarında iki düğüm arasındaki yolculuk çözümleri araştırılırken problem, en az aktarımla seyahat için uygun düğümlerin etiketlenmesi ve çözüm adımlarının ve bu adımlardan toplam yolculuk çözümlerinin hesaplanması alt problemlerine ayrılarak, çözüm için dinamik programlamaya benzer bir yaklaşım benimsenmiştir. Bununla birlikte bu işlem adımları henüz bir optimizasyon problemi içermediği için kullanılan yaklaşımın tam anlamıyla bir dinamik programlama yöntemi olduğu söylenemez. Problemin çözümünde dinamik programlama tekniğinin literatürdeki tanımına daha uygun olarak kullanıldığı asıl işlem adımı ise optimizasyon adımıdır.

Bu çalışmada, problemin çözüm uzayından en iyi k çözümün elde edilmesini amaçlayan optimizasyon işlemi, farklı bir adım olarak düşünülmeyip çözüm adımlarının hesaplanma işlemi içerisinde uygulanmaktadır. Bu bağlamda problemin çözüm adımı seçenekleri hesaplanırken sezgisel bir kurala bağlı olarak, optimum çözümler arasında yer alamayacağı belirlenebilen seçenekler çözüm dışı bırakılmaktadır. Aşağıda bu kural ve elemeler açıklanmaktadır.

Teorem 5: Bir başlangıç durağından bir varış durağına seyahat ederken kullanılabilecek tüm hatların ve bu hatlardan varış durağına en erken ulaşan hattın tüm yolcular tarafından bilindiği varsayımı altında toplu ulaşım ağında aynı iki durak arasında seyahat edecek yolculardan başlangıç durağına sonradan gelen yolcuların, varış noktasına, başlangıç durağına önce gelen yolculardan daha erken ulaşma şansı yoktur.

t_{iA}: i'yinci yolcunun başlangıç durağına geliş zamanı

t_{iB}: i'yinci yolcunun varış durağına geliş zamanı

P: olasılık fonksiyonu

olmak üzere;

$$t_{iA}<t_{jA} \Rightarrow P(t_{iB}<t_{jB}) = 0 \qquad (2.4)$$

İspat*: Başlangıç durağına daha erken gelen yolcu, sonradan gelen yolcunun kullanabileceği tüm hatları kullanabilecektir. Bu sebeple sonradan gelen yolcunun, erken gelen yolcunun sahip olduğu ulaşım seçeneklerinden daha iyi bir seçeneğe sahip olma şansı olamaz.*

Teorem 5'in dayandığı varsayım, gerçekte doğru olmaktan uzak olsa da çözüm algoritması işletilirken iki durak arasında kullanılabilecek tüm seçenekler hesaplanabildiğinden, optimizasyon işleminde bu varsayımın geçerli olacağı açıktır. Çünkü bu noktada varsayımın doğruluğu, yolculuğu gerçekleştirecek insanlar açısından değil yolculuğu planlayan algoritma açısından değerlendirilmelidir.

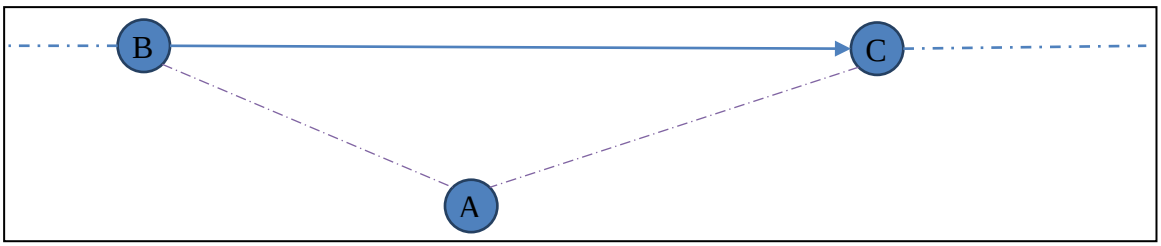
Kural 10: Çözüm Adımı Seçeneklerinin Elenmesi

Teorem 5'e dayanarak, bir adımdaki çözüm seçeneklerinin hat varış durağı aynı olanlar arasından, hat varış zamanı diğerlerinden büyük olanların nihai varış durağına daha önce ulaşma şansları yoktur. Ancak bu bilgi çözüm adımı seçeneklerinin elenmesi için yeterli değildir. Çünkü varış durağına geç ulaşan çözüm adımı seçeneklerinin, erken ulaşan seçeneklerle aynı zamanda nihai varış durağına ulaşma şansları devam etmektedir. Bu husus daha sonra geniş bir şekilde tartışılacaktır. Şimdilik aynı varış durağına daha geç ulaşan çözüm adımı seçeneklerinin elenmesi yoluna gidilerek algoritmanın hızlandırılması sağlanacaktır.

Kural 11: Aynı Hattı Kullanan Çözüm Adımı Seçeneklerinden Sefer Numarası Küçük Olanın Tercih Edilmesi

Aynı çözüm adımındaki seçenekler arasında, hat no, yolculuk başlangıç durağı ve hat varış durağı aynı olan ancak hat hareket durakları farklı olan birden çok seçenek olabilir. Bu durum, bir önceki çözüm adımının varış durağına bağlı olarak, ilgili hatta birkaç farklı duraktan binilebileceğinin bir göstergesidir. Bu noktada, sonraki adım hattının birden fazla durağına yürüme mesafesinde olan bir duraktan, hattın hangi durağına yürüneceğine karar verilecektir. Akla ilk gelen çözüm elbette yürüme mesafesi küçük olan duraktır. Ancak karar vermeden önce bir ayrıntıya dikkat etmek gerekir. Söz konusu ayrıntı Şekil 42 yardımıyla aşağıda açıklanmaktadır.

Şekil 41: Aktarım Durağının Seçimi



Şekil 41'de B ve C duraklarından geçen bir hatta yürüme mesafesinde olan A durağı gösterilmiştir. A durağının B durağına daha yakın olmasına rağmen, A durağındaki bir yolcunun B durağına çok yaklaşmış olan bir hat vasıtasını, C durağında yakalama şansı B durağına göre daha fazladır.

Yukarıdaki senaryonun gerçekleşme ihtimalinin çok düşük olacağı düşünülse de algoritmanın bu tür istisnai durumlarda da doğru karar vermesini sağlayacak kuralın ek bir işlem maliyeti olmayacaktır. Çünkü bu durumda algoritmanın doğru karar verebilmesi için kullanılacak kural sefer numarası küçük olan çözüm adımının seçilmesi olacaktır.

Kural 12: Aynı Hattı Kullanan ve Sefer Numaraları Eşit Olan Çözüm Adımı Seçeneklerinden Yürüme Mesafesi Küçük Olanın Tercih Edilmesi

Kural 11'e ek olarak çözüm adımlarının sefer numaralarının aynı olması durumda yürüme mesafesi küçük olan seçenek tercih edilmelidir. Bu kural optimizasyonda kullanılan amaç fonksiyonunun değerini etkilememekle birlikte genel eğilimler açısından daha tercih edilebilir çözümler üretilmesini sağlar.

2.7. Zaman Parametreli Çözüm Adımlarının Hesaplanma Algoritmasına Sezgisel Kuralların Uygulanması

1. Adım:

Boş bir Çözüm Adımları Listesi (ÇAL) oluştur.

2. Adım: Başlangıç zamanı değişkeninin (BZD) ilk değer ataması

Problemin başlangıç zamanı parametresini BZD'ne ata.

3. Adım: Başlangıç Durakları Listesinin (BDL) BZD Alanı ile Oluşturulması

Başlangıç durağı ve BZD ile BDL'ni oluştur.

4. Adım: Hareket Durakları Listesinin (HDL) Hat Hareket Durağına Varış Zamanı Alanı (HHDVZ), Sefer Numarası Alanı (SN) ve Hat Hareket Zamanı (HHZ) Alanları ile Oluşturulması (Döngü Başlangıcı)

BDL duraklarından ve bu duraklara yürüme mesafesindeki duraklardan geçen hatların ilgili durak satırlarını Hat Güzergâh tablosundan seçerek HDL'yi oluştur.

HDL'nin HHDVZ alanını yürüme mesafesine bağlı olarak Kural 6'ya göre ata.

5. Adım: HDL'nin SN ve HHZ Alanının Hesaplanması

HDL'nin her satırı için Hat Zaman Çizelgesi tablosundan SN ve HHZ alanlarını Kural 7'ye göre belirle.

6. Adım: Varış (Aktarma) Durak Listesinin (VDL) oluşturulması

HDL'deki her satır için aynı hat üzerinde bulanan ve Durak Sıra Numarası alanı ilgili HDL satırındaki durak sıra numarası alanından büyük olan ve yine aynı hat üzerindeki AS etiket değeri küçük olan durakları eşleştirerek VDL'yi oluştur.

7. Adım: ÇAL'ın güncellenmesi

HDL ve VDL listeleri üzerinden hat numarası aynı olan satırların kartezyen çarpımları ile çözüm adımlarını oluşturarak Kural 10, 11 ve 12'de elenmeyen seçenekleri ÇAL'a ekle.

8. Adım: ÇAL'ın Hat Varış Zamanı Alanının (HVZ) Hesaplanması

Listenin HVZ alanını, Kural 8'e göre belirle.

9. Adım: BDL'nin güncellenmesi

ÇAL'ın Hat Varış Durağı (HVD) ve HVZ alanları ile satırları tekrar etmeyen yeni BDL oluştur.

10. Adım: Durdurma Kriteri ve Döngü Sonu

Varış durağı BDL'de varsa döngüyü sonlandır. Aksi halde 3. Adıma dön.

2.8. Probleme Ek Kısıtların Eklenmesi

Toplu taşıma ağlarında iki nokta arasındaki bir çözümün kalitesini etkileyecek parametrelerden ve bu parametrelerin önceliklerinin kullanıcıdan kullanıcıya değişiklik gösterebileceğinden daha önce bahsedilmişti. Bu bağlamda problemin optimum çözümlerinin, kullanıcıların belirleyeceği önceliklere göre hesaplanabileceği çalışmanın tartışma bölümünde ele alınacaktır. Bu aşamada ise probleme kullanıcı tarafından belirlenecek kısıtların eklenmesi suretiyle çözümlerin kısmen de olsa kullanıcı tercihleri göz önüne alınarak üretilmesi amaçlanmaktadır. Bu çerçevede probleme üç farklı türde toplam beş kısıt eklenmiş çözümlerin bu kısıtlara uygun olarak nasıl hesaplanacağı her kısıt için ayrı ayrı ele alınmıştır. Kısıtlar aşağıdaki gibidir.

1. Yolculukta kullanılacak ulaşım modları kısıtı
2. Yürüme mesafesi kısıtı
    a. Bir adımda maksimum yürüme mesafesi
    b. Toplam maksimum yürüme mesafesi
3. Bekleme zamanı kısıtı
    a. Bir adımda maksimum bekleme süresi
    b. Toplam maksimum bekleme süresi

2.8.1. Ulaşım Modları Kısıtı

Ulaşım modları kısıtı ile yolcunun, toplam yolculuk çözümünde hangi ulaşım modlarının yer alabileceğine karar verebilmesi amaçlanmıştır. Çözümlerin yalnızca kullanıcının tercih ettiği ulaşım modlarına ait hatlardan oluşmasını sağlamak için çözüm adımlarının sadece bu hat modlarından seçilmesi yeterli olacaktır.

Çözüm adımlarının ulaşım modları kısıtına uygun olarak seçilmesi için bu kısıtın ters akım algoritmasında uygulanması en doğru yaklaşımdır. Çünkü ters akım algoritmasının işaretlemediği bir hattın çözüm adımlarında kullanılması mümkün değildir. Bu işlem için ters akım algoritmasının VIGH listesinin oluşturulduğu 3. adımını güncellemek yeterli olacaktır. Güncellenmiş algoritmada VIGH listesi yalnızca ulaşım modları kısıtında izin verilen hatlardan oluşacak şekilde seçilmelidir.

2.8.2. Yürüme Mesafesi Kısıtları

Yürüme mesafesi kıstı probleme, adım veya toplam yolculuk çözümleri temelinde olmak üzere iki farklı sevide uygulanabilir.

Bir adımdaki maksimum yürüme mesafesi kısıtının, ulaşım modları kısıtında olduğu gibi ters akım algoritmasında uygulanması uygun olacaktır.

Toplam maksimum yürüme mesafesi kısıtının ise yalnızca çözüm adımlarının eşleştirilmesi aşamasında uygulanması mümkündür. Toplam yolculuk çözümleri bu aşamada oluştuğu için bir çözümün bu kısıta uyup uymadığına ancak bu aşamada karar verilebilir.

2.8.3. Bekleme Süresi Kısıtları

Bekleme süresi kıstı da yürüme mesafesi kısıtında olduğu gibi adım veya toplam yolculuk çözümleri temelinde olmak üzere iki farklı sevide ele alınabilir.

Bir adımdaki maksimum bekleme süresi kısıtı aynı seviyedeki yürüme mesafesi kısıtından farklı olarak çözüm adımlarının hesaplanması aşamasında uygulanabilir. Çünkü bir çözüm adımı seçeneğinin zaman alanları bu aşamada hesaplanmaktadır. Bu işlem için algoritmanın hat hareket zamanının hesaplandığı 5. adımında hat hareket durağına varış zamanı ile hat hareket zamanı alanları arasındaki farkın, bir adımdaki maksimum bekleme süresi kısıtından büyük olduğu satırlar hareket durakları listesinden çıkartılır.

Toplam maksimum bekleme süresi kısıtı ise aynı seviyedeki yürüme mesafesi kısıtında olduğu gibi çözüm adımlarının eşleştirilmesi aşamasında uygulanacaktır.

2.8.4. Ulaşım Modları ve Bir Adımda Maksimum Yürüme Mesafesi Kısıtlarına Göre Güncellenmiş Ters Akım Algoritmasının İşleyiş Adımları

1. Adım:

Genel Aktarım Sayısı (GAS) değişkenini 1 ata.

2. Adım:

Varış durağını ve bu durağa yürüme mesafesi bir adımda maksimum yürüme mesafesi kısıtından küçük veya eşit olan durakları Varış Durakları (VD) listesine ekle

3. Adım: Döngü Başlangıcı

VD listesindeki duraklardan geçen hatları (VDGH) ulaşım modları kısıtına göre listele.

4. Adım: Etiketleme

VDGH listesindeki tüm hatların VD listesindeki duraklarını (durak hattın ilk durağı değilse) Transfer Noktası (TN etiketi) olarak etiketle. Aynı durakların Aktarım Sayısı (AS etiketi) etiketini GAS olarak ata.

5. Adım: Etiketleme

Adım 3'de etiketlenen durakların durak numarasından küçük olan etiketlenmemiş durakların AS etiketini GAS + 1 olarak ata.

6. Adım:

VI ve VIGH listelerini temizle.

7. Adım:

AS etiketi GAS + 1 olan durakları ve bu durağa yürüme mesafesi bir adımda maksimum yürüme mesafesi kısıtından küçük veya eşit olan durakları VI listesine ekle.

8. Adım:

GAS değişkenini bir artır.

9. Adım:

Başlangıç durağı AS etiketi atanmışsa işlemi sonlandır. Aksi durumda 3. Adıma geri dön.

Ulaşım modları ve bir adımdaki maksimum yürüme mesafesi kısıtlarının ters akım algoritmasında uygulanmasındaki temel neden, çözüm aranan iki durak arasındaki mümkün olan en az aktarımlı çözümlerin hiç birinin problem kısıtlarına uymaması halinde, toplam çözüm yaklaşımının diğer aşamalarının çözüm üretemeyecek olmasıdır. Çünkü ters akım algoritması iki durak arasındaki en az aktarımı mümkün kılan durakları etiketlemek üzere kurgulanmıştır.

Ayrıca bu yaklaşım, algoritmanın, problem kısıtına uymayan durakları etiketlemesini engelleyerek toplam çözüm yaklaşımının sonraki aşamalarını hızlandırmaktadır. Etiketlenen durak sayısının performansa etkisi daha önce tartışılmıştı.

2.8.5. Zaman Parametreli Çözüm Adımlarının Hesaplanma Algoritmasına Bir Adımda Maksimum Bekleme Süresi Kısıtı ile Sezgisel Kuralların Uygulanması

1. Adım:

Boş bir Çözüm Adımları Listesi (ÇAL) oluştur.

2. Adım: Başlangıç zamanı değişkeninin (BZD) ilk değer ataması

Problemin başlangıç zamanı parametresini BZD'ye ata.

3. Adım: Başlangıç Durakları Listesinin (BDL) BZD Alanı ile Oluşturulması

Başlangıç durağı ve BZD ile BDL'yi oluştur.

4. Adım: Hareket Durakları Listesinin (HDL) Hat Hareket Durağına Varış Zamanı Alanı (HHDVZ), Sefer Numarası Alanı (SN) ve Hat Hareket Zamanı (HHZ) Alanları ile Oluşturulması (Döngü Başlangıcı)

BDL duraklarından ve bu duraklara yürüme mesafesindeki duraklardan geçen hatların ilgili durak satırlarını Hat Güzergâh tablosundan seçerek HDL'yi oluştur.
HDL'nin HHDVZ alanını yürüme mesafesine bağlı olarak Kural 6'ya göre ata.

5. Adım: HDL'nin SN ve HHZ Alanının Hesaplanması

HDL'nin her satırı için Hat Zaman Çizelgesi tablosundan SN ve HHZ alanlarını Kural 7'ye göre belirle.HDL'nin HHDVZ ve HHZ alanları farkı, bir adımda maksimum bekleme süresi kısıtından büyük olan satırlarını listeden çıkart.

6. Adım: Varış (Aktarma) Durak Listesinin (VDL) oluşturulması

HDL'deki her satır için aynı hat üzerinde bulanan ve Durak Sıra Numarası alanı ilgili HDL satırındaki durak sıra numarası alanından büyük olan ve yine aynı hat üzerindeki AS etiket değeri küçük olan durakları eşleştirerek VDL'yi oluştur.

7. Adım: ÇAL'ın güncellenmesi

HDL ve VDL listeleri üzerinden hat numarası aynı olan satırların kartezyen çarpımları ile çözüm adımlarını oluşturarak Kural 10, 11 ve 12'de elenmeyen seçenekleri ÇAL'a ekle.

8. Adım: ÇAL'ın Hat Varış Zamanı Alanının (HVZ) Hesaplanması

Listenin HVZ alanını, Kural 8'e göre belirle.

9. Adım: BDL'nin güncellenmesi

ÇAL'ın Hat Varış Durağı (HVD) ve HVZ alanları ile satırları tekrar etmeyen yeni BDL oluştur.

10. Adım: Durdurma Kriteri ve Döngü Sonu

Varış durağı BDL'de varsa döngüyü sonlandır. Aksi durumda 3. Adıma dön.

2.8.6. Aktarım Hatlarının Nihai Varış Durağına Ulaşma Garantisi Teoreminin Yeniden Ele Alınması

Zaman parametreli problemde, Teorem 3 teorik olarak geçerliliğini sürdürmesine rağmen, hat hareket zamanının belirlenmesi kuralına (Kural 7) bağlı olarak pratikte bazı durumlarda geçerliliğini kaybetmektedir. Probleme, bir adımdaki maksimum bekleme süresi kısıtının eklenmesiyle birlikte bu tür durumların ortaya çıkma sıklığının artabileceği açıktır.

Teorem 3'ün geçerliliğini kaybetmesi, iki durak arasında daha fazla aktarımla seyahat etmek mümkün iken, ters akım algoritmasının tespit ettiği aktarım sayısına bağlı olarak, toplam çözüm yaklaşımının çözüm üretememesine neden olabilir.

En az aktarım yaklaşımının uygun olmadığı durumlar ele alınırken, zaman parametreli problemde sefer sıklığı az olan hatların bu tür sorunlara sebep olabileceğine değinilmişti. Benzer şekilde bir hattın günün son seferini tamamladıktan sonraki bir zamanda aktarım hattı olarak seçilmesi durumunda da toplam çözüm yaklaşımının çözüm üretemeyebileceği ifade edilmişti.

Bu aşamada zaman parametreli problemin çözüm üretemeyebileceği durumları azaltabilecek yaklaşımlar üzerinde durulacaktır. Bu yaklaşımlardan biri hiç şüphesiz daha önce de tavsiye edildiği gibi ters akım algoritmasının mümkün olan en az adımdan bir fazla aktarımlı çözüm seçeneklerini etiketleyecek şekilde uygulanmasıdır.

Hattın zaman çizelgesine bağlı olarak çözüm üretilememesi sorununu engellemek için ise temel yaklaşım, zaman çizelgesi sorgulanırken sorgu kriterinin, problemin başlangıç zamanı parametresinin gün bileşeninden bir gün sorasındaki hareket saatlerini de kapsayacak şekilde düzenlenmesidir. Bu yaklaşım özellikle gece aktarımlarında karşılaşılması daha muhtemel olan çözüm üretilememe sorununu önemli ölçüde ortadan kaldıracaktır.

Toplu taşıma hattı seferlerinin zaman çizelgeleri incelendiğinde, günlük planlamaların 00:00:23:59 periyodunda değil, genellikle 05:00-04:59 periyodunda yapıldığı görülmektedir. Buradan hareketle bir önceki yaklaşıma alternatif olarak, 05:00-04:59 periyodu dikkate alınarak hatların duraklardan geçiş saatleri, hat zaman çizelgesi veri yapısına işlenebilir. Bu işlem için zaman alanı 00:00-04:59 saatleri arasında olan satırların gün alanını bir önceki günü gösterecek şekilde güncellemek yeterli olacaktır. Algoritma geliştirme sürecinde iki yaklaşım da test edilmiş ve ikinci yaklaşımın performansının birinciye göre daha yüksek olduğu görülmüştür.

ÜÇÜNCÜ BÖLÜM

3. TUR ALGORİTMASININ KARŞILAŞTIRMALI PERFORMANS DEĞERLERNDİRMESİ

Son yıllarda TUA üzerinde yolculuk planlaması yapan uygulamalar hızla artmakta ve yaygınlaşmaktadır. Dünya genelinde yaklaşık 7-8 yıllık bir geçmişe sahip olan bu uygulamalar ülkemizde ise yeni yeni gelişmektedir. Genellikle internet üzerinden web ve mobil (akıllı cihaz) tabanlı çalışan uygulamalar, Amerika, Avrupa ve Avustralya kıtalarında yoğunlaşırken Türkiye'de sadece birkaç büyükşehir belediyesinin çalışmalarıyla sınırlı kalmıştır. Özellikle büyük teknoloji firmaları tarafından desteklenen, akademik çalışmaların sayısı ise uluslararası ölçekte hızla artarken, ülkemizde bu alanda yapılan akademik çalışma yok denecek kadar azdır.

TUA üzerinde yolculuk planlaması problemine yönelik yapılan çoğu akademik çalşıma, probleme çözüm yöntemleri önerirken, yöntemin sadece performansını göz önünde bulundurmuştur. Çözümlerin kalitesi ve gecikmelere bağlı uygulanabilirlik düzeyleri ise çok az çalışmada dikkate alınmıştır.

Bu bölümde TUR algoritması Visual Studio 2013 C# dilinde kodlanarak bir masaüstü uygulama geliştirilmiş ve uygulama Bursa, İzmir, Ankara, İstanbul ve Londra TUA'ları verileri ile koşturulmuştur. Elde edilen sonuçlar mevcut uygulama ve algoritmalarla karşılaştırılmıştır. Alandaki ulusal uygulamaların azlığı ve bu uygulamaların arka planlarında kullanılan algoritmaların belirsizliği sebebiyle, çalışmada önerilen yaklaşımın mevcut sistem ve yöntemlerle karşılaştırması iki aşama halinde verilmektedir. Birinci aşamada ulusal uygulamaların bir TUA için ürettiği çözümler önerilen yaklaşım çözümleriyle karşılaştırılmış ve bu aşamada performans karşılaştırılması yapılmamıştır. Birinci aşama karşılaştırmalarının amacı, bu çalışmada önerilen TUR algoritmasının ürettiği çözümler ile mevcut uygulama çözümlerinin çözüm kalitesi açısından karşılaştırmalı değerlendirmesini yapabilmektir. Böylece TUR algoritmasının ürettiği çözümlerin uygun çözümler olup olmadığının değerlendirilebilmesi mümkün olacaktır.

İkinci aşamada ise literatürde geliştirilen yaklaşımların performansları, TUR algoritmasının aynı TUA üzerinde benzer özellikli bilgisayar kullanılarak elde edilen performans değerleri ile karşılaştırılmıştır. Sonuç olarak TUR algoritmasının hem çözüm kalitesi açısından hem uygun çözümlerin hesaplanma süresi açısından performansı iki aşamalı olarak ortaya konulmuştur.

3.1. TUR Algoritması Çözümlerinin Mevcut Ulusal Sistemlerin Çözümleri ile Karşılaştırılması

Karşılaştırma çalışmalarının birinci aşamasında TUA verilerine erişilebilen Bursa, İzmir, Ankara ve İstanbul şehirleri seçilmiş, şehirlerin TUA verileri çeşitli sitelerden derlenerek bir veri tabanında birleştirilmiştir. Ardından TUR algoritması için geliştirilen uygulama çözümleri Trafi uygulaması çözümleri ile karşılaştırılmıştır.

Karşılaştırma için seçilen illerin TUA verilerine ilişkin bilgiler Tablo 22'de verilmiştir. Tablo 22'deki "Yürüme Mesafeleri" satırı TUA'daki birbirine yürüme mesafesinde bulunan durak çiftlerinin sayısını göstermektedir. Trafi uygulaması en fazla yürüme mesafesini 1000m ile sınırlarken TUR uygulamasında yürüme mesafesi 500m ile sınırlandırılmıştır. Yürüme mesafesi sınırı arttıkça birbirlerine yürüme mesafesinde bulanan durak çiftleri sayısı artacağından, iki durak arasında kullanılabilir hat sayısı da artacak ve buna paralel olarak problemin çözüm uzayı genişleyecektir. Böylece, daha önce ulaşılamayan duraklar ve hatların çözüme dâhil edilebilmesi ve çözüm kalitesinin yükseltilmesi mümkün olacaktır. Başka bir deyişle, yüksek yürüme mesafesi sınırının, hesaplama süresini artırırken, çözüm kalitesini yükseltmesi beklenir. TUR uygulamasında yürüme mesafesi sınırının düşük alınmasının sebebi, Trafi uygulamasının optimizasyon esnasında yürüme mesafelerini nasıl hesapladığının ve yürüme mesafesinde bulunan durak çiftlerinin sayısının bilinmemesidir. TUR uygulamasında yürüme mesafesi sınırı düşük tutularak, uygulamanın, dezavantajlı durum da yüksek kaliteli çözümler üretebildiğinin gösterilmesi amaçlanmaktadır.

TUA'larda aynı hattın ait farklı güzergâhları olabilmektedir. Bu güzergâhlar hattın gidiş-dönüş yönleri olabileceği gibi hattın farklı zamanlarda kullanılan ve ana güzergâhtan farklı duraklara uğrayan bir başka güzergâhı da olabilir. Tablo 22'nin "Güzergâh Sayısı"

satırı TUA'daki toplam güzergâh sayısını; "Güzergâh Durakları Sayısı" da bu güzergâhların geçtiği durakların sayıları toplamını göstermektedir. TUA yolculuk planlaması için kullanılan optimizasyon algoritmaları genellikle güzergah durakları ya da zaman çizelgeleri üzerinden işlem yaptıkları için performans değerlendirmesinde bu sayılar özellikle göz önünde bulundurulmaktadır.

Tablo 22'deki "Hareket Saati Sayısı" olarak verilen satır ise TUA'nın zaman çizelgesindeki satır sayısını göstermektedir. Bu sayı, hatların tüm güzergâh duraklarından Pazartesi gününe ait geçiş saati sayıları toplamıdır. Çünkü çoğu akademik çalışmada performans ölçümleri TUA'nın bir günlük zaman çizelgesi verileriyle yapılmaktadır.

Tablo 22: TUA İstatistiki Verileri

	Bursa	İzmir	Ankara	İstanbul
Durak Sayısı	4124	7671	7264	15988
Yürüme Mesafeleri	62416	77656	76278	245428
Hat Sayısı	214	329	424	1033
Güzergâh Sayısı	404	669	424	1514
Güzergâh Durakları Sayısı	16463	19916	29115	57719
Hareket Saati Sayısı	431165	692101	802465	3266363

Karşılaştırma işlemleri için kullanılacak durak çiftleri, her bir TUA için rastlantısal olarak seçilmiştir. Hareket saati parametresi de yine rastlantısal olarak 8-18 saatleri aralığında belirlenmiştir. Seçilen durak çiftleri aynı gün ve hareket saati parametresi ile TUR ve Trafi uygulamalarında ayrı ayrı sorgulanmış ve uygulamaların varış zamanı kriterine göre ürettiği optimal çözümler, tablolar halinde verilmiştir. Ayrıca çözümlerin görsel olarak da karşılaştırılabilmesi için harita çizimleri kullanılmıştır.

Sırasıyla Bursa, İzmir, Ankara ve İstanbul TUA'ları için seçilen durak çiftleri sırasıyla Tablo 23, 25, 27 ve 29'da gösterilmiştir. Karşılaştırma kolaylığı açısından her ilin sonuçları ilgili il için seçilen durak çiftleri tablosunun hemen arkasından tablolar halinde verilmiştir (Tablo 24, 26, 28, 30). Çözüm karşılaştırma tabloları üç ana sütundan oluşmaktadır. İlk sütun durak çifti numarasını; ikinci ve üçüncü sütunlar ise sırasıyla Trafi ve TUR uygulamalarının sonuçlarını göstermektedir. Her bir uygulama sütunu ise 4 alt

sütundan oluşmaktadır. Çözüm sütunu seçilen durak çiftleri arasında ilgili uygulamanın ürettiği optimal çözümün kullandığı hatları sıralı olarak göstermektedir. Varış zamanı sütunu ise seçilen durak çifti için hesaplanan optimal çözümün hedef durağa varış zamanıdır. Ayrıca üretilen çözümde aktarım durakları arasındaki maksimum mesafe (yürüme mesafesi) ve tüm çözüm boyunca toplam yürüme mesafeleri de uygulamalar tarafından üretilen çözümlerin kalitelerinin bir göstergesi olarak, sırasıyla 3. ve 4. alt sütunlarda belirtilmiştir. Buna ilaveten iki ayrı uygulama tarafından aynı durak çifti için üretilen çözümlerin harita çizimleri de yan yana gelecek şekilde sunulmuştur (Şekil 42 – 81).

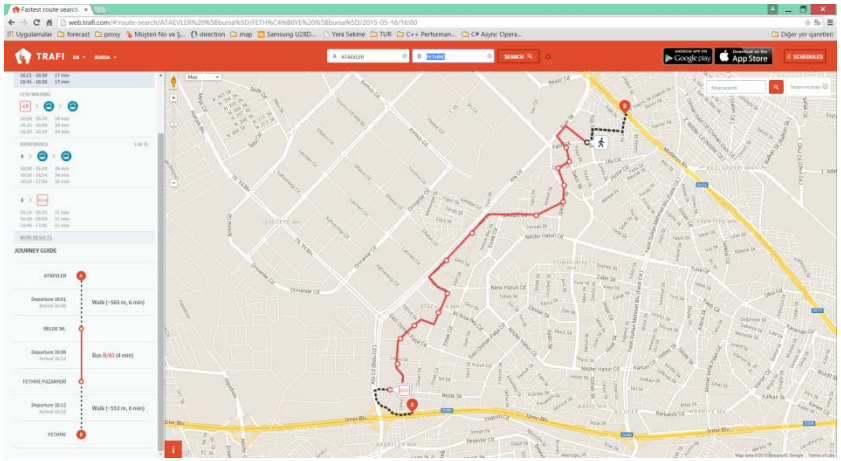
Tablo 23: Bursa TUA'sı için Seçilen Durak Çiftleri

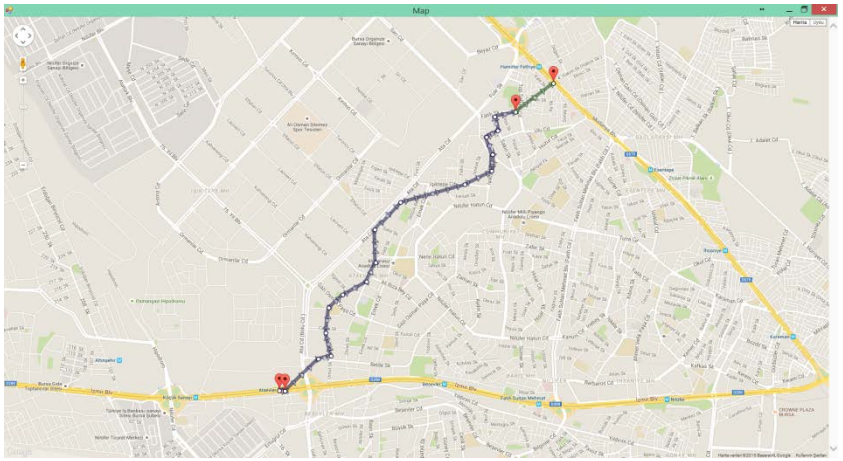
Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati	Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati
1	ATAEVLER	FETHİYE	16:00	6	CAĞDAŞ CAD. 1	BEŞEVLER CAD. 8	16:00
2	VATAN ANAOKULU	BİLLUR SK.	13:00	7	GEÇİT SAĞLIK OCAĞI	GÖLYAZI 6	08:00
3	BOYU GÜZEL SİTESİ	2. DERYA SK.	11:00	8	ALACAHIKA CAMİİ	ŞÜKRÜ ŞANKAYA A.L.	12:00
4	POLİS KOLEJİ	2. CEVİZ SK.	12:00	9	FETHİYE İSTASYONU	F.S.M. İSTASYONU	15:00
5	HİSAR MAH.9	KUMLA KÖYÜ 1	09:00	10	KIŞLA SK.	BAŞARAN İ.Ö.O.	08:00

Tablo 24: Bursa TUA'sı Çözüm Karşılaştırma Tablosu

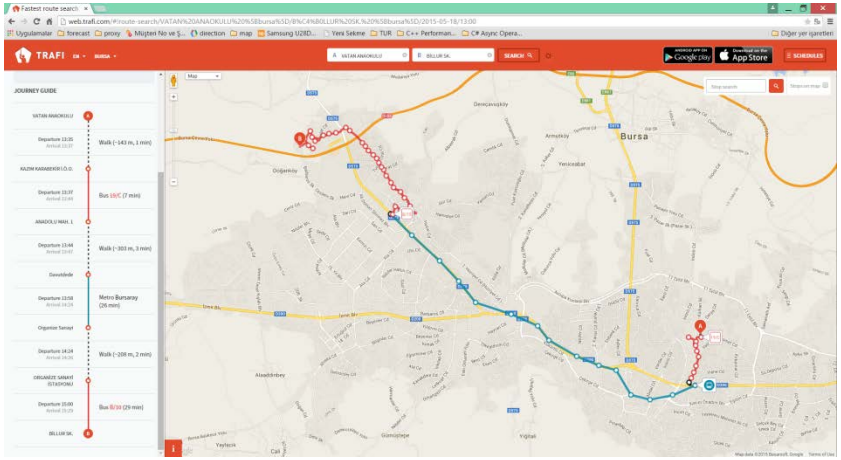
Durak Çifti Numarası	Trafi				TUR			
	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme
1	B/43	16:18	565	1118	B/43	16:12	49	98
2	19/C → Metro → B10	15:29	303	654	20 → B41/C →B10	15:25	65	65
3	B/36C → Metro → 17A	14:20	665	1066	B/36-C → Metro → 17/D	14:20	240	288
4	18/B	13:03	22	22	18/İ	13:05	227	268
5	139 → 137	10:09	799	982	104 → 138	10:09	68	76
6	1/TH → B44/B	17:23	0	0	4G → 48/A	16:45	190	233
7	B41/B → Metro → 5G	09:33	179	262	B41/B → Metro →5G	09:33	137	142
8	4/İ → 15/H	12:50	121	148	4/İ → 16/A	12:50	18	31
9	Metro → Metro	15:24	72	116	B/25	15:16	147	263
10	15/A → 9/İ	08:45	387	387	43/D → 1/A	08:24	210	413

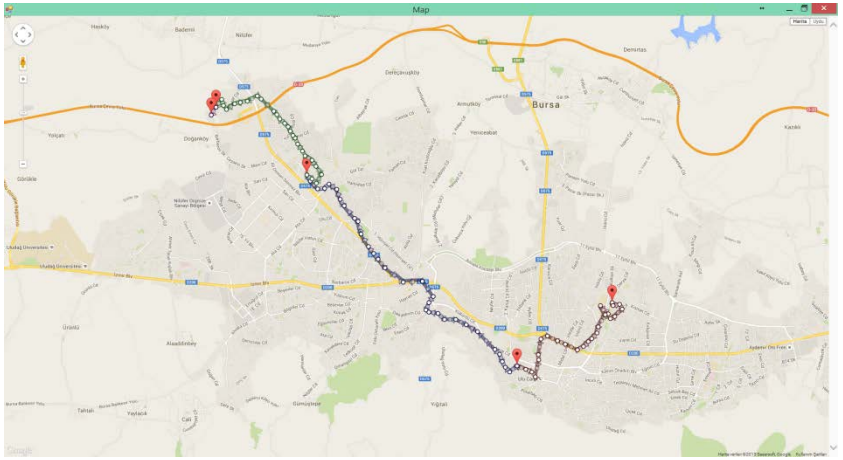
Şekil 42: Bursa TUA'sı Birinci Örnek Durak Çifti için Çözüm Haritaları





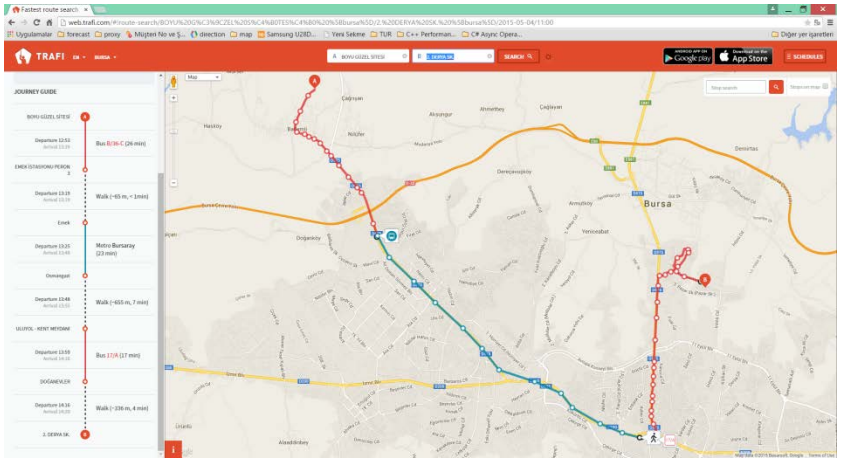
Şekil 43: Bursa TUA'sı İkinci Örnek Durak Çifti için Çözüm Haritaları

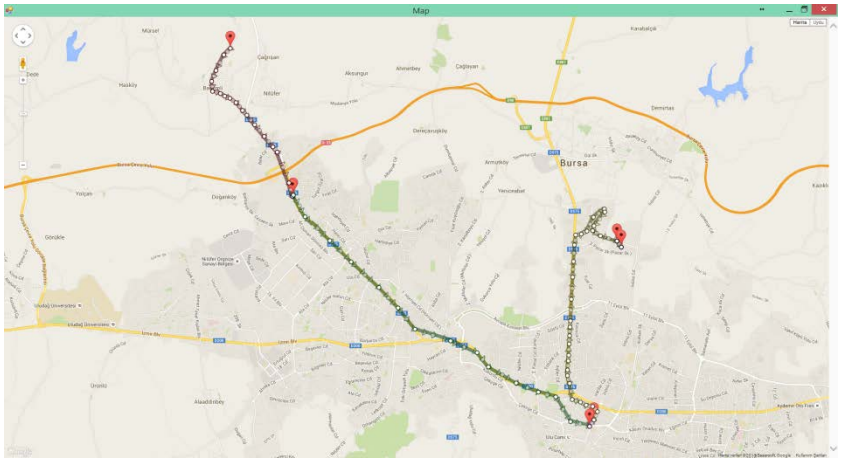




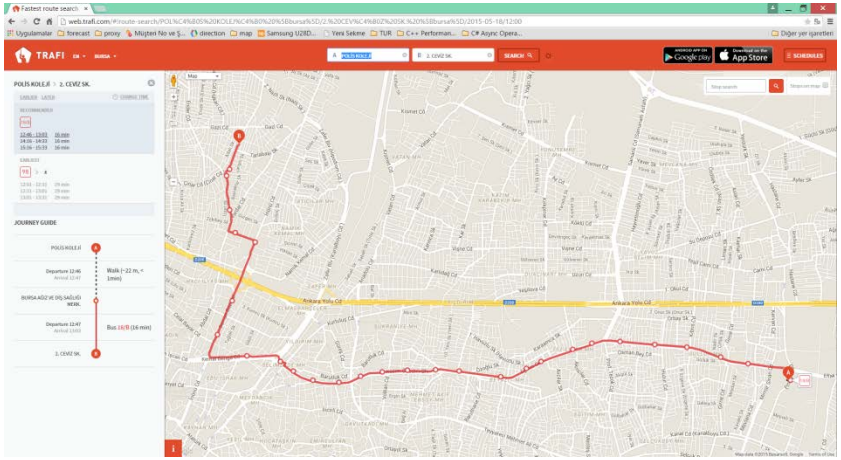
Trafi TUR

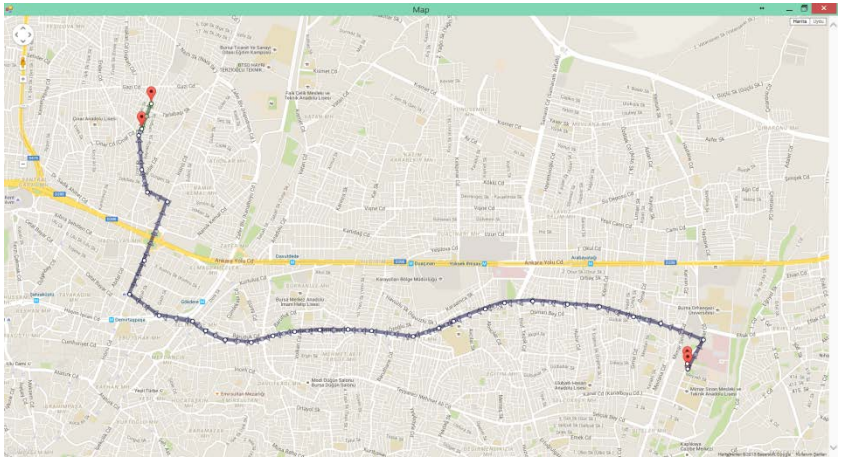
Şekil 44: Bursa TUA'sı Üçüncü Örnek Durak Çifti için Çözüm Haritaları





Şekil 45: Bursa TUA'sı Dördüncü Örnek Durak Çifti için Çözüm Haritaları

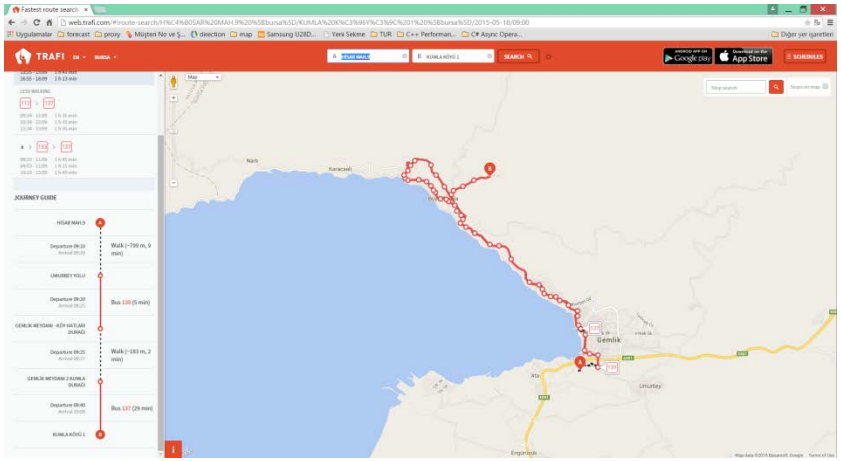


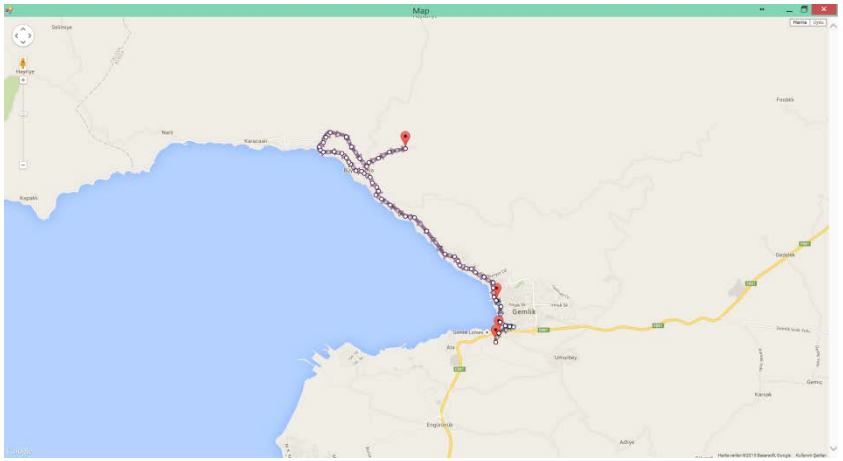


Trafi

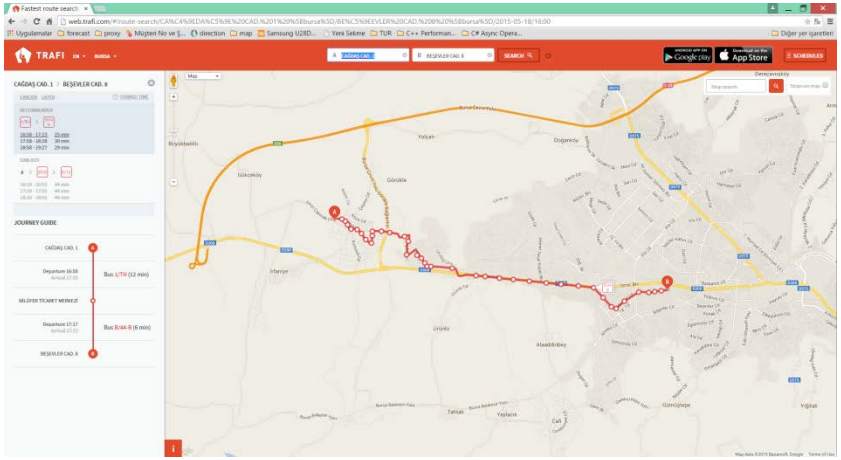
TUR

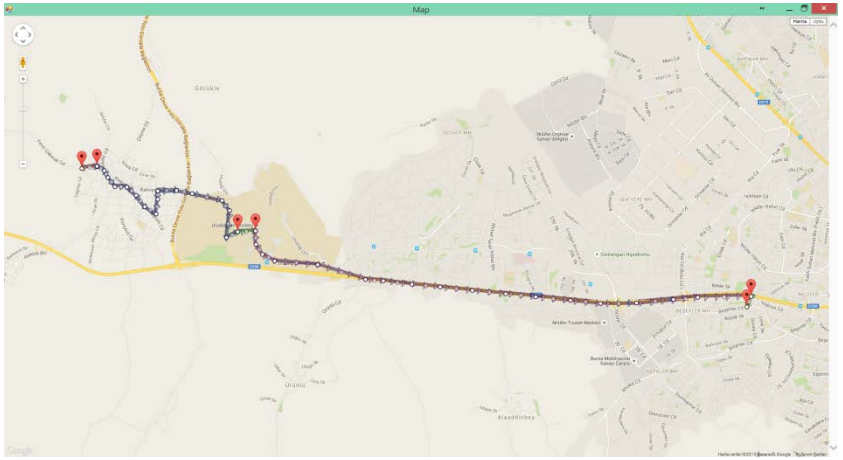
Şekil 46: Bursa TUA'sı Beşinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 47: Bursa TUA'sı Altıncı Örnek Durak Çifti için Çözüm Haritaları

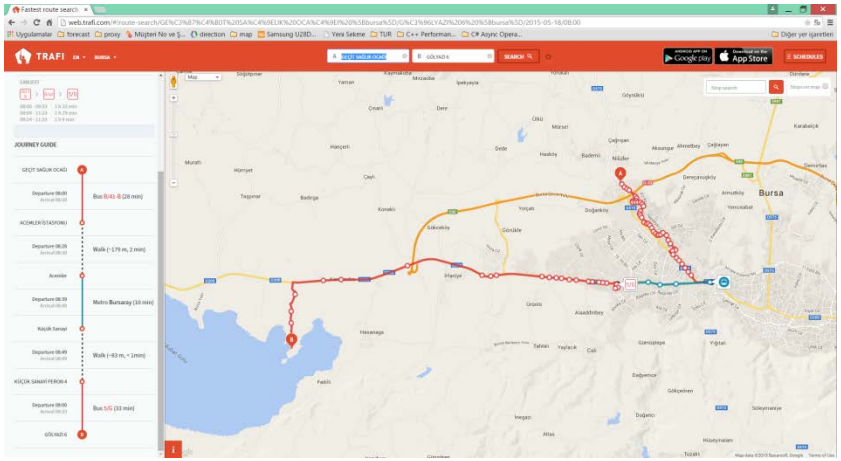


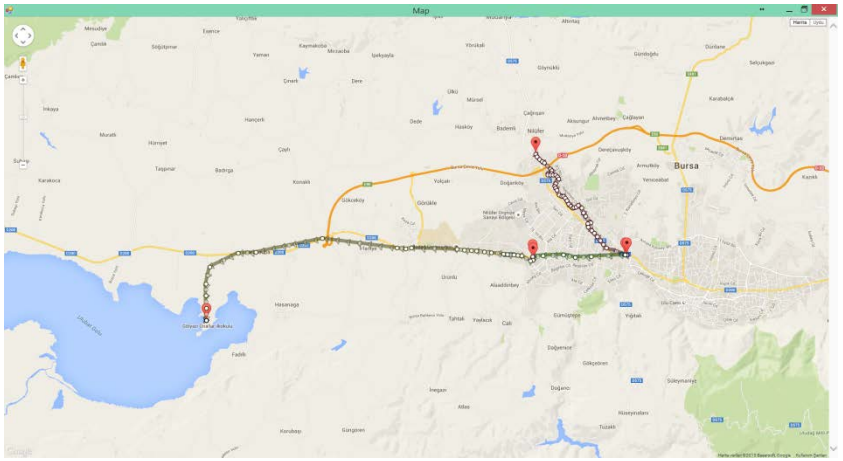


Trafi

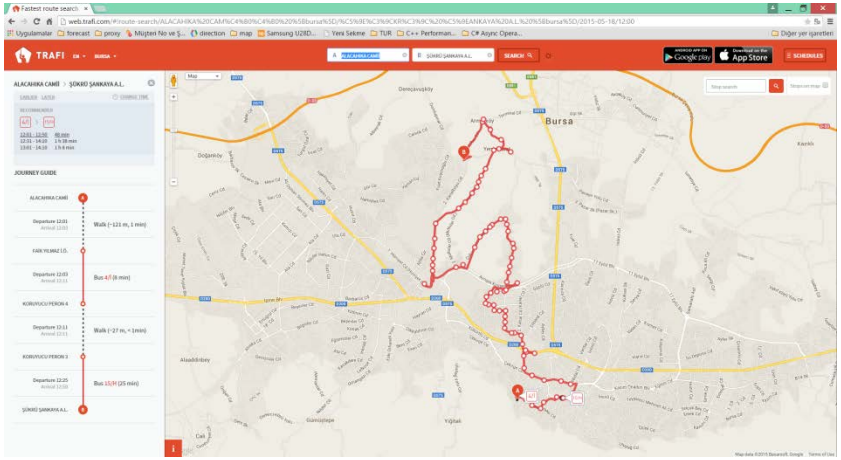
TUR

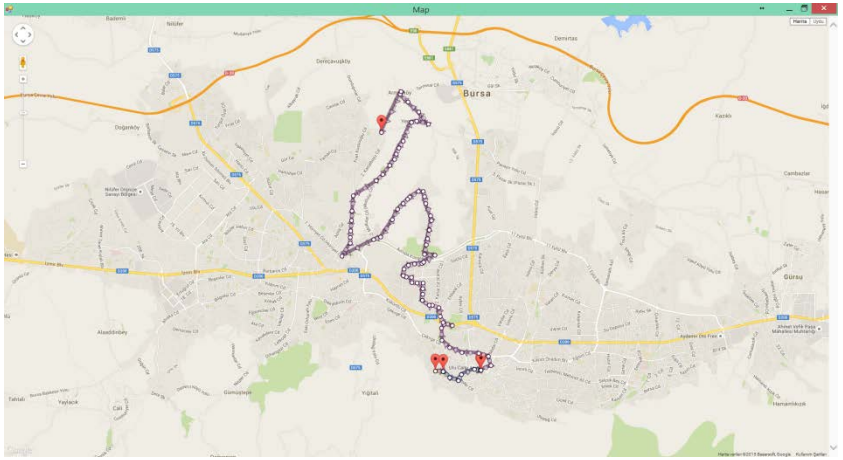
Şekil 48: Bursa TUA'sı Yedinci Örnek Durak Çifti için Çözüm Haritaları





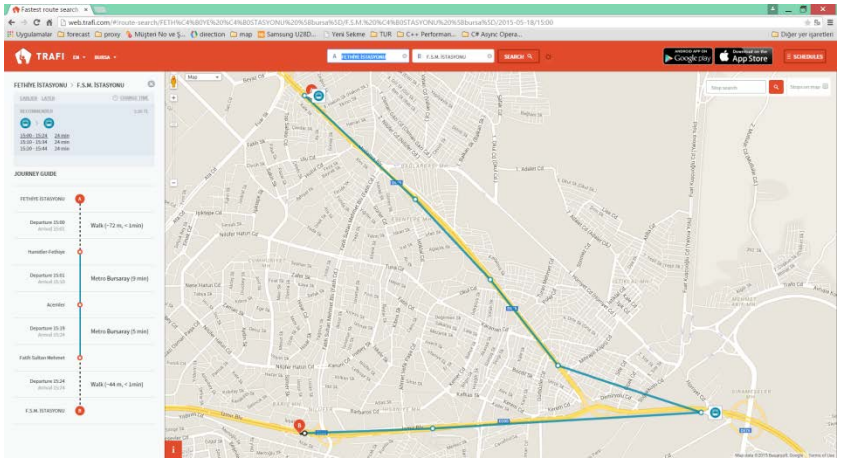
Şekil 49: Bursa TUA'sı Sekizinci Örnek Durak Çifti için Çözüm Haritaları

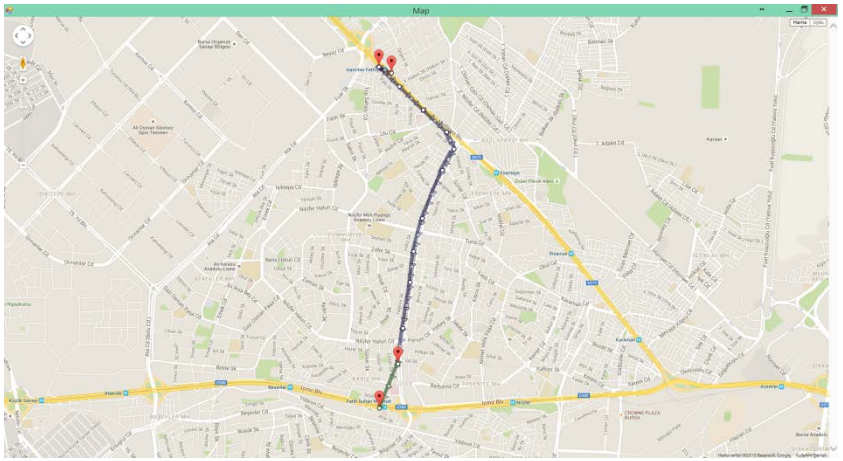




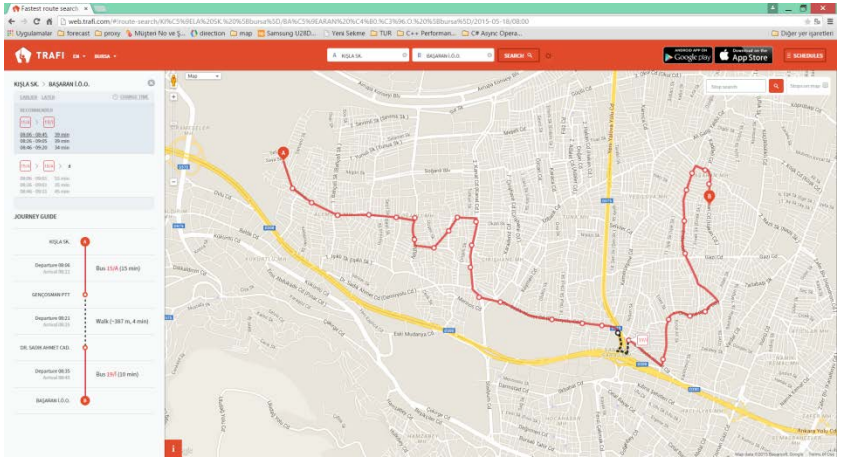
Trafi TUR

Şekil 50: Bursa TUA'sı Dokuzuncu Örnek Durak Çifti için Çözüm Haritaları

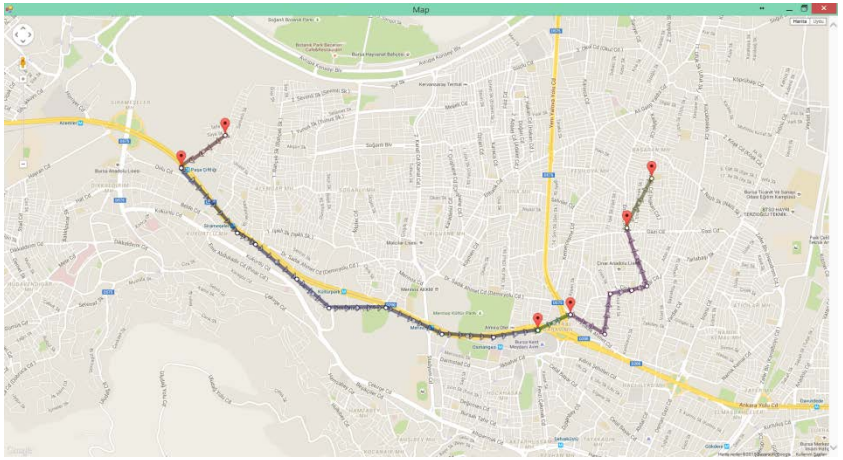




Şekil 51: Bursa TUA'sı Onuncu Örnek Durak Çifti için Çözüm Haritaları



Trafi



TUR

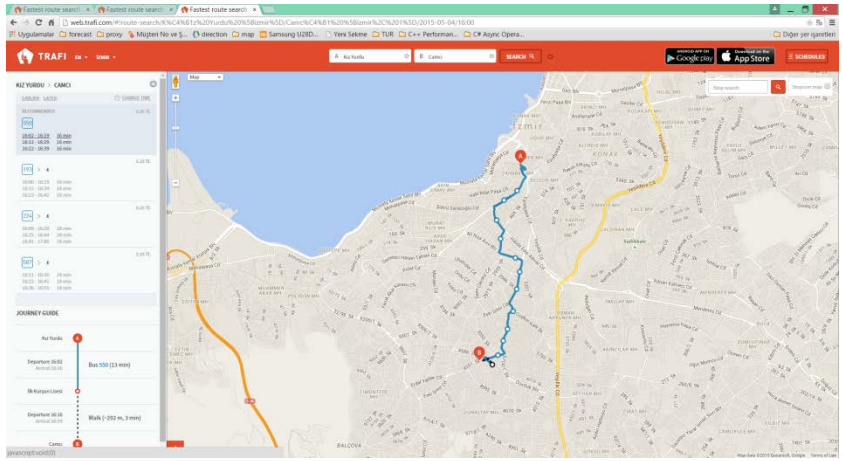
Tablo 25: İzmir TUA'sı için Seçilen Durak Çiftleri

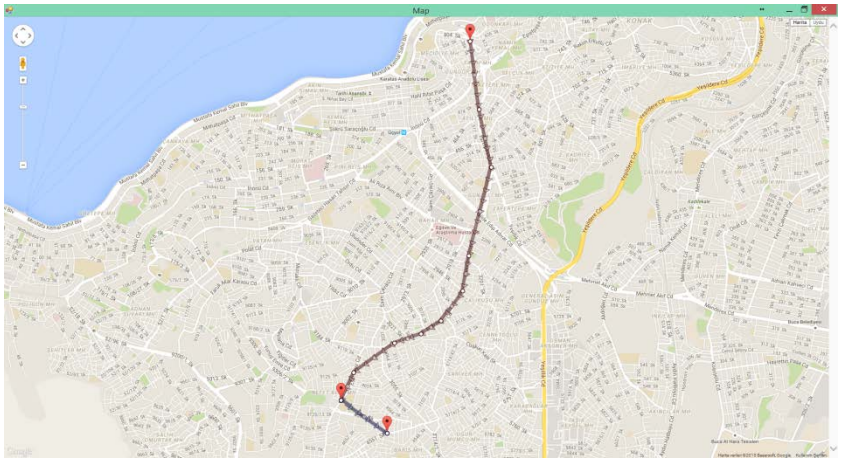
Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati	Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati
1	KIZ YURDU	CAMCI	16:00	6	KARAÇAM GİRİŞ	EREN	08:00
2	SEVDA	MELTEM	12:00	7	ÖZTÜRK	KÖPRÜ	09:00
3	ALPARSLAN MAHALLESİ MUHTARLIK	TARLA	15:00	8	GİŞELER	KORUCUK İLKÖĞRETİM OKULU	11:00
4	OSMAN DİRİK PARKI	SÖĞÜTÖREN MAVİ	11:00	9	YILDIZTEPE	YILDIZ MANDIRA	13:00
5	HÜRRİYET	NALDÖKEN	17:00	10	ANSIZCA GİRİŞ	ARMUTLU PARK	17:00

Tablo 26: İzmir TUA'sı Çözüm Karşılaştırma Tablosu

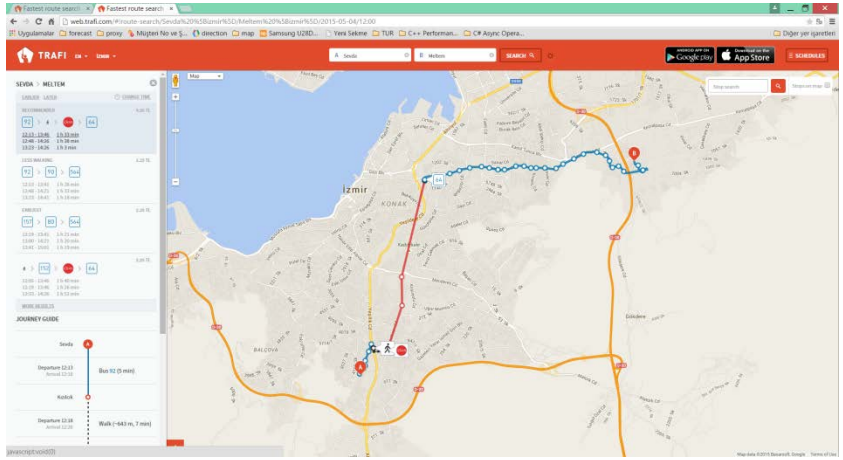
Durak Çifti Numarası	Trafi				TUR			
	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme
1	550	16:19	292	292	23	16:15	240	240
2	92→İZBAN→64	13:46	643	679	887→İZBAN→565	13:02	150	267
3	502 → Metro → 268	15:53	484	641	599→328→368	15:41	140	340
4	727 → 791	15:51	17	17	72→791	15:51	0	0
5	254→İZBAN	18:10	161	161	255→İZBAN	18:03	211	292
6	315→585	09:26	943	943	315→587	09:16	233	451
7	240→435	09:30	767	957	361→200	09:25	361	561
8	720→785	17:07	104	104	720→785	17:08	179	251
9	565→M→İZBAN→756	15:06	108	206	168→İZBAN→758	17:38	220	582
10	766→767	20:14	92	92	767→767	20:13	107	131

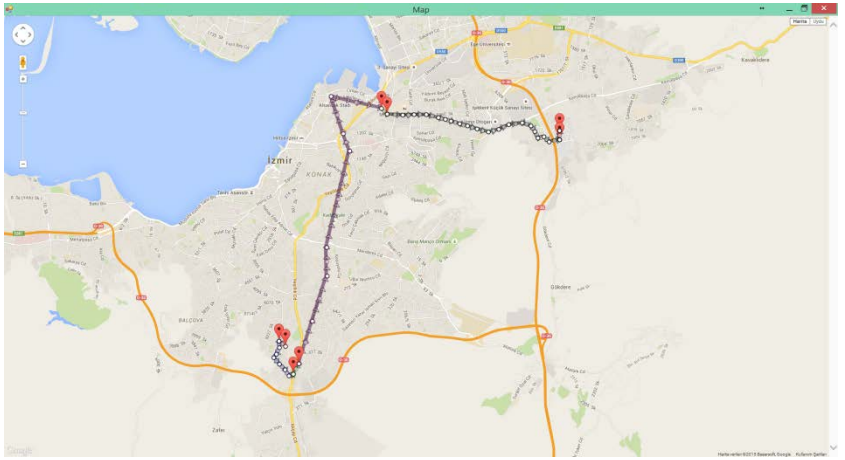
Şekil 52: İzmir TUA'sı Birinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 53: İzmir TUA'sı İkinci Örnek Durak Çifti için Çözüm Haritaları

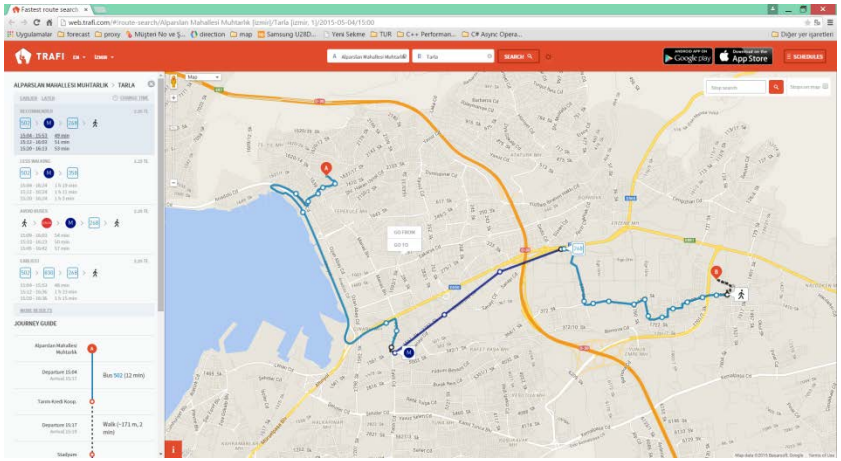


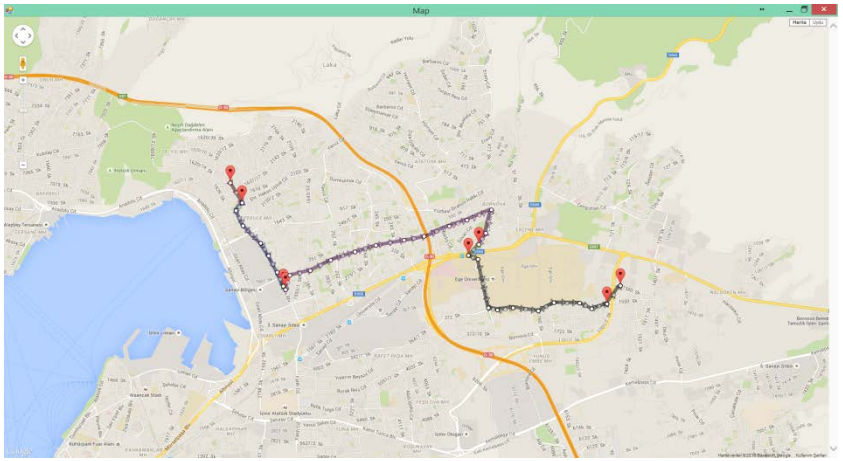


Trafi

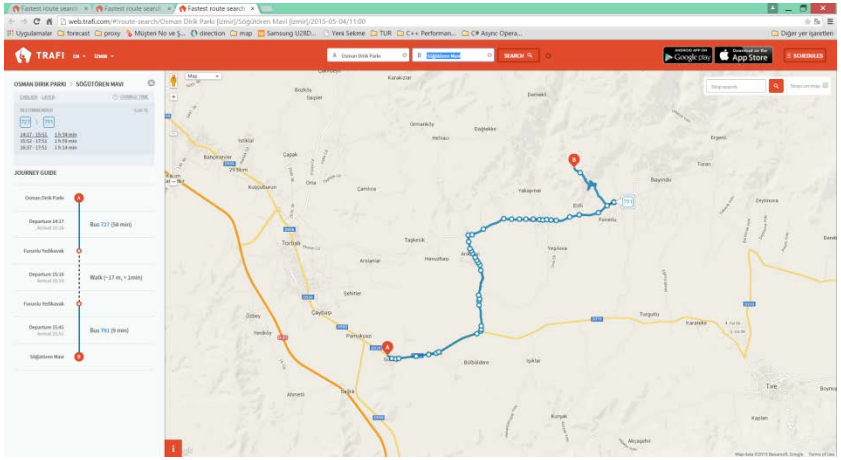
TUR

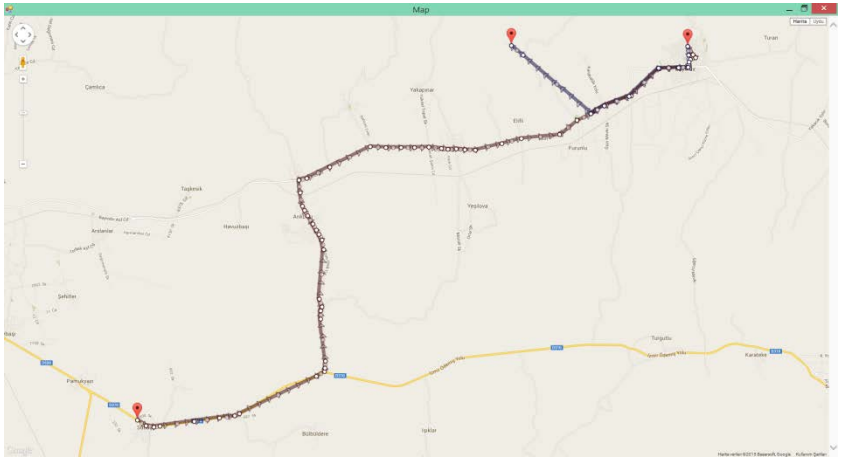
Şekil 54: İzmir TUA'sı Üçüncü Örnek Durak Çifti için Çözüm Haritaları





Şekil 55: İzmir TUA'sı Dördüncü Örnek Durak Çifti için Çözüm Haritaları

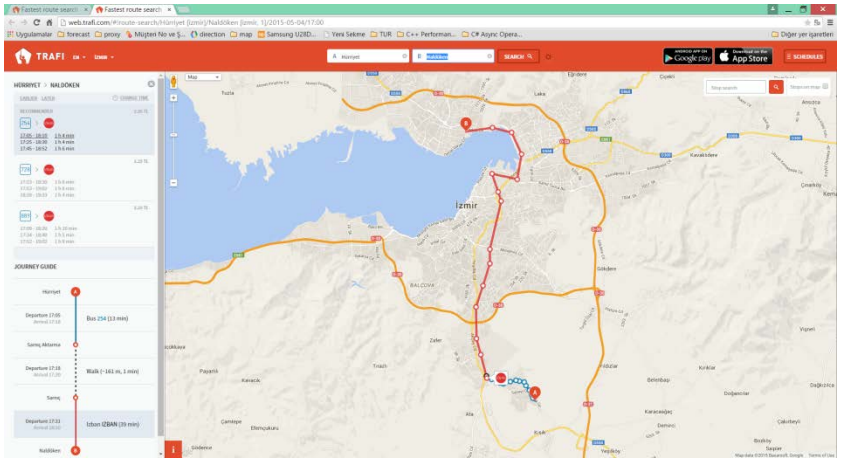


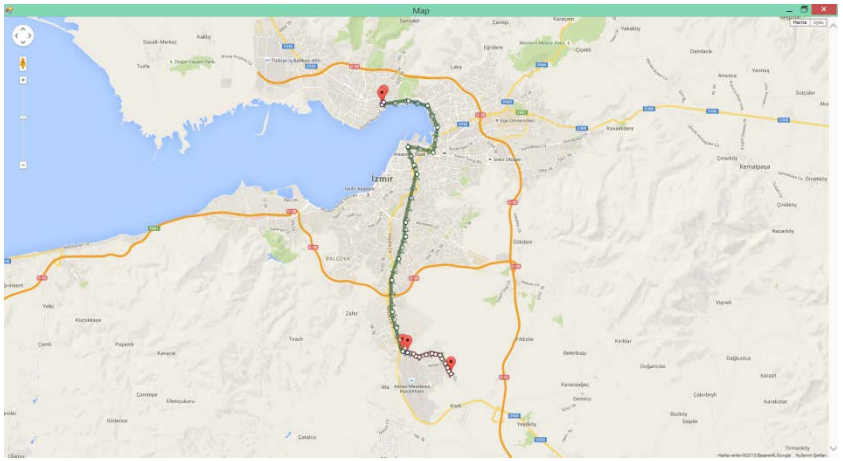


Trafi

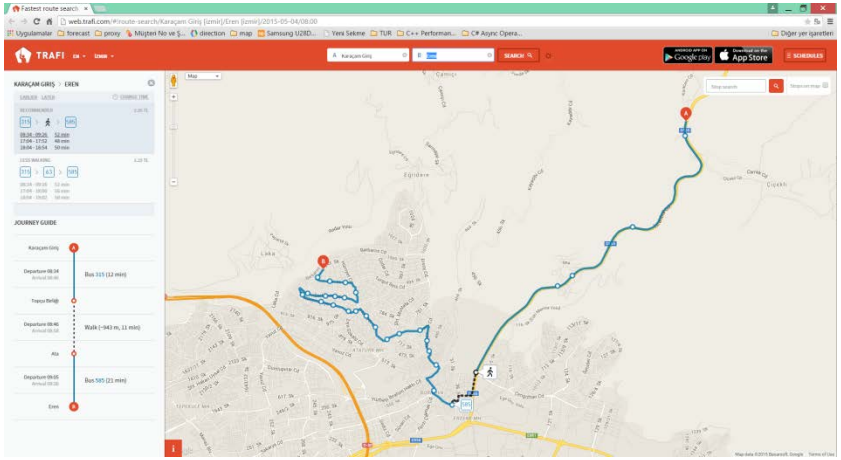
TUR

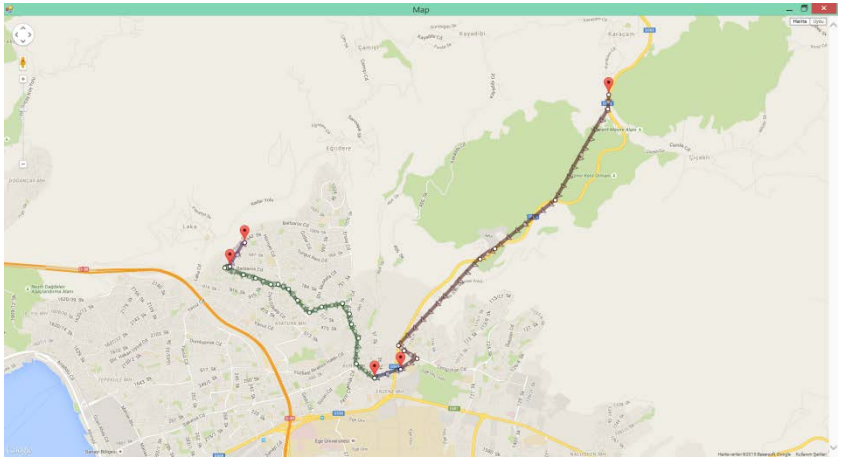
Şekil 56: İzmir TUA'sı Beşinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 57: İzmir TUA'sı Altıncı Örnek Durak Çifti için Çözüm Haritaları

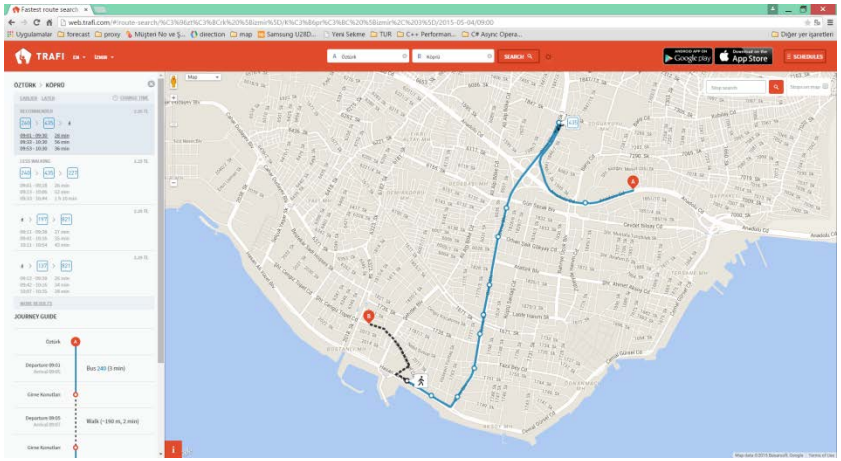


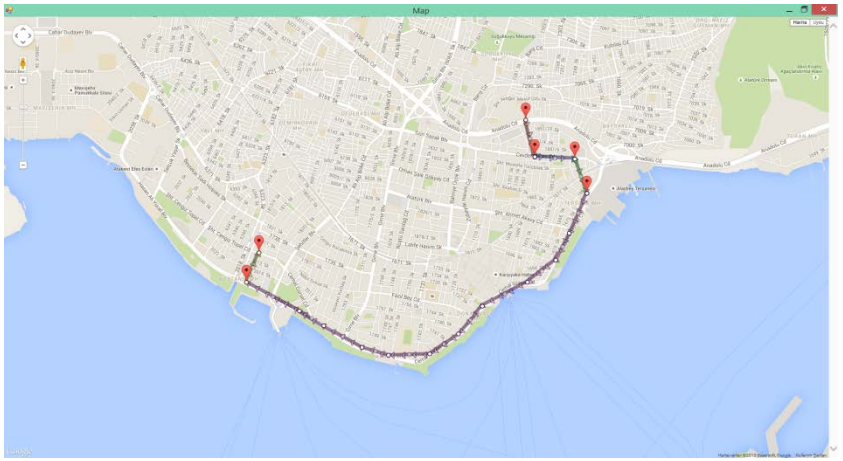


Trafi

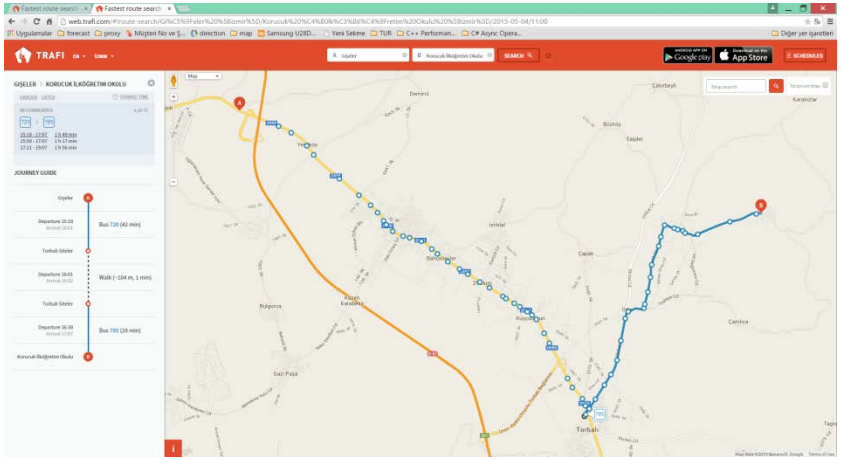
TUR

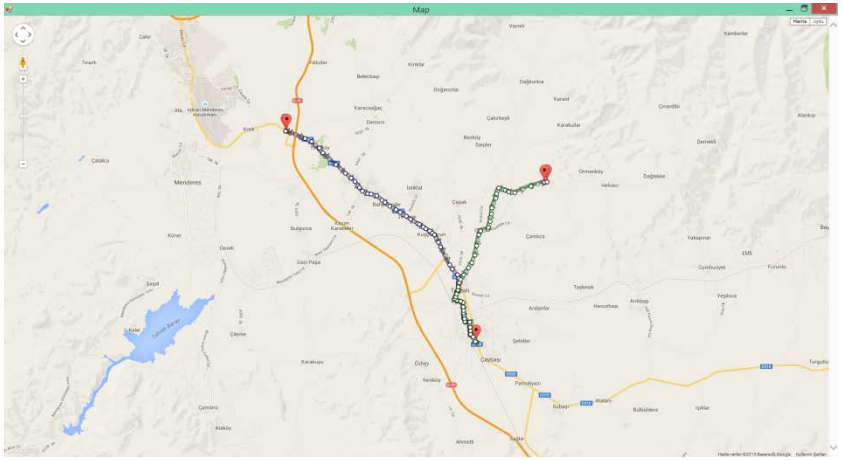
Şekil 58: İzmir TUA'sı Yedinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 59: İzmir TUA'sı Sekizinci Örnek Durak Çifti için Çözüm Haritaları

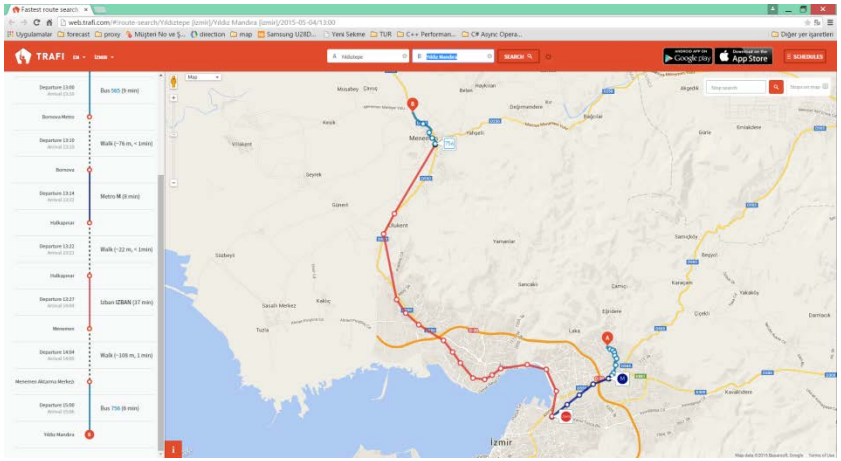


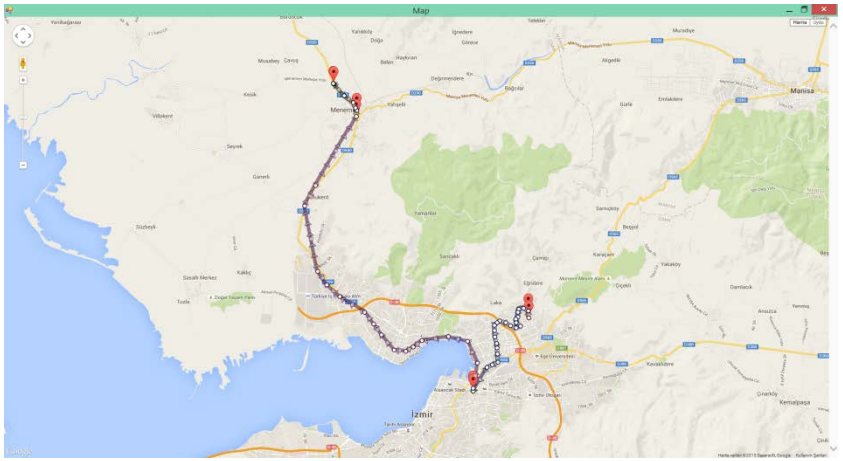


Trafi

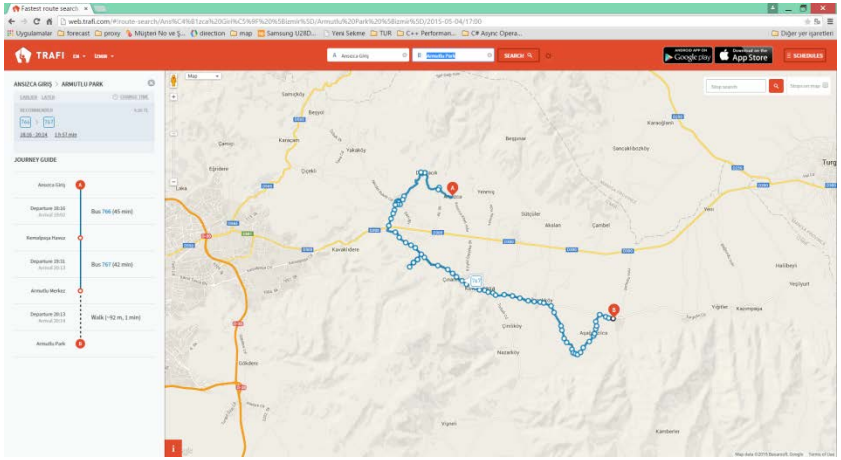
TUR

Şekil 60: İzmir TUA'sı Dokuzuncu Örnek Durak Çifti için Çözüm Haritaları

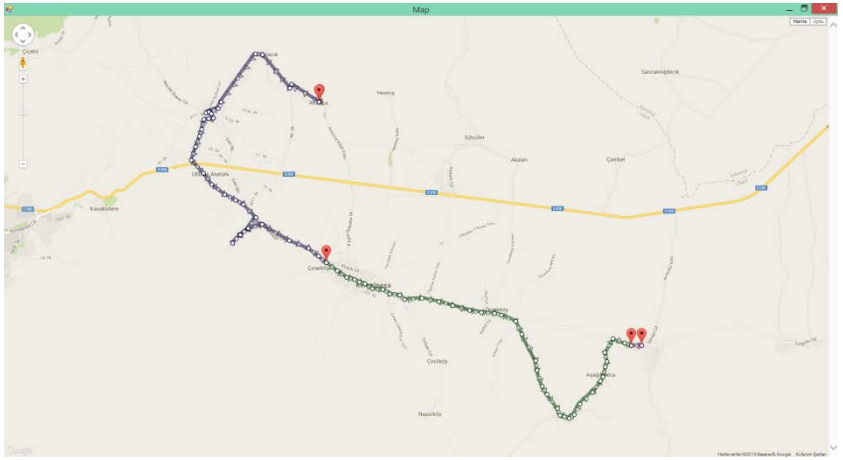




Şekil 61: İzmir TUA'sı Onuncu Örnek Durak Çifti için Çözüm Haritaları



Trafi



TUR

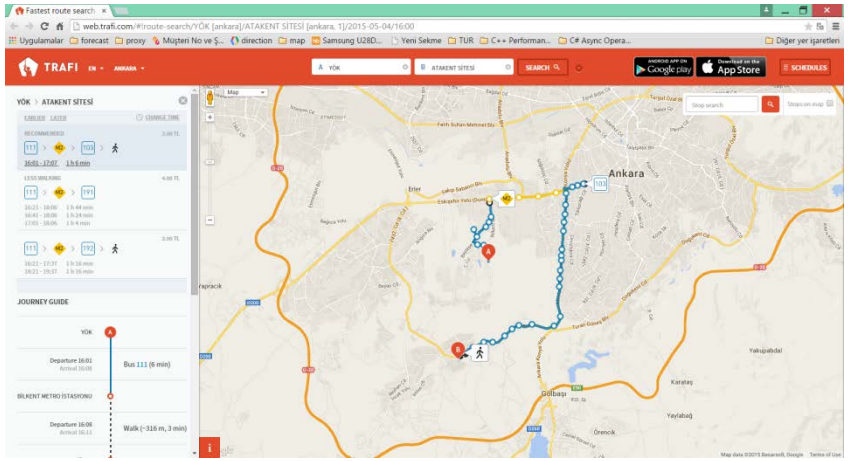
Tablo 27: Ankara TUA'sı için Seçilen Durak Çiftleri

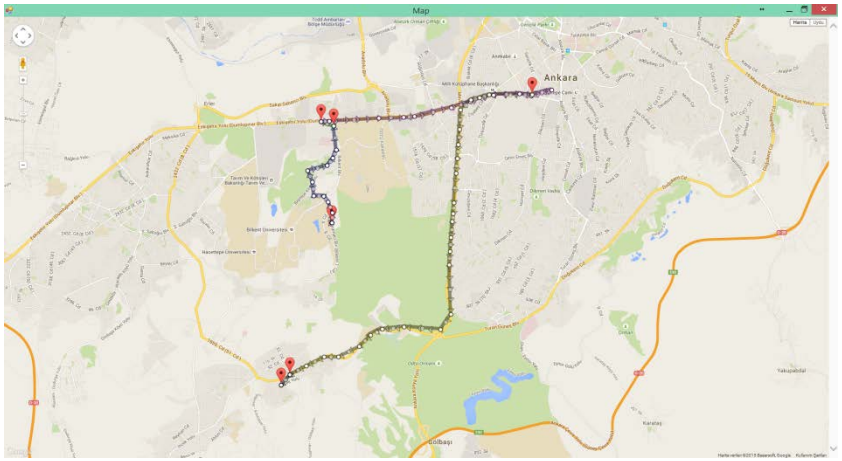
Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati	Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati
1	YÖK	ATAKENT SİTESİ	16:00	6	MECLİS	SAKIZ AĞACI CADDESİ 1.DURAK	10:00
2	ANAFARTALAR	SİNPAŞ	15:00	7	Fatih	MURAT KARA PARKI	17:00
3	ULUS YİBA ÇARŞISI	DALGIÇ 2. DURAK	12:00	8	ORTAKÖY 4. DURAK	YUVA KÖY DÖNÜŞ YOLU 4 DURAK	09:00
4	ASELSAN İLKOKULU	TEPEBAŞI	17:00		BAKIMEVİ	ERYAMAN MİGROS	12:00
5	NUMUNE HASTANESİ	YAŞAR DOĞU CADDESİ KESİŞİMİ	11:00	9	YÜZBAŞI MUSTAFA ERTUĞRUL CADDESİ 1. DURAK	HATÇA ANA	10:00

Tablo 28: Ankara TUA'sı Çözüm Karşılaştırma Tablosu

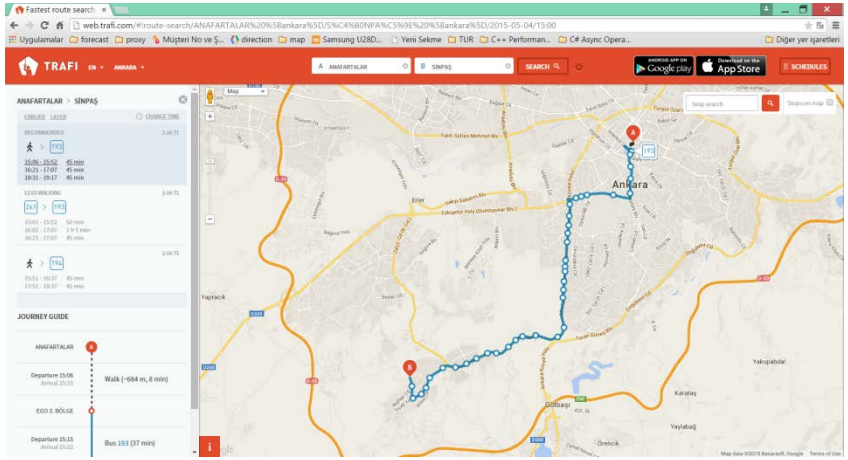
Durak Çifti Numarası	Trafi				TUR			
	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme
1	111→Metro-2→103	17:07	749	1199	111-1→513→104	17:02	196	403
2	193	15:52	684	684	195	15:54	231	458
3	421-3	12:20	409	629	422	12:11	224	332
4	297-3→422	17:49	326	326	298→462	17:35	232	242
5	Metro-1→Metro-3	12:04	847	1178	515-3	12:00	220	354
6	203-3	10:42	652	854	253	15:06	158	299
7	Metro-3→Merto-1→284	18:27	185	185	526→282	18:25	129	64
8	354-3→Metro-1→217	14:06	113	197	361→Metro-1→220-3	14:06	28	57
9	113→540	13:06	290	290	114→542	12:59	61	111
10	563-3→Metro-1→145-3	11:14	831	1165	565→145-3	10:54	176	482

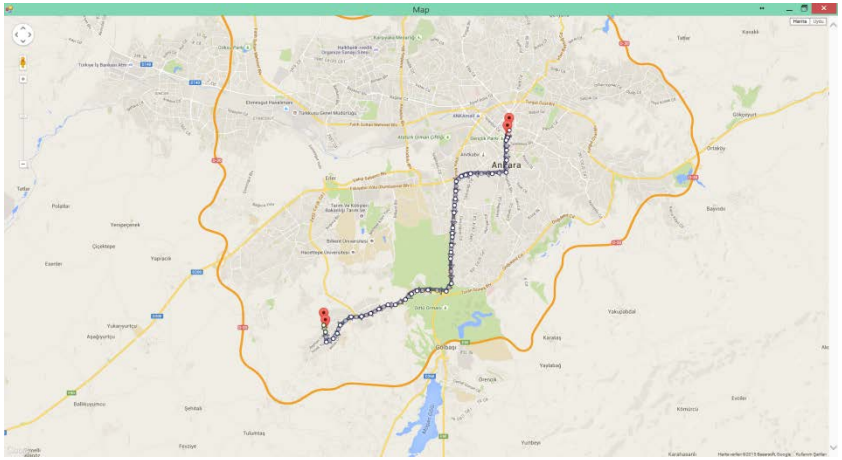
Şekil 62: Ankara TUA'sı Birinci Örnek Durak Çifti için Çözüm Haritaları





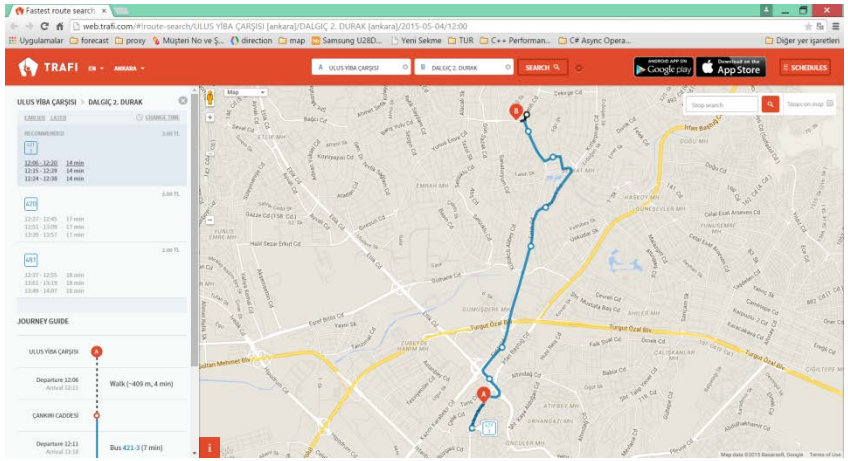
Şekil 63: Ankara TUA'sı İkinci Örnek Durak Çifti için Çözüm Haritaları

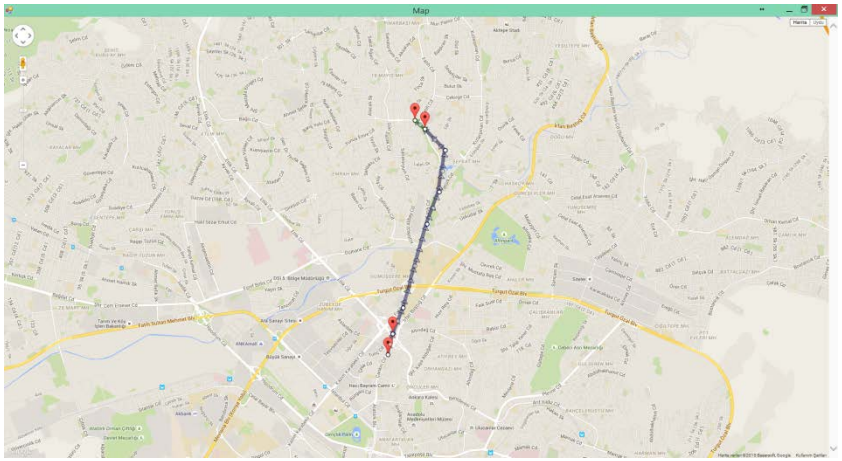




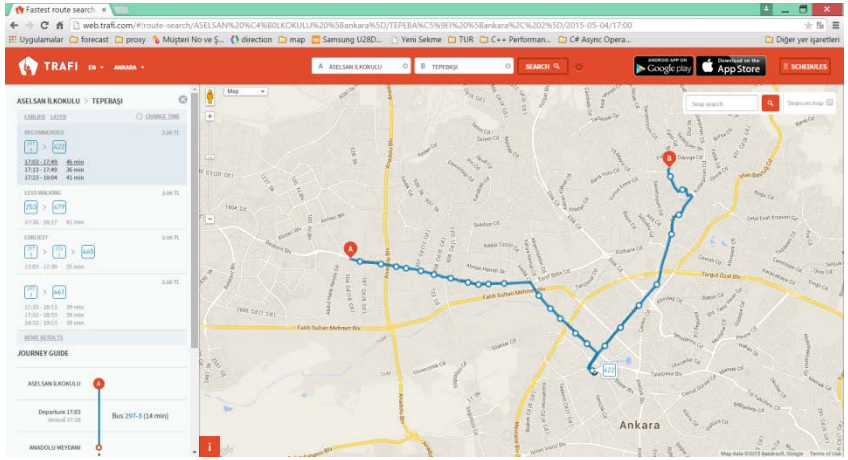
Trafi TUR

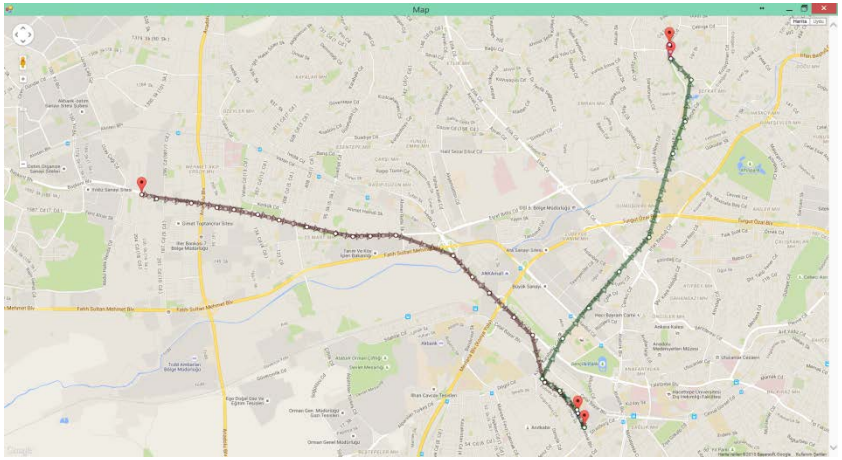
Şekil 64: Ankara TUA'sı Üçüncü Örnek Durak Çifti için Çözüm Haritaları





Şekil 65: Ankara TUA'sı Dördüncü Örnek Durak Çifti için Çözüm Haritaları

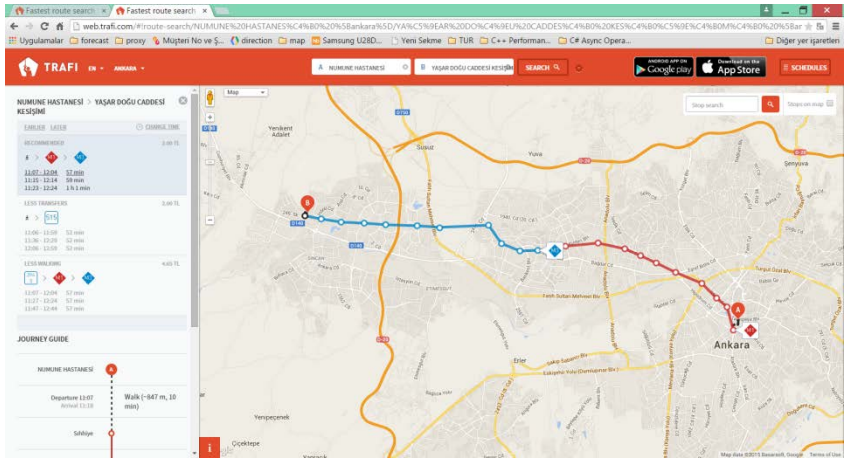


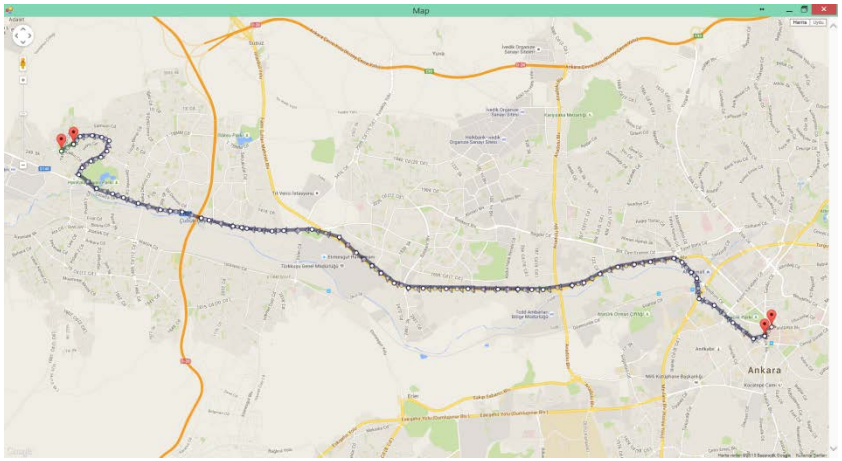


Trafi

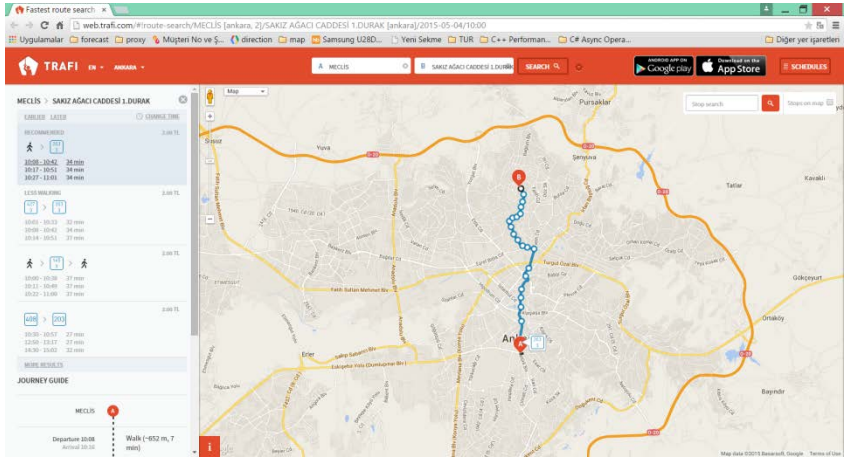
TUR

Şekil 66: Ankara TUA'sı Beşinci Örnek Durak Çifti için Çözüm Haritaları

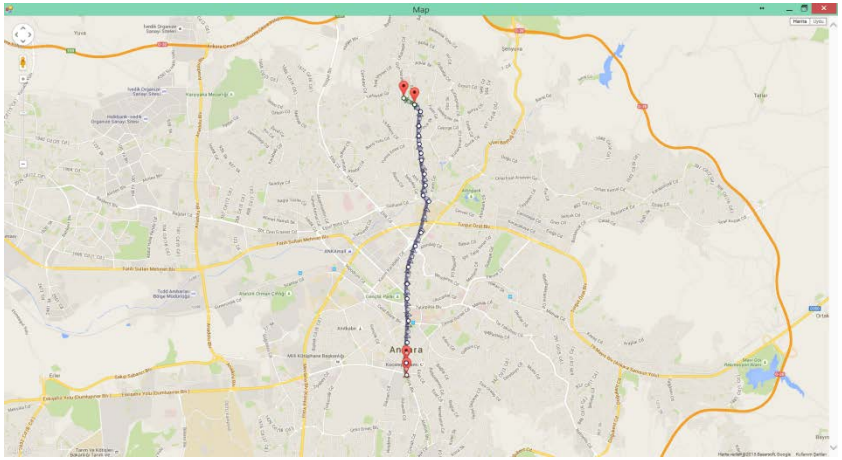




Şekil 67: Ankara TUA'sı Altıncı Örnek Durak Çifti için Çözüm Haritaları

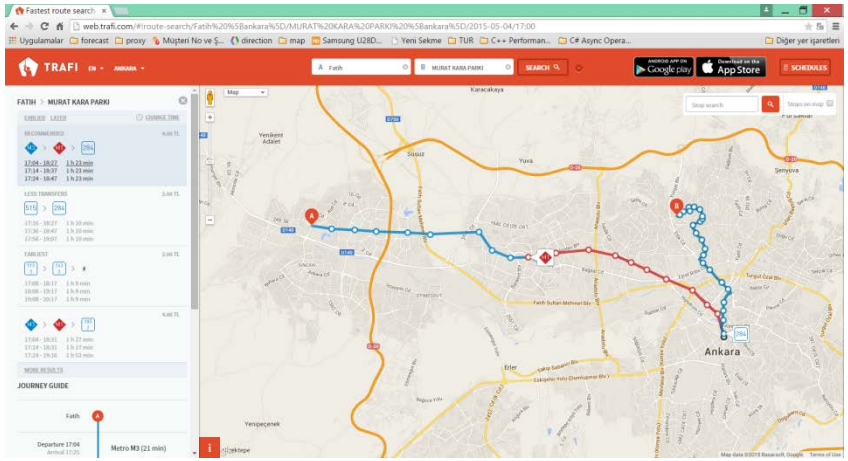


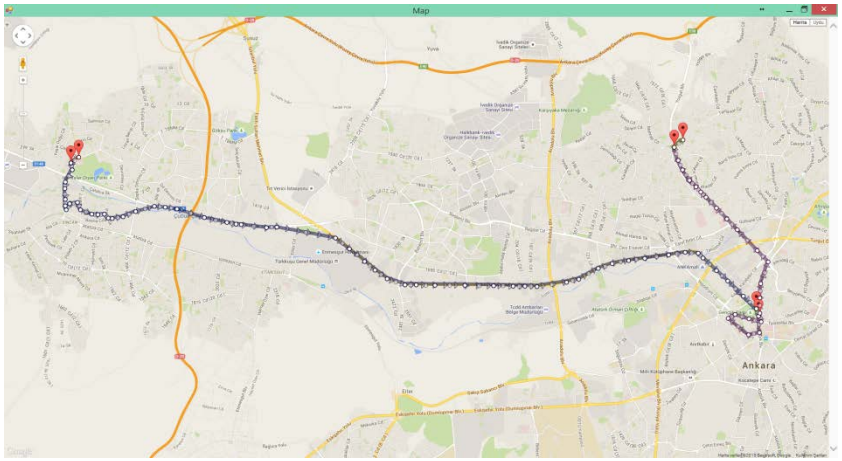
Trafi



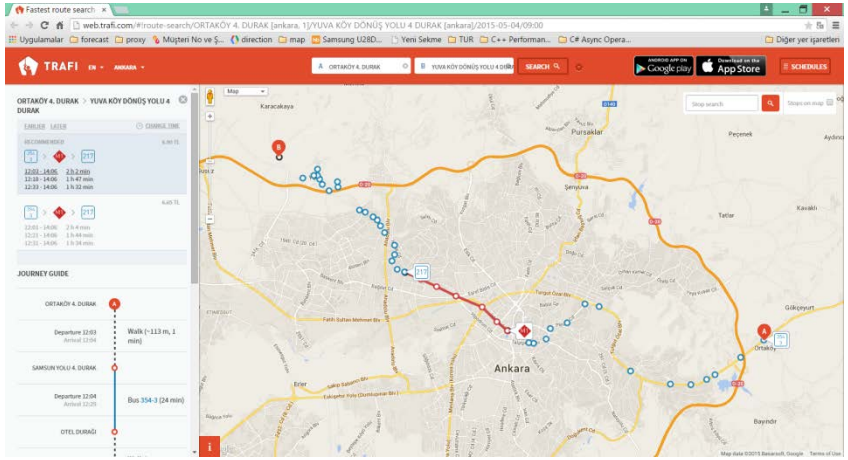
TUR

Şekil 68: Ankara TUA'sı Yedinci Örnek Durak Çifti için Çözüm Haritaları

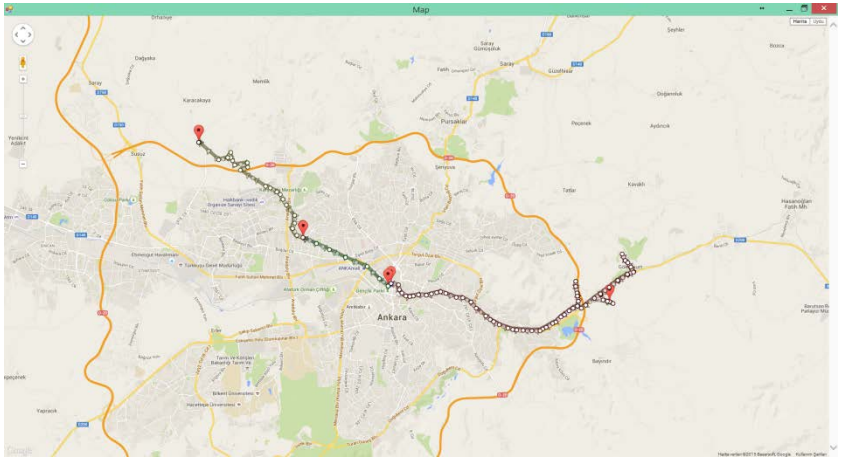




Şekil 69: Ankara TUA'sı Sekizinci Örnek Durak Çifti için Çözüm Haritaları

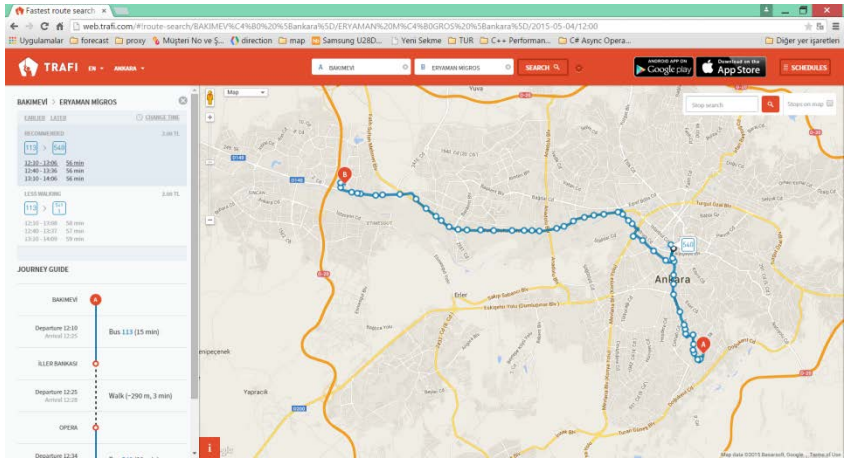


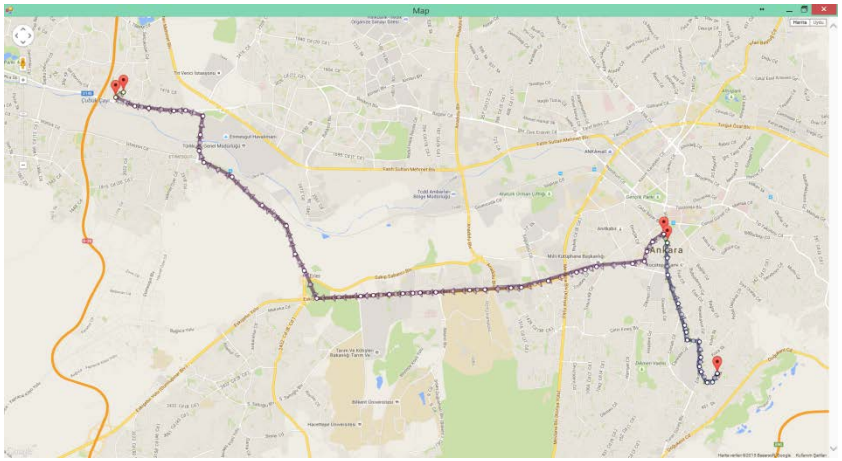
Trafi



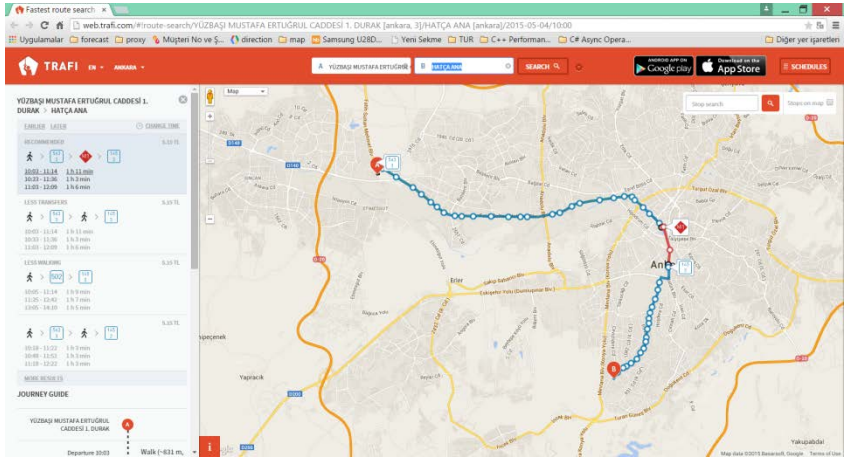
TUR

Şekil 70: Ankara TUA'sı Dokuzuncu Örnek Durak Çifti için Çözüm Haritaları

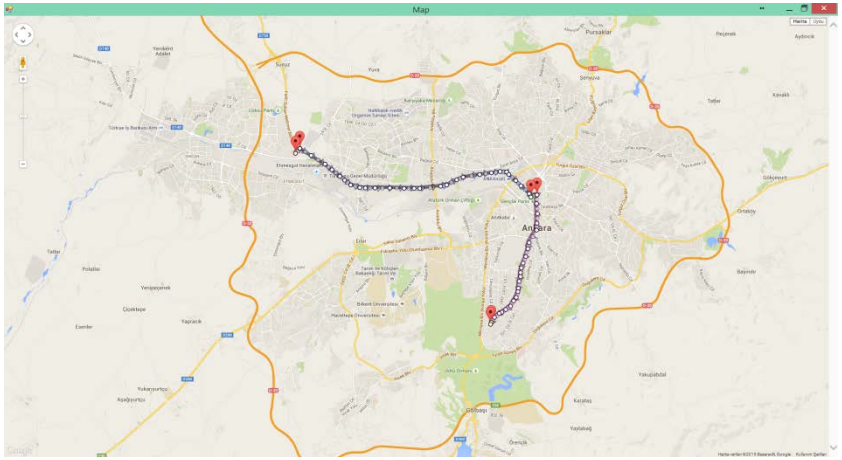




Şekil 71: Ankara TUA'sı Onuncu Örnek Durak Çifti için Çözüm Haritaları



Trafi



TUR

Tablo 29: İstanbul TUA'sı için Seçilen Durak Çiftleri

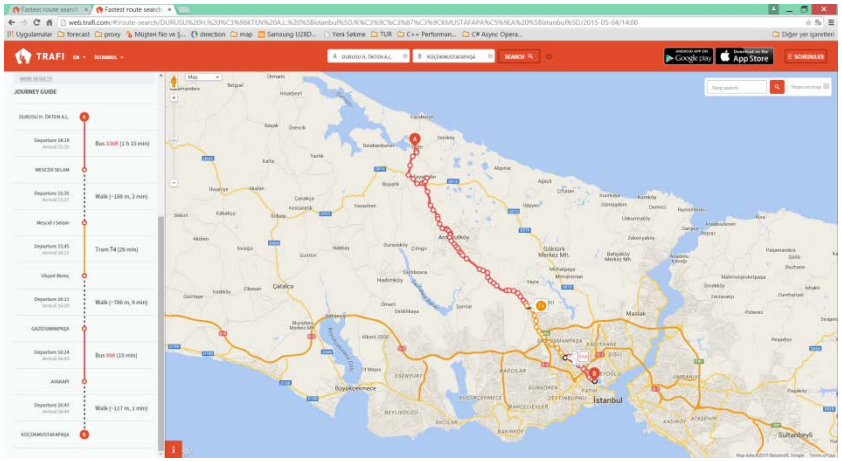
Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati	Durak Çifti Numarası	Başlangıç Durak Adı	Varış Durak Adı	Hareket Saati
1	DURUSU H. ÖKTEN A.L.	KÜÇÜKMUSTAFAPAŞA	14:00	6	F.ÇAKMAK MAHALLESİ	CANSU SOKAK	10:00
2	ALT SOKAK	HALİÇ METRO	12:00	7	SABRİ AKIN İ.Ö.O	ESKİ SİLİVRİ CAD.	10:00
3	ZEYTİN SOKAK	4.LEVEND METRO	15:00	8	HASANPAŞA	11. CADDE	16:00
4	MERV CADDESİ	PLEVNE CADDESİ	16:00	9	ÇİFTEHAVUZLAR	ÇİFTEÇINARLAR	11:00
5	ORTABAYIR	TOKİ EVLERİ	12:00	10	BEYİT SOKAK	HÜSEYİN T.SİPAHİ İÖO	11:00

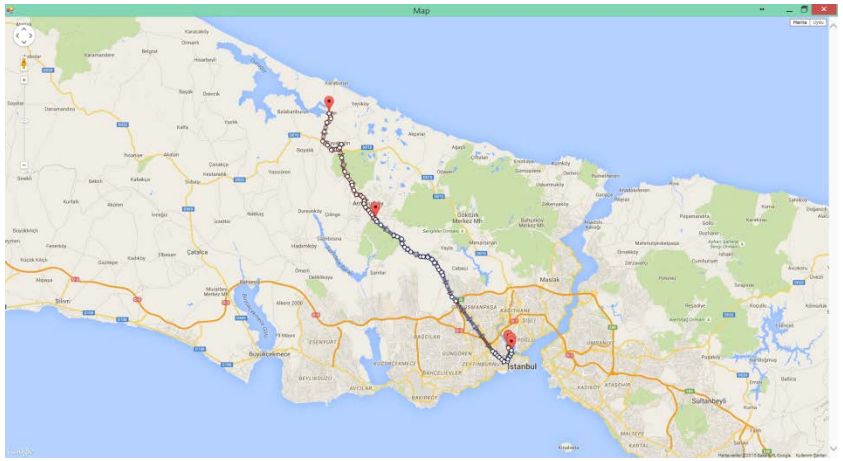
Tablo 30: İstanbul TUA'sı Çözüm Karşılaştırma Tablosu

Durak Çifti Numarası	Trafi				TUR			
	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme	Çözüm	Varış Zamanı	Maksimum Yürüme	Toplam Yürüme
1	336K→T4→99A	16:44	786	1091	336M → 336 → 99Y	16:29	108	203
2	151 → M2	13:03	378	658	59RS → M3	12:55	37	61
3	500T	16:35	758	1474	132V → 122H	17:44	95	172
4	MR10→Marmaray→M4→16C	17:30	623	1539	93T→34B →16Z	16:35	200	253
5	62 → 522ST → 139A	13:48	363	423	41ST → 522B	15:00	241	585
6	132P	11:13	765	1357	KM23 → 132C	11:13	158	441
7	36Bİ → T4 → 34BZ →401T	13:22	240	602	36Bİ → 79G → 401T	13:02	152	204
8	522ST → 34BZ	18:23	754	229	54Ç → 34G	00:03	231	255
9	4 → 15BK	12:37	315	336	D[5] → 150	12:20	227	351
10	Çözüm Bulunamadı				19E	11:28	154	276

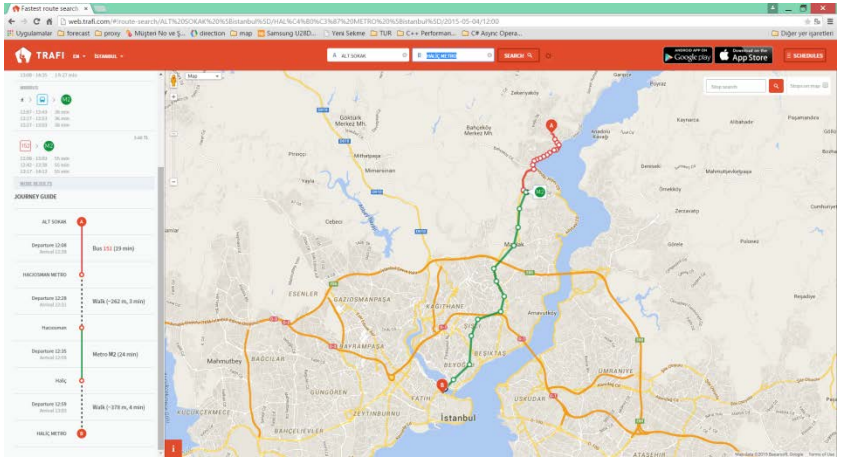
[5] Bostancı - Tavukçu Yolu – Dudullu Dolmuş Hattı

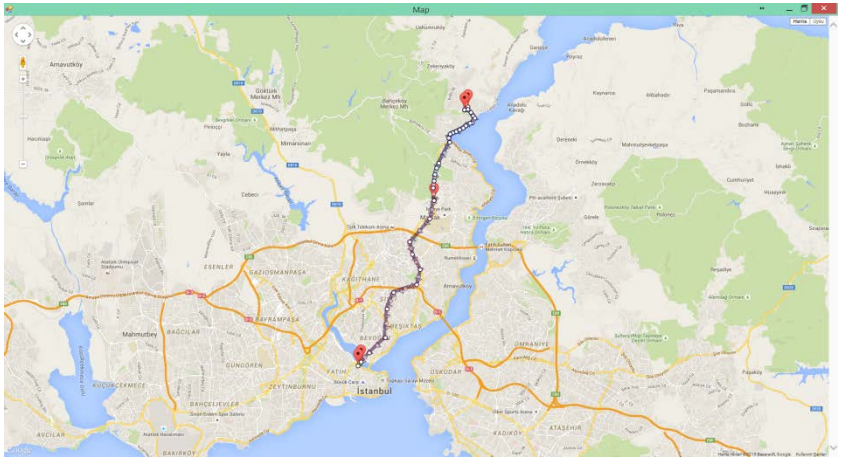
Şekil 72: İstanbul TUA'sı Birinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 73: İstanbul TUA'sı İkinci Örnek Durak Çifti için Çözüm Haritaları

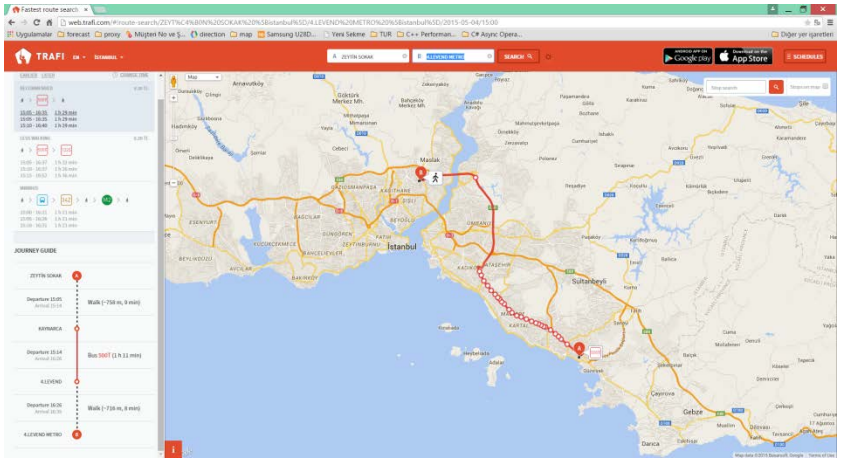


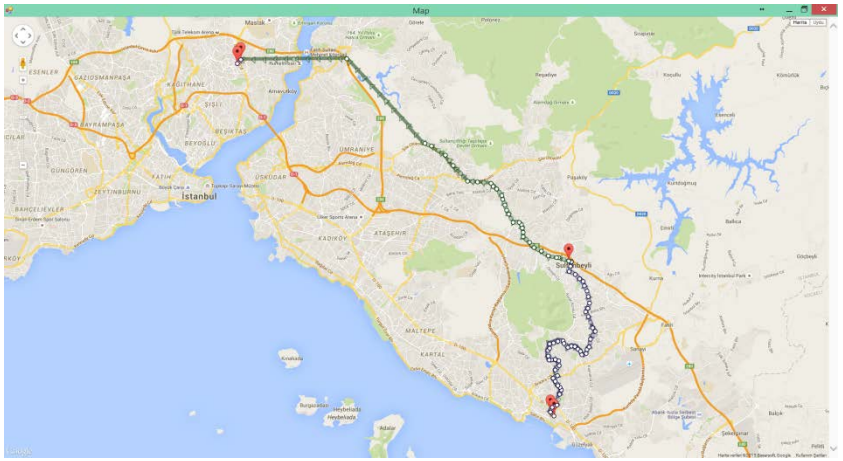


Trafi

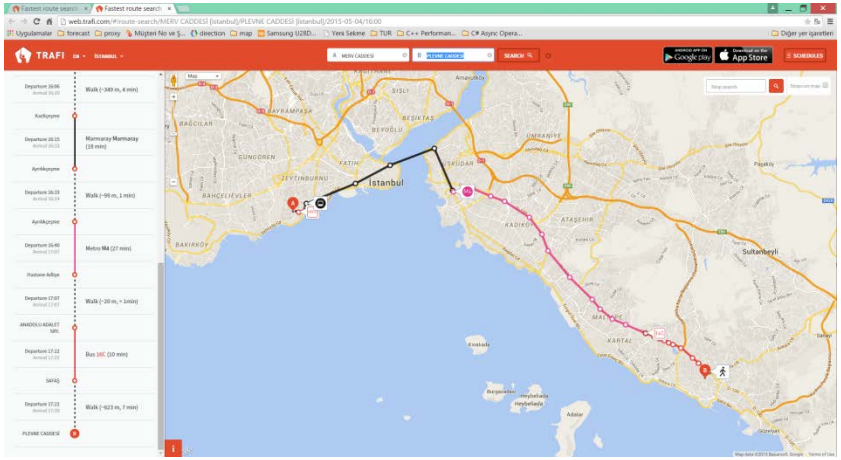
TUR

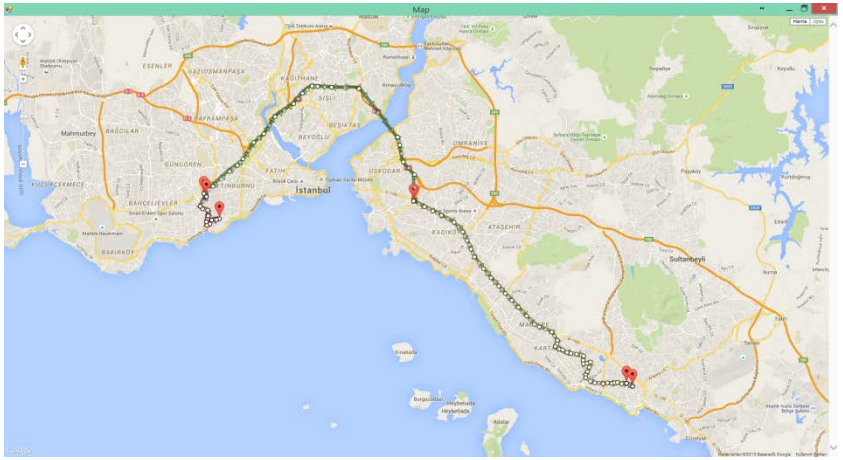
Şekil 74: İstanbul TUA'sı Üçüncü Örnek Durak Çifti için Çözüm Haritaları





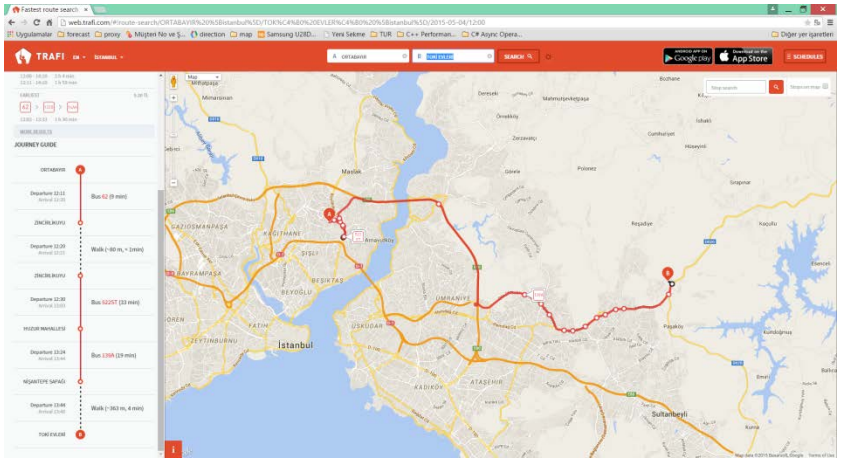
Şekil 75: İstanbul TUA'sı Dördüncü Örnek Durak Çifti için Çözüm Haritaları

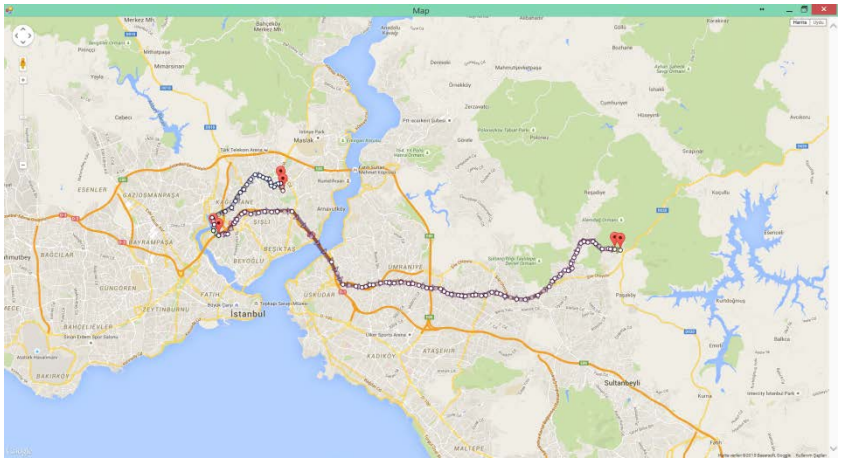




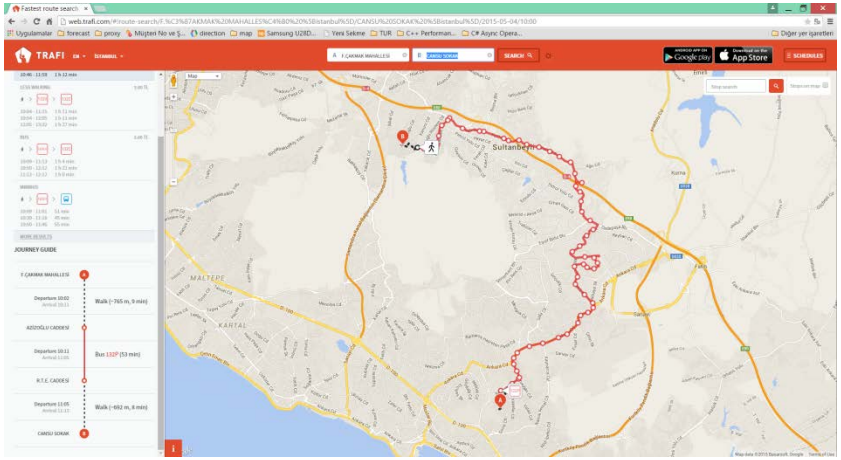
Trafi TUR

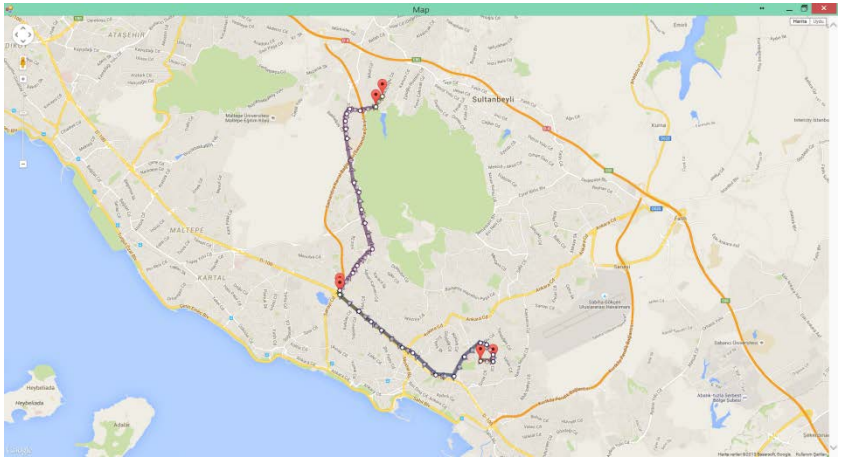
Şekil 76: İstanbul TUA'sı Beşinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 77: İstanbul TUA'sı Altıncı Örnek Durak Çifti için Çözüm Haritaları

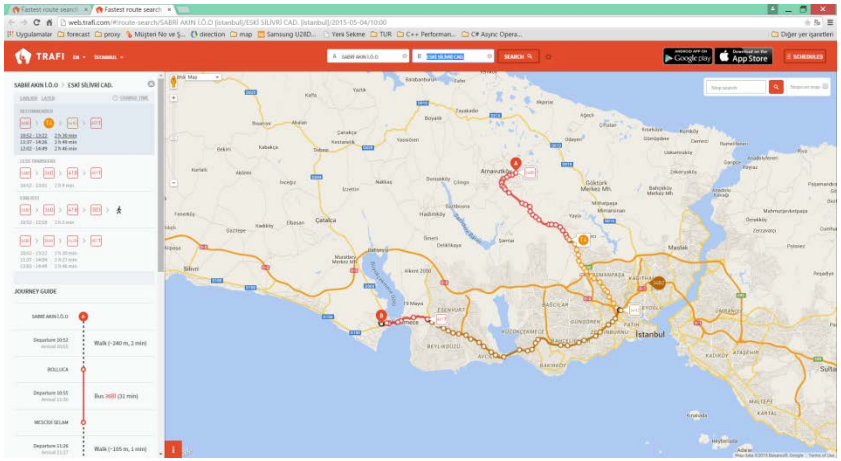


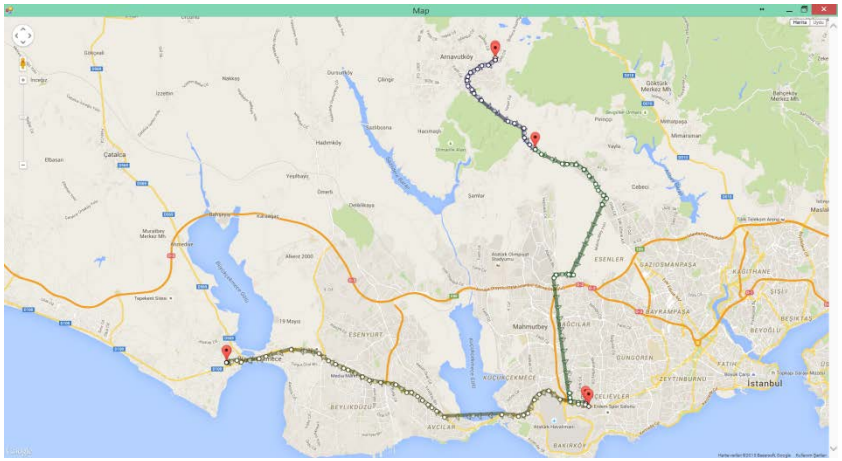


Trafi

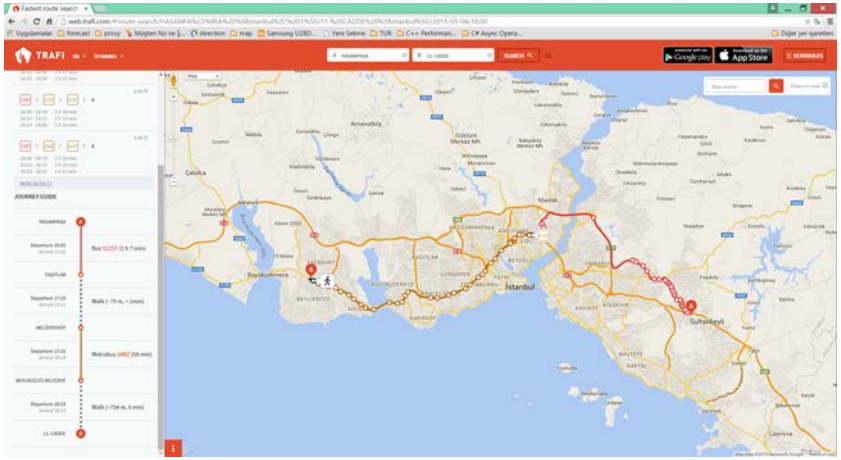
TUR

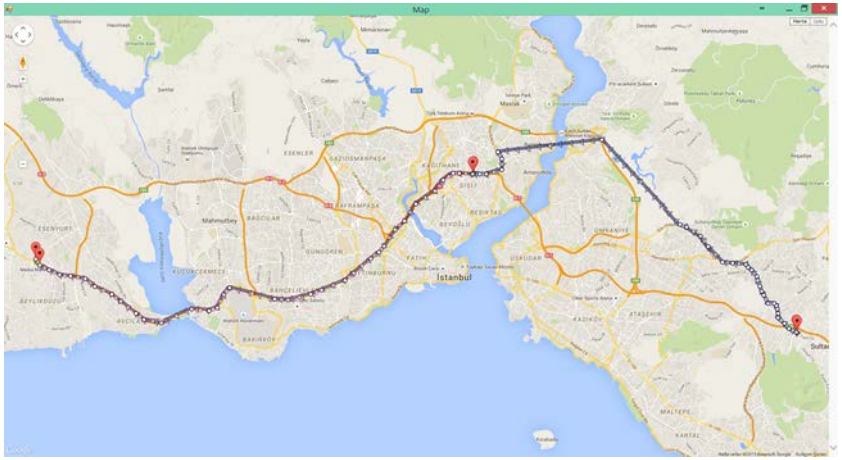
Şekil 78: İstanbul TUA'sı Yedinci Örnek Durak Çifti için Çözüm Haritaları





Şekil 79: İstanbul TUA'sı Sekizinci Örnek Durak Çifti için Çözüm Haritaları

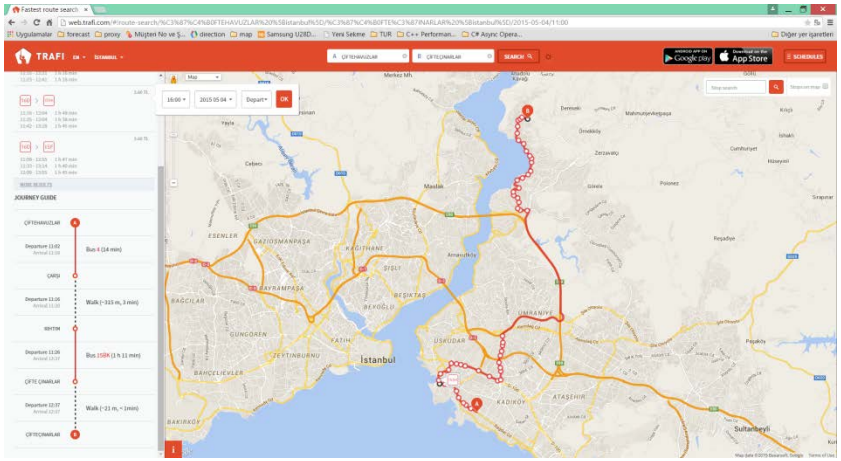


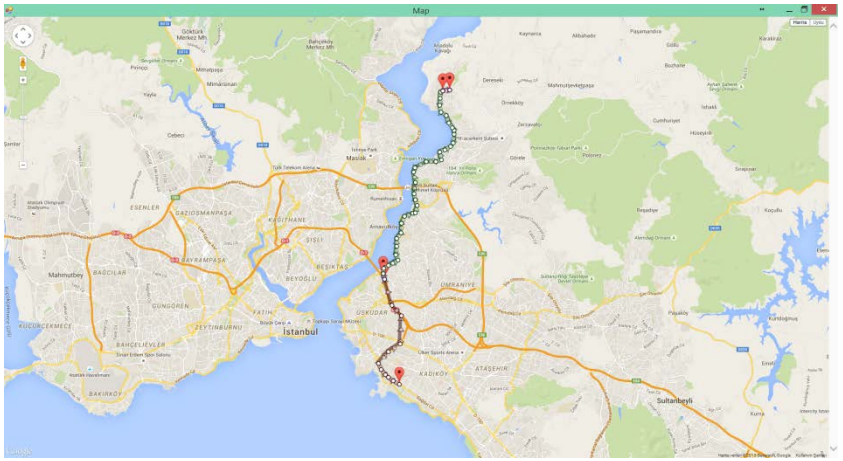


Trafi

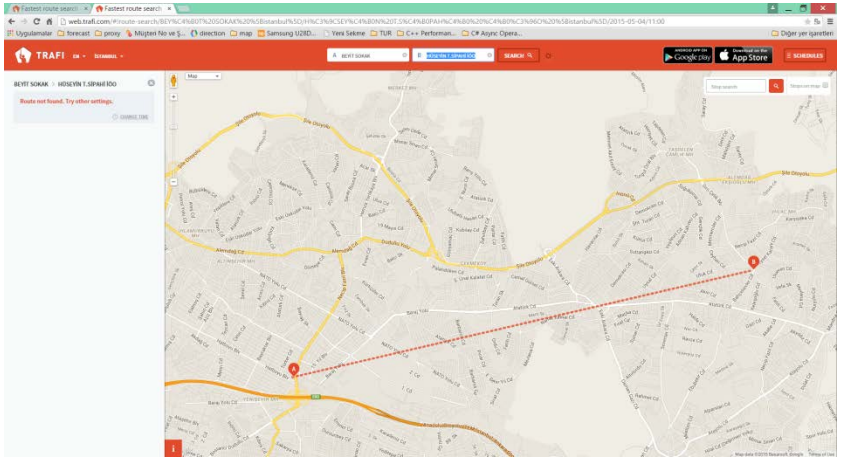
TUR

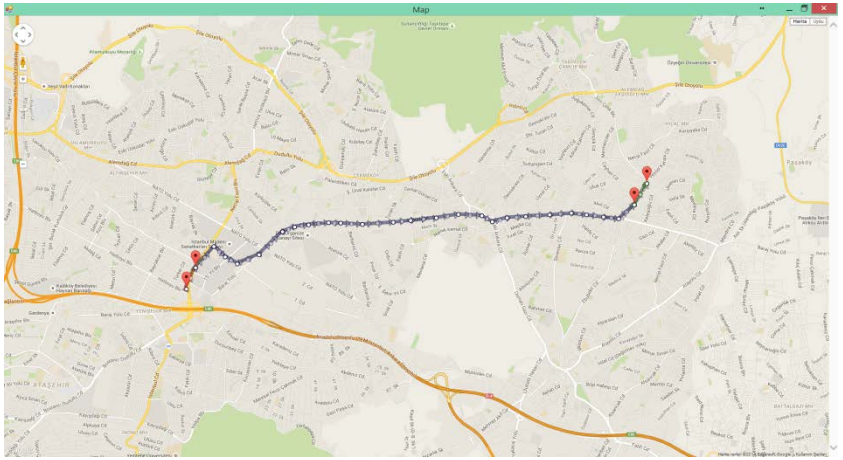
Şekil 80: İstanbul TUA'sı Dokuzuncu Örnek Durak Çifti için Çözüm Haritaları





Şekil 81: İstanbul TUA'sı Onuncu. Örnek Durak Çifti için Çözüm Haritaları





Trafi

TUR

Harita çizimlerinden (Şekil 42-81), aynı durak çifti için iki uygulama tarafından üretilen optimal çözüm güzergâhlarının birbirlerine çok benzedikleri görülmektedir. TUA verileri çok sık güncellendiği için zamanla uygulamaların kullandıkları veriler arasında küçük farklılıklar oluşabileceği dikkate alınırsa, aynı hatları kullanan çözüm güzergâhlarında küçük farklılıklar olması normal karşılanabilir.

Çözüm güzergâhlarının benzerliğine karşın, çözüm karşılaştırma tablolarında (Tablo 24, 26, 28, 30) TUR uygulaması çözümlerinin Trafi uygulamasına göre daha az yürüme mesafeleri içerdiği, açıkça göze çarpmaktadır. Trafi uygulaması, yürüme mesafesi sınırını daha yüksek tuttuğu için daha fazla yürüme kullanarak bazı durak çiftlerinde daha az aktarıma, bazılarında ise daha erken varış zamanına sahip çözümler üretebilmiştir. İstanbul TUA'sındaki 6 ve 8 numaralı durak çiftlerine ait çözümler (Tablo 30), yürüme mesafesini artırarak, aktarım sayısı ve varış zamanı açısından daha yüksek kalitede çözümlere ulaşılabileceğinin birer örneği olarak gösterilebilir.

Çözümlerin geneline bakıldığında ise TUR uygulamasının çoğu durak çifti için Trafi uygulamasına göre daha az aktarıma ve/veya daha erken varış zamanına sahip çözümler ürettiği görülmektedir. Ancak bu durum, TUR uygulaması çözümlerinin Trafi uygulaması çözümlerinden daha kaliteli olduğunu söylemek için yeterli değildir. Çünkü, iki optimizasyon algoritmasının ürettiği optimal çözümlerin sağlıklı bir şekilde karşılaştırılabilmesi için algoritmaların aynı verileri ve aynı amaç fonksiyonunu kullandığının bilinmesi gerekir. Trafi uygulamasının optimizasyon için kullandığı amaç fonksiyonu bilinmemektedir. Bununla birlikte, yürüme mesafesi sınırı düşük alınmasına rağmen TUR uygulamasının seçilen durak çiftlerinin çoğunda, Trafi uygulamasından daha yüksek kaliteli çözümler üretmiş olmasına dayanarak, uygulamanın genel anlamda makul ve yüksek kalitede çözümler üretebildiği rahatlıkla söylenebilir.

3.2. TUR Algoritmasının Literatürdeki Yöntemlerle Performans Karşılaştırması

TUA üzerinde yolculuk planlama problemine ilişkin özellikle son 5-6 yılda yapılan birçok çalışmada daha çok problemin çözüm algoritmaları üzerinde durulmuştur. Bu problem başlangıçta karayolları üzerinde güzergah hesaplama (navigasyon) problemi için

kullanılan algoritmaların TUA'ya uyarlanması ile çözülmeye çalışılmıştır. Ancak karayolları ağları TUA'lara göre çok daha büyük olmalarına rağmen, bir navigasyon hesabını nano saniyeler içinde hesaplayan algoritmalar, TUA'larda aynı performansı sergileyememiştir. Bu durum araştırmacıları yeni yaklaşımlar geliştirmeye sevk etmiştir. Özellikle son yıllarda geliştirilen algoritmalar, TUA'yı bir graf olarak işlemek yerine, ağı özel veri yapıları ile ele alarak hesaplamalarda önemli performans artışları sağlamışlardır. TUA'lar için oluşturulan özel veri yapıları, ileri düzey programla teknikleriyle birleştirilerek algoritmaların hesaplama süreleri milisaniyeler seviyesine düşürülebilmiştir.

Bast ve diğerleri (2014: 32) yaptıkları çalışmada son yıllarda geliştirilen algoritmaları, kullanılan optimizasyon kriterlerine göre gruplandırarak performanslarını karşılaştırmışlardır (Bast ve diğerleri, 2014: 32). Tablo 31'de orijinal haliyle verilmiş olan karşılaştırma tablosunda son sütun (time [ms]), 10000 rastgele durak çifti için ortalama hesaplama süresini milisaniye cinsinden göstermektedir[6]. Onuncu sütun (prep. [h]) ise ön işlem adımı içeren algoritmaların, ön işlem süresini saat olarak ifade etmektedir. Kullanılan ağdaki bağlantı sayısı dördüncü sütunda (conn. [10^6]) milyon adet olarak verilmiştir. Bu çalışmada Londra TUA'sı yürüyüş yolları sayısı ise 46 bin adet olarak ifade edilmiştir. Algoritmalar 2.6 GHz hızında Intel Xeon E5-2670 bir işlemcili ve DDR3-1600 MHz hızında 64GB Ram kapasiteli bir bilgisayarda koşturulmuştur. Ayrıca paralel işlem avantajlarının performans değerlerini etkilememesi için hesaplama işlemleri 16 çekirdekli işlemcinin tek çekirdeği üzerinden yürütülmüştür (Bast ve diğerleri 2014: 31).

Tablo 31'den görülebileceği gibi çoğu algoritmanın performans testleri Londra TUA'sı kullanılarak yapılmıştır. Bundan dolayı TUR algoritması da Londra Belediyesi veri paylaşım sitesinden[7] elde edilen Londra TUA verileri ile test edilmiştir. Algoritmanın performans ölçümü için kullanılan bilgisayar, 1.9 GHz hızında Intel Core i7 3517U bir işlemciye ve DDR3-1600 MHz veri transfer hızında 4 GB ram kapasitesine sahiptir.

[6] Diğer ayrıntılar için bknz. http://research.microsoft.com/pubs/207102/MSR-TR-2014-4.pdf
[7] http://data.london.gov.uk/

Tablo 31: Literatürdeki TUA Yolculuk Planlama Algoritmaları Performans Karşılaştırması

		INPUT		CRITERIA						QUERY		
algorithm	impl.	name	conn. [10^6]	arr.	tran.	rng.	fare	rel.	prep. [h]	comp. /stop	jn.	time [ms]
TE		London	5.1	●	○	○	○	○	–	50.6	0.9	44.8
TD	[79]	London	5.1	●	○	○	○	○	–	7.4	0.9	11.0
CH	[113]	Europe (LD)	1.7	●	○	○	○	○	<0.1	<0.1	n/a	0.3
CSA	[89]	London	4.9	●	○	○	○	○	–	26.6	n/a	2.0
ACSA	[215]	Germany	46.2	●	○	○	○	○	–	n/a	n/a	8.7
T. Patterns	[24]	Madrid	4.8	●	○	○	○	○	19	–	n/a	0.7
LD	[79]	London	5.1	●	●	○	○	○	–	15.6	1.8	28.7
MLS	[79]	London	5.1	●	●	○	○	○	–	23.7	1.8	50.0
RAPTOR	[79]	London	5.1	●	●	○	○	○	–	10.9	1.8	5.4
T. Patterns	[24]	Madrid	4.8	●	●	○	○	○	185	–	n/a	3.1
T. Patterns*	[22]	N. America	56.6	●	●	○	○	○	2304	–	n/a	12.0
CH	[113]	Europe (LD)	1.7	●	○	●	○	○	<0.1	<0.1	n/a	27.0
SPCS	[89]	London	4.9	●	○	●	○	○	–	372.5	98.2	843.0
CSA	[89]	London	4.9	●	○	●	○	○	–	436.9	98.2	161.0
ACSA	[215]	Germany	46.2	●	○	●	○	○	8	n/a	n/a	171.0
rRAPTOR	[89]	London	4.9	●	●	●	○	○	–	1634.0	203.4	922.0
CSA	[89]	London	4.9	●	●	●	○	○	–	3824.9	203.4	466.0
T. Patterns	[24]	Madrid	4.8	●	●	●	○	○	185	–	n/a	3.1
MLS	[79]	London	5.1	●	●	○	●	○	–	818.2	8.8	304.2
McRAPTOR	[79]	London	5.1	●	●	○	●	○	–	277.5	8.8	100.9
MLS	[79]	London	5.1	●	●	○	○	●	–	286.6	4.7	239.8
McRAPTOR	[79]	London	5.1	●	●	○	○	●	–	89.6	4.7	71.9

Kaynak: Bast ve diğerleri, 2014: 32

TUR algoritması Tablo 31'de çerçeve içerisinde gösterilen ikinci grup algoritmalarla aynı optimizasyon kriterlerini (varış zamanı ve aktarım sayısı) kullandığından performans karşılaştırması bu gruptaki algoritmalar ile yapılmıştır.

Performans ölçümü amacıyla rastlantısal olarak seçilen 10000 durak çifti için ard arda sorgulamalar yapılmış ve ortalama sorgu süresi hesaplanmıştır. Ölçüm için kullanılan Londra TUA'sı istatistiki verileri ve ölçüm sonuçları Tablo 32'de verilmiştir. Algoritmada en fazla 9 hatta kadar kullanan çözümler hesaplanmıştır.

Tablo 32: Londra TUA'sı İstatistiki Verileri

Durak Sayısı	19435
Yürüme Mesafeleri	44663
Hat Sayısı	698
Güzergâh Sayısı	2628
Güzergâh Durakları Sayısı	88839
Hareket Saati Sayısı	4860083
10000 Durak Çifti için Ortalama Sorgu Süresi	1,3433

Algoritmanın koşturulduğu bilgisayar, Tablo 31'deki algoritmaların test edildiği bilgisayardan çok daha düşük işlem kapasitesine sahip olmasına rağmen, performans ölçüm sonuçları, TUR algoritmasının tüm ön işlem adımsız algoritmalardan çok daha hızlı sonuç ürettiğini göstermektedir. Tabloda, birinci gruptaki ön işlem adımı içermeyen algoritmalar, sadece varış zamanını optimize etikleri halde TUR algoritmasından daha yüksek sorgu sürelerine sahiptirler.

Algoritmanın hesaplanan çözümlerde kullanılan hat sayısına göre ortalama sorgu süreleri, her bir TUA için ayrı ayrı hesaplanarak tablolar şeklinde (Tablo 33-37) aşağıda verilmiştir. Tablolarda hat sayısının sıfır (0) olduğu satırlar, seçilen durak çiftlerinin birbirine yürüme mesafesinde olduğunu göstermektedir. Hat sayısı ifade edilmeyen satırlar ise algoritmanın seçilen bu durak çiftleri arasında çözüm üretemediği anlamına gelmektedir. Bu durum durak çiftleri arasından 9 hatta kadar çözümün mevcut olmamasından kaynaklanmaktadır. Genel satırındaki değerler sırası ile toplam durak çifti sayısını ve sürelerinin ağırlıklı ortalama sorgu göstermektedir.

Algoritmaların performans değerlendirmelerinde sorgu süreleri kadar kullanılan hat sayıları da önem arz etmektedir. Örneğin Tablo 33'de 6'dan daha fazla hat kullanan çözüm olmaması, Londra TUA'sında iki durak arasında en fazla 6 hat ile seyahat edilebildiğini göstermektedir. İzmir TUA'sında (Tablo 35) ise rastgele seçilen durak çiftleri arasında 3'den daha fazla hat kullanan çözümlerin sayısı hızlı bir şekilde azalmaktadır. Ankara TUA'sında (Tablo 36) ise bu değer 2'dir. Aslında seçilen durak çifti sayısının belirgin bir şekilde azalmaya başladığı satırdaki hat sayısı, ilgili TUA'nın durakları arasındaki yolculuklar için kullanılması gereken hat sayılarının mod değeridir ve bu değer TUA'lar

için önemli bir parametredir. Bir algoritmanın ürettiği çözümlerde kullandığı ortalama hat sayısının bu değeri aşmaması istenir.

Ayrıca problemin çözüm uzayı hat sayısına bağlı olarak geometrik şekilde artmasına rağmen, ortalama sorgu süresinde artışın geometrik olmadığı görülmektedir. Aslında sorgu süresi, çözümde kullanılan hat sayısı kadar, bu hatların geçiş güzergâhlarındaki hat yoğunluğu ile de ilişkilidir. Tablo 37'de İstanbul TUA'sı için rastgele seçilen durak çiftlerinden bir tanesinin çözümü 5 hat içerdiği halde, sorgu süresinin ortalama sorgu süresinden küçük olduğu görülmektedir. Bu sonuç ancak iki durak arasındaki güzergâhı kullanan hat sayısının ilgili TUA ortalamasından daha az olması ile açıklanabilir.

Tablo 33: TUR Algoritması Londra TUA'sı Performans Değerleri

		Durak Çifti Sayısı	**Ortalama Süre**
Hat Sayısı	0	2	0,0027
	1	132	0,0264
	2	1624	0,3063
	3	4649	1,1742
	4	3212	2,0573
	5	343	2,3588
	6	10	2,4634
	-	28	1,1058
	Genel	**10000**	**1,3433**

Çözüm üretilemeyen durak çiftleri için ortalama sorgu sürelerinin düşüklüğü ise, böyle bir durumda TUR algoritmasının yalnızca birinci aşamasının işletilmesinden kaynaklanmaktadır. Çünkü iki durak çifti arasında verilen maksimum aktarım sayısında bir çözüm bulunmadığında, algoritmanın birinci aşamasında belirlenmeye çalışılan en az aktarım parametresi hesaplanamayacak ve ikinci aşamaya geçilemeyecektir. Tur algoritması birinci aşamada sadece güzergâh duraklarını işlemektedir. TUA'nın en çok satıra sahip verisi olan zaman çizelgeleri ise algoritmanın ikinci aşamada ele alınmaktadır. Bu sebeple verilen kısıtlara (en az katarım, en fazla yürüme mesafesi vb.) uygun çözümün bulunmadığı durak çiftleri için sorgulama işlemi ortalama bir sorgudan daha kısa sürmektedir.

Tablo 34: TUR Algoritması Bursa TUA'sı Performans Değerleri

		Durak Çifti Sayısı	Ortalama Süre
Hat Sayısı	0	40	0,0037
	1	2198	0,1023
	2	5532	0,5016
	3	1932	0,7422
	4	205	0,9085
	5	4	0,8110
	-	89	0,0460
	Genel	**10000**	**0,4628**

Tablo 35: TUR Algoritması İzmir TUA'sı Performans Değerleri

		Durak Çifti Sayısı	Ortalama Süre
Hat Sayısı	0	21	0,0033
	1	509	0,0442
	2	2182	0,2676
	3	4241	0,5797
	4	2222	0,7669
	5	713	0,8745
	6	94	0,9243
	7	18	0,9193
	Genel	**10000**	**0,5496**

Tablo 36: TUR Algoritması Ankara TUA'sı Performans Değerleri

		Durak Çifti Sayısı	Ortalama Süre
Hat Sayısı	0	21	0,0053
	1	1044	0,1400
	2	6190	0,6889
	3	2477	0,9803
	4	245	1,1917
	5	11	1,2102
	-	12	0,0033
	Genel	**10000**	**0,7144**

Dikkate değer diğer bir ayrıntı da TUR algoritmasının İstanbul TUA'sındaki performansının Londra'dakinden düşük olmasıdır (Tablo 37 ve 33). Aslında Londra TUA'sında bulunan hareket saati sayısı, İstanbul'dakinden yaklaşık %25 fazla olduğu için bu sonuç ilk bakışta şaşırtıcı gelebilir. Ancak, TUA'ların yürüme mesafeleri sayıları karşılaştırıldığında sonucun hiç de şaşırtıcı olmadığı anlaşılacaktır. Zira İstanbul TUA'sı, Londra'dan yaklaşık 4 kat daha fazla birbirlerine yürüme mesafesinde bulunan durak çiftine sahiptir.

Tablo 37: TUR Algoritması İstanbul TUA Performans Değerleri

		Durak Çifti Sayısı	**Ortalama Süre**
Hat Sayısı	0	11	0,0057
	1	1361	0,1955
	2	5827	2,1396
	3	2671	4,3395
	4	120	5,4365
	5	1	3,6195
	-	9	0,4112
	Genel	**10000**	**2,4984**

SONUÇ ve ÖNERİLER

Toplu ulaşım ağları (TUA) üzerinde yolculuk planlaması yapan yöntemler üzerine yapılan çalışmalar son yıllarda hızla artmaktadır. Çalışmalarda çoğunlukla, en kısa yol problemi çözümünde kullanılan yöntemlerin TUA'lar için uyarlanmış türevleri üzerinde durulmuştur. Ayrıca literatürde, mevcut algoritmaların karşılaştırmalı olarak ele alındığı çalışmalara da sıkça rastlanmaktadır. Bu alanda yapılan çalışmalarda TUA'lar, genellikle bir graf olarak ele alınmaktadır. Son zamanlarda geliştirilen yöntemler ise bu geleneksel yaklaşımdan farklı olarak, TUA verilerini özel veri yapıları ile işlemek suretiyle çözüm hesaplama performansının önemli ölçüde artmasına katkı sağlamışlardır (RAPOR, CSA gibi.).

Çok kriterli bir optimizasyon problemi olan TUA'lar üzerinde yolculuk planlama problemi, önceleri varış zamanına göre tek kriterli olarak ele alınsa da son yıllarda geliştirilen yöntemler, problemi çok kriterli olarak da çözebilmektedir.

Bu çalışmada varış zamanının yanı sıra aktarım sayısı kriterini de dikkate alan TUR adlı yeni bir algoritma geliştirilmiştir. Geliştirilen algoritma, literatürdeki mevcut uygulama ve diğer yöntemlerle karşılaştırılmıştır. Karşılaştırma işlemleri sonucunda TUR algoritmasının optimal çözümü, literatürdeki benzer algoritmalardan yaklaşık 4 kat daha hızlı hesaplayabildiği görülmüştür.

Toplu Ulaşım Ağları üzerinde yolculuk planlama problemine ilişkin çalışmaların çoğunda, algoritmaların ürettikleri çözümlerin kalitesi hakkında bilgi verilmemektedir. Bu durum, algoritmaların sağlıklı olarak karşılaştırılmasını zorlaştırmaktadır. Problemin doğası gereği, bir çözümün diğerlerine göre varış durağına daha erken ulaşması, o çözümün diğerlerinden daha iyi bir çözüm olduğunu söylemek için yeterli değildir. Çözümde kullanılan hat sayısının, çözümün uygulanabilirliğini önemli düzeyde etkileyeceği mutlaka dikkate alınmalıdır. Aksi takdirde, üretilen çözümlerin kullanılabilir olmaması muhtemeldir. Aktarım sayısının çözüm kalitesi açısından önemi, bu çalışmanın ikinci bölümünde ayrıntılı olarak ele alınmıştır.

Son yıllarda geliştirilen ve bu alanda performansı en yüksek yöntemlerden biri olan RAPTOR algoritmasının Londra TUA'sındaki test sonuçlarında, çözümlerin hat sayısı ortalaması 8,4 olarak verilmiştir (Delling ve diğerleri, 2013: 7). TUR algoritmasında ise bu değer yaklaşık 3,2'dir. Alandaki en yüksek performanslı algoritmalar arasında yer alan CSA'nın ise aktarım sayısını optimize etmediği bilinmektedir.

Geliştirilen algoritma, optimum çözümleri en az aktarım odaklı olarak hesapladığından, hat sayısı ortalaması bakımından en iyi çözümleri üretmektedir. Algoritma sorgu süresi bakımından da ön işlem adımı içermeyen algoritmalar arasında en yüksek performansa sahiptir.

TUR algoritma, varış zamanı kriteri açısından değerlendirildiğinde, hesaplanan çözümlerin varış zamanlarının genel itibarı ile makul olduğu söylenebilir. Bu algoritmaları, varış zamanı kriterine göre karşılaştırabilmek için, ürettikleri çözümlerin ayrıntılı olarak bilinmesi gerekir. Ancak literatürdeki çalışmaların çoğunda üretilen çözümlere ilişkin ayrıntılı bilgi verilmemiştir. Bu sebeple algoritmaların varış zamanı kriterine göre karşılaştırılması mümkün olmamaktadır. Ancak bu noktada TUR algoritması, sadece en az aktarım içeren çözümleri hesapladığından, daha fazla hat kullanarak elde edilebilecek çözümleri dikkate almamaktadır. Çalışmanın ikinci bölümünde de açıklandığı gibi algoritma daha fazla aktarımları da hesaplayacak şekilde kolayca düzenlenebilir. Ancak testlerde algoritmanın bu amaçla düzenlenmiş bir türevinin kullanılmamış olmasının temel sebebi, problemin çok kriterli bir optimizasyon problemi olması dolayısıyla, daha fazla aktarım içeren çözümlerin aktarım sayısı az olan çözümlere tercih edilip edilmeyeceğinin belirsiz olmasıdır. Ayrıca algoritma çözümleri Trafi uygulaması çözümleri ile karşılaştırıldığında, en az aktarım yaklaşımı ile elde edilen çözümlerin çoğu durumda varış zamanına göre de optimum olduğu gösterilmiştir. Buna ek olarak bir çözümde kullanılacak hat sayısı arttığında, önceki hatlardaki gecikme vb. sebeplere bağlı olarak çözümün sonraki hatları için kullanılabilecek hareket saatlerinin değişeceği gözden kaçırılmamalıdır. Böyle bir durumda çözüm kalitesi önemli oranda azalabilir hatta çözüm tamamen kullanılamaz hale gelebilir.

TUA'larda yolculuk planlama probleminin çok kriterli olarak ele alındığı çalışmalarda, optimizasyon işlemi için pareto optimal çözümler hesaplanmaktadır. Bu

yaklaşıma göre, bir çözümün alternatif çözümlere tercih edilebilmesi için o çözümün ele alınan optimizasyon kriterlerinden ez az birine göre diğerlerinin çözümlerden daha iyi olması gerekirken; çözümlerin diğer kriterlere göre eşit olması gerekir. Bu kural, çözümlerden biri daha erken varış zamanına ancak daha çok aktarıma sahip olduğu zaman, tercih yapılmasını zorlaştırmaktadır. TUR algoritması en az aktarım odaklı olması sebebiyle, böyle durumlarda aktarım sayısı daha az olan çözümü tercih etmektedir. Problemde optimal çözümlerin hesaplanması için farklı bir yaklaşım olarak, optimizasyon kriterlerinin önceliklerine göre ağırlıklandırılmış bir amaç fonksiyonu da kullanılabilir.

RAPTOR algoritmasında çözümler genişlik öncelikli olarak hesaplanmaktadır. Böylece algoritma, her adımda kullanılacak hatları yalnızca bir kez işlemektedir. Ancak Şekil 33'de gösterildiği gibi parabolik güzergâha sahip bir hattın, çözümün mümkün olan ilk aktarımında değil; bir sonraki aktarımında çözüme dâhil edilmesi, varış zamanını azaltacaktır. Bu sebeple TUR algoritması, hattı mümkün olan tüm durumlarda çözüme sokarak, optimum çözümü araştırmaktadır. Aslında, TUR algoritması da RAPTOR gibi genişlik öncelikli arama yapmaktadır. Fakat çözümün önceki adımlarında kullanılabilen bir hat, sonraki adımlarda da çözüme dâhil edilebilmektedir.

TUR algoritmasının yüksek performansla çözümler üretebilmesini sağlayan temel etken, algoritmanın birinci aşamasını oluşturan ters akım yöntemidir. Yöntemin, problemin çözüm uzayını önemli önemli ölçüde daralttığı, çalışmanın ikinci bölümünde ayrıntılı olarak ifade edilmiştir. TUA'nın sadece güzergâh durakları verilerini işleyen ters akım yöntemi, çok daha büyük olan hareket saati verisini işlemeden önce, durak çifti arasındaki mümkün olan tüm yolları etiketleyerek algoritmanın toplam performansına önemli katkı sağlamaktadır. Ancak genişlik öncelikli arama yerine derinlik öncelikli aramanın kullanılması durumunda, dinamik programlamaya dayalı bu aşamada, problem uzayının daha hızlı bir şekilde daraltılabileceği düşünülmektedir. Genişlik öncelikli arama işlemi bir çözümün tüm hatlarının kalkış ve varış saatlerini ancak son adımda hesaplayabilmektedir. Derinlik öncelikli arama yöntemi ise ilk adımda bir aday çözüm üretebilmektedir ve algoritmanın sonraki adımlarında bu çözümün tüm duraklara varış saatleri bilinebilmektedir. Böylece optimal çözüme ulaşamayacağı kesin olarak bilinen çözümlerin daha etkin bir biçimde elenebilmesi mümkün olabilir. Ayrıca literatürde bu tür algoritmalarla birlikte kullanılan A* gibi hızlandırma yöntemlerinin, derinlik öncelikli

olarak işleyen algoritmalara uyarlanması daha kolaydır. Bu çerçevede, TUR algoritmasının derinlik öncelikli bir türevinin hızlandırma algoritmalarıyla hibrit olarak kullanılması fikri, alandaki gelecek dönem çalışmaları için iyi bir motivasyon noktası olarak önerilebilir. Ayrıca algoritmanın paralel programlama teknikleri kullanılarak hızlandırılmasının mümkün olup olmadığı da başka bir dikkate değer araştırma konusudur. Ancak bu alan daha çok bilgisayar bilimleri ile alakalı olduğundan, konunun bu boyutuna çalışmada değinilmemiştir.

YARARLANILAN KAYNAKLAR

Ahuja, Ravindra K. ve diğerleri (1990), "Faster Algorithms For The Shortest Path Problem", **Journal of the Association for Computing Machinery**, 37(2), 213-223.

Aljazzar, Husain ve Leue, Stefan (2011), "K∗: A Heuristic Search Algorithm For Finding The K Shortest Paths", **Artificial Intelligence**, 175(2011), 2129-2154.

Antsfeld, Leonid ve Walsh, Toby (2012a), "Finding Optimal Paths In Multi-Modal Public Transportation Networks Using Hub Nodes And TRANSIT Algorithm", Lutz Frommberger, Kerstin Schill, Bernd Scholz-Reiter (Ed.), **3rd Workshop on Artificial Intelligence and Logistics** *içinde* (7-11), http://www.sfbtr8.uni-bremen.de/papers/SFB_TR8Rep031-08_2012.pdf (25.05.2015).

Antsfeld, Leonid ve Walsh, Toby (2012b), "Finding Multi-Criteria Optimal Paths In Multi-Modal Public Transportation Networks Using The Transit Algorithm.", **Intelligent Transport Systems World Congress 2012**, http://www.nicta.com.au/pub-download/full/5920 (04.05.2015).

Artigues, Christian ve diğerleri (2013), "State-Based Accelerations and Bidirectional Search for Bi-Objective Multi-Modal Shortest Paths", **Transportation Research Part C: Emerging Technologies**, 27(2013), 233-259.

Arz, Julian ve diğerleri (2013), "Transit Node Routing Reconsidered", Vincenzo Bonifaci, Camil Demetrescu, Alberto Marchetti-Spaccamela (Ed.), **Lecture Notes In Computer Science** *içinde* (55-66), New York: Springer, http://arxiv.org/pdf/1302.5611.pdf (04.05.2015).

Bang-Jensen, Jorgen ve Gutin, Gregory (2007), **Digraphs Theory, Algorithms and Applications**, Newyork: Springer.

Bannister, Michael J. ve Eppstein, David (2011), "Randomized Speedup of the Bellman-Ford Algorithm", http://siam.omnibooksonline.com/2012ANALCO/data/papers/005.pdf (04.05.2015).

Barrett, Chris ve diğerleri (2006), "Implementations of Routing Algorithms for Transportation Networks", **9th DIMACS Implementation Challenge: The Shortest Path Problem**, http://www.dis.uniroma1.it/challenge9/papers/barrett.pdf (04.05.2015).

Bast, Hannah ve diğerleri (2014), "Route Planning in Transportation Networks", **Microsoft Research - Technical Report (MSR-TR-2014-4)**, http://research.microsoft.com/pubs/207102/MSR-TR-2014-4.pdf (04.05.2015).

Bast, Holger ve diğerleri (2006), "TRANSIT - Ultrafast Shortest-Path Queries with Linear-Time Preprocessing", **9th DIMACS Implementation Challenge: The Shortest Path Problem**, http://people.mpi-inf.mpg.de/~dmatijev/papers/DIMACS06.pdf (04.05.2015).

Bast, Hannah ve Storandt, Sabine (2014), Flow-Based Guidebook Routing, Catherine C. McGeoch ve Ulrich Meyer (Ed.), **16th Workshop on Algorithm Engineering and Experiments** *içinde* (155-165), New York: Springer, http://epubs.siam.org/doi/book/ 10.1137/1.9781611973198 (04.05.2015).

Batz G., Veit ve diğerleri (2008), "Time Dependent Contraction Hierarchies - Basic Algorithmic Ideas", http://algo2.iti.kit.edu/documents/tdch.pdf (04.05.2015).

Bauer, Reinhard ve Delling, Daniel (2009), "SHARC: Fast and Robust Unidirectional Routing", **Journal of Experimental Algorithmics**, 14(2009), 1-27, http://i11www.iti.uni-karlsruhe.de/extra/publications/bd-sharc-09.pdf (18.05.2015).

Bauer Reinhard ve diğerleri (2010), "Preprocessing Speed-Up Techniques Is Hard", **Algorithms and Complexity**, 6078, 359-370, http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/1978704 (18.05.2015).

Berger, Annabell ve diğerleri (2010), "Fully Dynamic Speed-Up Techniques for Multi-criteria Shortest Path Searches in Time-Dependent Networks", **Experimental Algorithms**, 6049, New York: Springer.

Bielli, Maurizio ve diğerleri (2006), "Object Modeling and Path Computation for Multimodal Travel Systems", **European Journal of Operational Research**, 175 (2006), 1705–1730,

Bondy, J. A. ve Murty, U. S. R. (1982), **Graf Theory with Applications**, Fifth Edition, New York: North-Holland.

Booth, Joel ve diğerleri (2009), "A Data Model for Trip Planning in Multimodal Transportation Systems", Kesten Martin ve diğerleri (Ed.), **12th International Conference on Extending Database Technology: Advances in Database Technology** *içinde* 995-1005, Newyork: ACM Newyork.

Brandes, Ulrik ve diğerleri (2004), "Generating Node Coordinates for Shortest-Path Computations in Transportation Networks", **Journal of Experimental Algorithmics**, 9(2004), 1-16.

Brodal, Gerth Stølting ve Jacob, Riko (2003), "Time-Dependent Networks as Models to Achieve Fast Exact Time-Table Queries", **Electronic Notes in Theoretical Computer Science**, 92(2003) 1-12.

Chen, Huey-Kuo ve Chou, Cheng-Yi (1999), "Comparisons of Yen's and A Modified Generalized Floyd T-Shortest Path Algorithms", **Journal of the Eastern Asia Society for Transportation Studies**, 3(6), 233-246.

Chen, Mo ve diğerleri (2007), "Priority Queues and Dijkstra's Algorithm", **UTCS Technical Report (TR-07-54)**, http://www.cs.utexas.edu/ftp/techreports/tr07-54.pdf (18.05.2015).

Cherkassky, Boris V. ve diğerleri (1996), "Shortest Paths Algorithms: Theory and Experimental Evaluation", **Mathematical Programming**, 73 (1996), 129-174.

Columbus, Tobias (2012), **Search Space Size in Contraction Hierarchies**, Yayımlanmamış Yükseklisans Tezi, Karlsruhe Institute of Technology, Informatics Institute of Theoretical Computer Science.

Cormen, Thomas H. ve diğerleri (2009), **Introduction to Algorithms**, 3rd Edition, Cambridge: The MIT Press.

Delling, Daniel (2008), "Time-Dependent SHARC-Routing", Dan Halperin, Kurt Mehlhorn (Ed.) , **Algorithms - ESA 2008**, New York: Springer.

Delling, Daniel (2009), **Engineering and Augmenting Route Planning Algorithms**, Yayımlanmamış Doktora Tezi, Universitat Fridericiana zu Karlsruhe, Fakultat fur Informatik.

Delling, Daniel ve diğerleri (2013), "Round-Based Public Transit Routing", **14th Meeting on Algorithm Engineering and Experiments**, http://research.microsoft.com/pubs/156567/raptor_alenex.pdf (18.05.2015).

Delling, Daniel ve Wagner, Dorothea (2009), "Time-Dependent Route Planning", Ravindra K. Ahuja, Rolf H. Möhring, Christos D. Zaroliagis (Ed.), **Robust and Online Large-Scale Optimization** *içinde* 5856, New York: Springer.

Dibbelt, Julian ve diğerleri (2013), "Intriguingly Simple and Fast Transit Routing", **Lecture Notes in Computer Science**, 9057, New York: Springer.

Diestel, Reinhard (2005), **Graph Theory**, New York: Springer

Dijkstra, E. W., (1959), "A Note on Two Problems in Connexion with Graphs", **Numerische Mathematik**, l(1959), 269-271.

Eppstein, David (1997), "Finding the K Shortest Paths", **SIAM Journal of Computing**, 28(2), 652–673 https://www.ics.uci.edu/~eppstein/pubs/Epp-SJC-98.pdf (21.05. 2015).

Fan, Lang ve Mumford, Christine (2010), "A Metaheuristic Approach to the Urban Transit Routing Problem", **Journal of Heurisric**, 16(3), 353-372.

Geisberger, Robert (2008), **Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks**, Yayımlanmamış Yüksek Lisans Tezi, Universität Karlsruhe, Institut für Theoretische Informatik.

Geisberger, Robert ve diğerleri (2008), "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks", **Lecture Notes in Computer Science**, 9057, New York: Springer.

Goldberg, Andrew V. ve Harrelson, Chris (2004), "Computing the Shortest Path: A* Search Meets Graph Theory", **Microsoft Research-Technical Report (MSR-TR-2004-24)**, http://research.microsoft.com/pubs/154937/soda05.pdf (21.05.2015).

Goldberg, Andrew V. ve Werneck, Renato F. (2005), "Computing Point-to-Point Shortest Paths from External Memory", **SIAM Workshop on Algorithms Engineering and Experimentation**, http://www.avglab.com/andrew/pub/alanex05.pdf (21.05. 2015).

Goldberg, Andrew V. ve diğerleri (2007a), "Reach for A*: Ecient Point-to-Point Shortest Path Algorithms", **SIAM Workshop on Algorithms Engineering and Experimentation**, http://research.microsoft.com/pubs/60764/ tr-2005-132.pdf (25. 05.2015).

Goldberg, Anrew V. ve diğerleri (2007b), "Better Lanndmsrks Within Reach", Camil Demetrescu (Ed.), **Experimental Algorithms**, 4525 *içinde* (38-51), New York: Springer.

Gutman, Ron (2004), "Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks", **Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithmics and Combinatorics**, http://www.siam.org/meetings/alenex04/ abstacts/rgutman1.pdf (21.05.2015).

Harju, Tero (2011), **Lecture Notes on Graph Theory**, http://cs.bme.hu/fcs/graphtheory.pdf (21.05.2015).

Hart, Peter E. ve diğerleri (1968), "A Formal Basis for the Heristic Determinationof Minimum Cost Path", **IEEE Transaction of System Science and Sibernetics**, 4(2), 100-107.

Huguet, Marie-Jose ve diğerleri (2013), "Eficient algorithms for the 2-Way Multi Modal Shortest Path Problem", **International Network Optimization Conference** https://ddv.ull.es/users/inoc2013/public/Proceedings/ENDM-072.pdf (21.05.2015).

Pohl, Ira (1969), **Bi-Directional And Heuristic Search In Path Problems**, Standford: Standford University, http://www.slac.stanford.edu/pubs/slacreports/reports04/slac-r-104.pdf (21.05.215).

Jariyasunant, Jerald ve diğerleri (2010), "Algorithm for Finding Optimal Paths in A Public Transit Network with Real-Time Data", http://www.baytripper.org/TRB [RESUBMIT].pdf (21.05.2015).

Joyner, David ve diğerleri (2013), **Algorithmic Graph Theory and Sage**, https://graphbook.googlecode.com/files/latest-r1991.pdf (25.05.2015).

Kalpana, R. ve Thambidurai, P. (2010), "Combining Speedup Techniques based on Landmarks and Containers", **International Journal on Computer Science and Engineering**, 2(6), 2212-2222.

Koszelew, Jolanta (2008), "Two Methods of Quasi-Optimal Routes Generation in Public Transportation Network", **7th Computer Information Systems and Industrial Management Applications**, http://jolantakoszelew.pl/attachments/File/Two_Methods_of_Quasi-Optimal_Routes_Generation.pdf (25.05.2015).

Köhler, Ekkehard ve diğerleri (2006), "Fast Point-to-Point Shortest Path Computations with Arc-Flags", **9th DIMACS Implementation Challenge**, http://www.dis.uniroma1.it/challenge9/papers/kohler.pdf (25.05.2015).

Kumari, S.Meena ve Geethanjali, N. (2010), "A Survey on Shortest Path Routing Algorithms for Public Transport Travel", **Global Journal of Computer Science and Technology**, 9(5), 73-76.

Li, Jianfu (2012), "A Improved Yen Algorithm Based on A*", **Journal of Computational Information Systems**, 8(21), 9017-9024.

Li ,Jing-Quan ve diğerleri (2012), "A Multimodal Trip Planning System Incorporating the Park-and-Ride Mode and Real-time Traffic/Transit Information", **Journal of Intelligent Transportation Systems**, 16(2), 1-9.

Liebig, Thomas ve diğerleri (2014), "Predictive Trip Planning – Smart Routing in Smart Cities", **EDBT/ICDT 2014 Joint Conference**, 1133, 331-338.

Liu, Lu (2010), **Data Model and Algorithms for Multimodal Route Planning with Transportation Networks**, Yayımlanmamış Doktora Tezi, Technischen Universität München, Fakultät für Bauingenieur-und Vermessungswesen.

Martinek, Vladislav, ve Michal Zemlicka (2009), "Speeding Up Shortest Path Search in Public Transport Networks", http://ceur-ws.org/Vol-471/paper1.pdf (25.05.2015).

Meng, Foo Hee ve diğerleri (1999), "A Multi-Criteria, Multi-Modal Passenger Route Advisory System", http://www.mysmu.edu/faculty/hclau/IES-CTR-99.pdf (25.05. 2015).

Merrifield, Timothy (2010), **Heuristic Route Search in Public Transportation Networks,** Yayımlanlmamış Yüksek Lisans Tezi, University of Illinois, Computer Science.

Müller-Hannemann, Matthias ve diğerleri (2006), "Timetable Information: Models and Algorithms", http://i11www.iti.uni-karlsruhe.de/extra/publications/mswz-tima-06.pdf (25.05.2015).

Nannicini, Giacomo ve diğerleri (2010), “Bidirectional A* Search on Time-Dependent Road Networks”, http://www.lix.polytechnique.fr/~liberti/bidirtimedepj.pdf (25.05. 2015).

Nosrati, Masoud ve diğerleri (2012), “Investigation of the * (Star) Search Algorithms: Characteristics, Methods and Approaches”, **World Applied Programming**, 2(4), 251-256.

Pyrga, Evangelia ve diğerleri (2008), “Efficient Models for Timetable Information in Public Transportation Systems”, **ACM Journal of Experimental Algorithmics**, 12(2008), 1-39.

Rios, Luis Henrique Oliveira ve Chaimowicz, Luiz (2009), “PNBA*: A Parallel Bidirectional Heuristic Search Algorithm”, **Communications in Computer and Information Science**, 28(2009), 73-84, http://homepages.dcc.ufmg.br/~chaimo/ public/ENIA11.pdf (25.05.2015).

Ruohonen, Keijo (2013), **Graph Theory**, (Çev. Janne Tamminen, Kung-Chung Lee and Robert Piché), http://math.tut.fi/~ruohonen/GT_English.pdf (25.05.2015).

Sanders, Peter ve Schultes, Dominik (2007), “Engineering Fast Route Planning Algorithms”, **6th Workshop on Experimental Algorithms**, http://algo2.iti.kit.edu/documents/routeplanning/weaOverview.pdf (25.05.2015).

Sanders, Peter ve Schultes, Dominik (2006), “Robust, Almost Constant Time Shortest-Path Queries in Road Networks”, **9th Dimacs Implementation Challenge**, http://algo2.iti.kit.edu/schultes/hwy/hhTransit.pdf (25.05.2015).

Sanders, Peter ve Schultes, Dominik (2005), “Highway Hierarchies Hasten Exact Shortest Path Queries”, **13th European Symposium on Algorithms**, 3669, 568-597, http://algo2.iti.kit.edu/schultes/hwy/esaHwyHierarchies.pdf (25.05.2015).

Schultes, Dominik (2005), **Highway Hierarchies Hasten Exact Shortest Path Queries**, Yayımlanmamış Yüksek Lisans Tezi, University of Saarlandes, Computer Sciences.

Schultes, Dominik (2008), **Route Planning in Road Networks**, Yayımlanmamış Doktora Tezi, Universität Fridericiana zu Karlsruhe, Fakultät für Informatik.

Schultes, Dominik ve Sanders, Peter (2007), "Dynamic Highway-Node Routing", **6th Workshop on Experimental Algorithms**, 4525, 66-79, http://algo2.iti.kit.edu/schultes/hwy/dynamic.pdf (25.05.2015).

Schulz, Frank ve diğerleri (2000), "Dijkstra's Algorithm On-Line: An Empirical Case Study from Public Railroad Transport", **Journal of Experimental Algorithmics**, 5(12), 1-22, http://i11www.iti.uni-karlsruhe.de/extra/publications/sww-daole-00.pdf (25.05.2015).

Sen, Sandeep (2013), **Lecture Notes for Algorithm Analysis and Design**, Iindian Institute of Technology: New Delhi, http://www.cse.iitd.ernet.in/~ssen/csl356/root.pdf (25.05.2015).

Shu-Xi, Wang (2012), "The Improved Dijkstra's Shortest Path Algorithm and Its Application", **Procedia Engineering**, 29(2012), 1186-1190.

Van Steen, Maarten (2010), **An Introduction to Graph Theory and Complex Networks**, http://www.di.unipi.it/~ricci/book-watermarked.pdf (25.05.2015).

Strasser, Ben ve Wagner, Dorothea (2014), "Connection Scan Accelerated", **Sixteenth Workshop on Algorithm Engineering and Experiments**, http://epubs.siam.org/doi/pdf/10.1137/1.9781611973198.12 (25.05.2015).

Thorup, Mikkel (2000), "On Ram Prioirty Queuies", **SIAM Journal on Computing**, 30(1), 86-109, https://static.aminer.org/pdf/PDF/000/611/317/on_ram_priority_queues.pdf (25.05. 2015).

Voloshin, Vitaly I. (2009), **Introduction To Graph Theory**, New York: Nova Science Publishers.

Wagner, Dorothea ve diğerleri (2003), "Dynamic Shortest Paths Containers", **Electronic Notes in Theoretical Computer Science**, 92(1), 1-19.

Wagner, Dorothea ve Willhalm, Thomas (2007), "Speed-Up Techniques for Shortest-Path Computations", **24th International Symposium on Theoretical Aspects of Computer Science**, 4393, 23-26.

Wagner, Dorothea ve diğerleri (2005), "Geometric Containers for Efficient Shortest-Path Computation", A**CM Journal of Experimental Algorithmics**, 10, 1-30.

Wu, Lingkun ve diğerleri (2012), S"hortest Path and Distance Queries on Road Networks: An Experimental Evaluation", **Very Large Data Base Endowment**, 5(5), 406-417.

Wu, Qıujın ve Hartley, Joanna (2004), "Using K-Shortest Paths Algorithms to Accommodate User Preferences in the Optimization of Public Transport Travel", **Applications of Advanced Technologies in Transportation Engineering**, 1, 181-186, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.456&rep=rep1&type=pdf (25.05.2015).

Yen, Jin Y. (1971), "Finding the K Shortest Loopless Paths in a Network", **Management Science**, 17(11), 712-716.

Zhang, Jianwei ve diğerleri (2012), "A Multimodal Transport Network Model for Advanced Traveler Information System", **Journal of Ubiquitous Systems & Pervasive Networks**, 4(1), 21-27.

Zhao, Liang ve diğerleri (2008), "A* Algorithm for the time-dependent shortest path problem", **11th Japan Korea Joint Wokshop on Algorithms and Computations**, http://www.gsais.kyoto-u.ac.jp/members/liang/research/waac08.pdf (25.05.2015).

ÖZGEÇMİŞ

1977 yılında Trabzon'da doğdu. İlk ve orta ve lise öğrenimini Trabzon'da tamamladıktan sonra 1997 yılında Karadeniz Teknik Üniversitesi, Fen Edebiyat Fakültesi, İstatistik ve Bilgisayar Bilimleri Bölümünü kazanarak lisans eğitimine başladı. 2001 yılında bu bölümden mezun olarak 2002 yılında Karadeniz Teknik Üniversitesi, Enformatik Bölümü'nde öğretim görevlisi olarak çalışmaya başladı. 2007 yılında çalıştığı bölüm ders kitabı olarak Temel Bilgisayar ve Uygulamalı İleri Excel kitaplarında yazar olarak görev aldı. 2009 yılında Karadeniz Teknik Üniversitesi Sosyal Bilimler Enstitüsü Ekonometri Ana Bilim Dalında yüksek lisans öğrenimini tamamlayarak aynı yıl Recep Tayyip Erdoğan Üniversitesi Fındıklı Meslek Yüksekokulu'na öğretim görevlisi olarak atandı. Evli ve iki çocuk babasıdır.

Ali AKAY