T.C.
KIRIKKALE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DEMİRYOLU TAŞIMACILIĞI İÇİN EN KISA YOL VE MAKSİMUM AKIŞ OPTİMİZASYONU

BEKİR KESKİN
ENDÜSTRİ MÜHENDİSLİĞİ ANA BİLİM DALI

YÜKSEK LİSANS TEZİ

DANIŞMAN
Doç. Dr. Evrencan ÖZCAN

KIRIKKALE-2023

T.C.
KIRIKKALE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DEMİRYOLU TAŞIMACILIĞI İÇİN EN KISA YOL VE MAKSİMUM AKIŞ OPTİMİZASYONU

BEKİR KESKİN
ENDÜSTRİ MÜHENDİSLİĞİ ANA BİLİM DALI

YÜKSEK LİSANS TEZİ

DANIŞMAN
Doç. Dr. Evrencan ÖZCAN

KIRIKKALE-2023

KABUL ve ONAY

Bekir KESKİN tarafından hazırlanan "DEMİRYOLU TAŞIMACILIĞI İÇİN EN KISA YOL VE MAKSİMUM AKIŞ OPTİMİZASYONU" adlı aşağıdaki jüri tarafından OY BİRLİĞİ ile Kırıkkale Üniversitesi Fen Bilimleri Enstitüsü Endüstri Mühendisliği Anabilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman: Doç. Dr. Evrencan ÖZCAN

İmza

Endüstri Mühendisliği Anabilim Dalı, Kırıkkale Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

Başkan: Doç. Dr. Ercüment Neşet DİZDAR

İmza

İş Sağlığı ve Güvenliği Anabilim Dalı, Çankırı Karatekin Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

Üye: Prof Dr. Tamer EREN

İmza

Endüstri Mühendisliği Anabilim Dalı, Kırıkkale Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum.

Tez Savunma Tarihi: 12/01/2023

Jüri tarafından kabul edilen bu tezin Yüksek Lisans Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum.

Prof. Dr. Recep ÇALIN

Fen Bilimleri Enstitü Müdürü

ETİK BEYANI

Kırıkkale Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etikkurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlakkurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarakkaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimibeyan ederim.

Bekir KESKİN

12.01.2023

ÖZET

DEMİRYOLU TAŞIMACILIĞI İÇİN EN KISA YOL VE MAKSİMUM AKIŞ OPTİMİZASYONU

KESKİN, Bekir

Kırıkkale Üniversitesi

Fen Bilimleri Enstitüsü

Endüstri Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi

Danışman: Doç. Dr. Evrencan ÖZCAN

Ocak 2023, 56 sayfa

Son yıllarda sanayide yaşanan hızlı gelişmeler beraberinde ham madde, yarı mamul, ürün ve stokların akışını hızlandırmıştır. Ayrıca üretim girdi maliyetlerinin artması geri dönüşüm olgusunun gelişmesine öncülük etmiş böylece ürün ve atıkların akışı da hızlanarak taşıma faaliyetleri küresel boyutta hareketlilik kazanmıştır. Söz konusu bu akış beraberinde küresel lojistik sektörünün büyümesine neden olmuştur. Günden güne gelişen lojistik sektörü, hamule akışını kontrol etmekte ve ihtiyacı karşılamakta zorlanmıştır. İşte bu küresel yük taşımacılığını kontrol etmeyi ve ihtiyacı karşılamayı kolaylaştırmak için alternatif çözümler gündeme gelmiştir. Bunlardan bir tanesi de taşıma modları arasından demiryolu taşımacılığını geliştirme düşüncesidir. Bu noktada da tek kuşak-tek yol sloganı ortaya çıkmıştır. Tek kuşak tek yol projesi küresel boyutta bir proje olmakla birlikte oldukça geniş bir alana sahiptir. Proje üç ayaklı olarak Kuzey Koridoru, Orta Koridor ve Güney Koridoru olarak planlanmıştır. Modern ipek yolu projesiyle orta koridor kullanılarak Asya ve Avrupa arasında kısa, güvenli, ekonomik ve risk faktörlerinin minimize edileceği kombine taşımacılık yapılması planlanmıştır. Söz konusu bu hat kesimi için taşıma projeksiyonuna bakıldığında ilk etapta yıllık bir milyon yolcu ve altı buçuk milyon ton yük taşıma kapasitesi ile, 2034 yılı sonunda üç milyon yolcu ve on yedi milyon ton yük taşınması planlanmaktadır.

Güney Koridoru üzerinde "Evergreen" olarak anılan, "Ever Given" adlı geminin Süveyş Kanalı'nda karaya oturması, gemi trafiğini durdurması ile dünya deniz ticaretine sadece bir haftalık süre içinde milyarlarca dolar zarar vermesi, Rusya üzerinden geçen Kuzey koridoru üzerinde yaşanan Rusya-Ukrayna savaşı gibi olumsuz gelişmelerden dolayı orta koridorlara ilişkin demiryolu şebekesinin Ahılkelek-Kapıkule hat kesimi için şebeke optimizasyonu yapılması amaçlanmıştır.

Bu itibarla yüklerin demiryolu ile Türkiye üzerinden Asya-Avrupa geçişinde Ahılkelek-Kapıkule arasındaki demiryolu ağı şebeke olarak seçilmiş, güzergâh üzerinde önemli istasyonlar şebekenin düğümleri olarak belirlenmiştir. Bu seçimler ile 29 düğüm ve 34 daldan oluşan bir şebeke elde edilmiştir.

Bu çalışmada farklı olarak Floyd-Warshall algoritması en kısa yol hesaplaması için Python'da kodlanarak çözülmüş olup, ayrıca söz konusu demiryolu ağı üzerinde hem mevcut hat kapasitesine hem de gelecek projeksiyonuna göre maksimum akış modeli oluşturulmuştur. Seçilen en kısa yol ile maksimum akış modelleri kıyaslama yapılarak ortaya çıkan sonuçlar yorumlanmıştır.

Anahtar Kelimeler: Şebeke optimizasyonu, Demiryolu taşımacılığı, En kısa yol, Floyd-Warshall algoritması, maksimum akış modeli, Ford-Fulkerson algoritması

ABSTRACT

SHORTEST PATH MAXIMUM FLOW OPTIMIZATION FOR RAIL TRANSPORTATION

KESKİN, Bekir

Kırıkkale University

Graduate School of Natural and Applied Sciences

Department of Industrial Engineering, Master Science Thesis

Supervisor: Assoc. Prof. Dr. Evrencan ÖZCAN

February 2023, 56 pages

In recent years, rapid developments in the industry have accelerated the flow of raw materials, semi-finished products, products and stocks. In addition, production has led to the increase in input costs and the development of the phenomenon of recycling, so that the flow of products and wastes has accelerated and transportation activities have gained mobility on a global scale. The flow of these products and wastes has led to the growth of the global logistics industry. The logistics sector, which is developing day by day, has had difficulties in controlling the flow of freight and meeting the need. And alternative solutions have come to the fore in order to control this global freight transport and make it easier to meet the need. One of these solutions is the idea of developing rail transport among transport modes. At this point, the slogan of "One Belt, One Road" emerged. The "One Belt, One Road" project, besides being a global project, also has a very wide area. The project is planned as three legs, these are the north corridor, middle corridor and south corridor. And what is planned with the modern silk road project is to make combined transportation between Asia and Europe by using the middle corridor, which will be short, safe, economical and risk factors will be minimized. Considering the transportation projection for this line separation, it is planned to carry three million passengers and seventeen million tons of load by the end of 2034, with an annual capacity of one million passengers and six and a half million tons of load.

The ship named "Ever Given", known as "Evergreen" on the south corridor, ran aground in the Suez Canal and stopped the ship traffic, causing billions of dollars of damage to the world maritime trade in just one week and it was aimed to work on the network optimization and maximum flow of the line for the Ahılkelek-Kapikule line section of the railway network related to the middle corridor.

In this respect, the railway network between Ahılkelek-Kapıkule was chosen as the network in the transit of loads via Turkey through Asia-Europe, and important stations on the route were determined as the nodes of the network. With these selections, a network consisting of 29 nodes and 34 twigs was obtained.

In this study, differently, the Floyd-Warshall algorithm was coded in Python for the shortest path calculation, and a maximum flow model was created on the railway network in question according to both the current line capacity and the future

projection. And by comparing the selected shortest path and maximum flow models, the results were interpreted.

Key Words: Network optimization, Rail transport, Shortest path, Floyd-Warshall algorithm, maximum flow model, Ford-Fulkerson algorithm

TEŞEKKÜR

Yüksek lisans eğitimim boyunca tez hazırlık sürecinde sürekli motive eden, cesaret veren ve tezimin hazırlanmasında tüm tecrübesi ile bana rehberlik eden ve inanan tez danışman hocam Sayın Doç. Dr. Evrencan ÖZCAN beyefendiye kalbi şükranlarımı sunarım.

Çalışma hayatımda olduğu gibi eğitim çalışmalarımda da yardımını esirgemeyen değerli meslektaşlarıma ayrıca desteğini, sevgisini ve duasını hiçbir zaman eksik etmeyen kıymetli aileme teşekkür ederim.

İÇİNDEKİLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

ŞEKİLLER DİZİNİ

ŞEKİL

Sayfa

KISALTMALAR DİZİNİ

AB	Avrupa Birliği
AHP	Analytic Hierarhy Proses (Analitik Hiyerarşi Prosesi)
ANP	Analytic Network Proses (Analitik Ağ Prosesi)
BTK	Bakü, Tiflis, Kars Demiryolu
CBS	Coğrafi Bilgi Sistemi
ÇKKV	Çok Kriterli Karar Verme
ELECTRE	Elemination and Choice Translating Reality English
HES	Hidroelektrik Santrali
TOPSIS	Technique for Order of Preference by Similarity to Ideal Solution
VİKOR	VIseKriterijumsa Optimizacija (Çok Kriterli Optimizasyon ve Uzlaşık Çözüm)

1. GİRİŞ

Ulaşım ülkelerin ekonomik gücünü belirleyen unsurların en başında gelmektedir. Ekonomi ekosisteminin kılcal damarlarını oluşturan ulaştırma sektörü refah ve kalkınmanın en önemli unsurlarındandır. Geçmişten günümüze insanoğlunun ulaşım ve taşıma ihtiyacı sürekli artarak süregelmiştir. Tekerleğin icadı ile başlayan gelişim süreci ve yenilikler kara araçları, deniz araçları, hava araçları ve demiryolu araçları ile günümüzde de devam etmektedir. Bu gelişmeler ulaşımı ve taşımayı kolaylaştırmış ve en kısa yoldan en ucuza ulaşma talebini de beraberinde getirmiştir. Dolayısıyla lojistik faaliyetler, birçok alana dahil olan etkin konumuyla optimizasyon çalışmalarının sürekli merkezinde yer almıştır [1].

Çin-Avrupa arasındaki uluslararası ulaşım koridorlarına baktığımızda Rusya'nın içinde yer aldığı Kuzey Koridoru, ülkemizin de içinde bulunduğu Orta Koridoru ve Süveyş Kanalı üzerinden geçiş yapılan denizyolu güzergâhı Güney Koridoru karşımıza çıkmaktadır [2]. Çin-Avrupa arasındaki demiryolu yük taşımacılığında Orta Koridorun ön plana çıkması Bakü Tiflis Kars (BTK) Demiryolu Hattı'nın hizmete girmesiyle ortaya çıkmıştır. Bununla beraber Rusya-Ukrayna savaşı ve Süveyş Kanalında yaşanan krizle birlikte uluslararası ulaşım koridorları dikkate alındığında, diğer ulaşım koridorları için Orta Koridor güçlü bir alternatif haline gelmeye başlamıştır [3]. Çin'den Avrupa'ya doğru demiryolu ile yola çıkan bir tren, Orta Koridoru tercih etmesi halinde yedi bin kilometreyi on iki günde kat etmektedir. Yük treninin Rusya üzerinden Kuzey Koridoru'nu tercih etmesi halinde on bin kilometrelik bir mesafeyi kat etme durumu ve en az on beş günlük seyir süresi söz konusu olmaktadır. Söz konusu yükler için Güney Koridoru'nun tercih edilmesi halinde ise deniz yoluyla Süveyş Kanalı üzerinden yirmi bin kilometrelik yol ancak elli gün civarında kat edilerek Avrupa'ya ulaşılabilmektedir. Bu veriler göz önüne alındığında Orta Koridorun küresel ticarette, Ukrayna-Rusya savaşı ile Süveyş kanalında yaşanan aksaklıklar da dikkate alındığında ne kadar avantajlı olduğu anlaşılmaktadır.

Ancak Bakü-Tiflis-Kars (BTK) demiryolu hattında taşımacılığa başlanmasıyla Çin-Avrupa-Çin arasındaki demiryolu yük trafiğinde Orta Koridorun etkinliği ortaya çıkmaya başlamıştır. Orta Koridor üzerinden, Çin'den Çekya'ya varmak için yola çıkan Çin-Avrupa arasındaki ilk blok transit konteyner treni Ankara'ya 6 Kasım 2019'da ulaşmıştır. Bu taşımada Marmaray Tüp Geçidi'nden geçen tren, Çin-Türkiye parkurunu 12 günde, 11.483 km'lik toplam Çin-Çekya parkurunu da 18 günde tamamlamıştır. Çin'den Avrupa'ya giden bu tren, Şekil 1.1'de gösterilen Bakü-Tiflis-Kars Demir İpek Yolu üzerinden Avrupa'ya ulaşan ilk yük treni olmuştur [4].

Şekil 1.1. Bakü-Tiflis-Kars demiryolu haritası

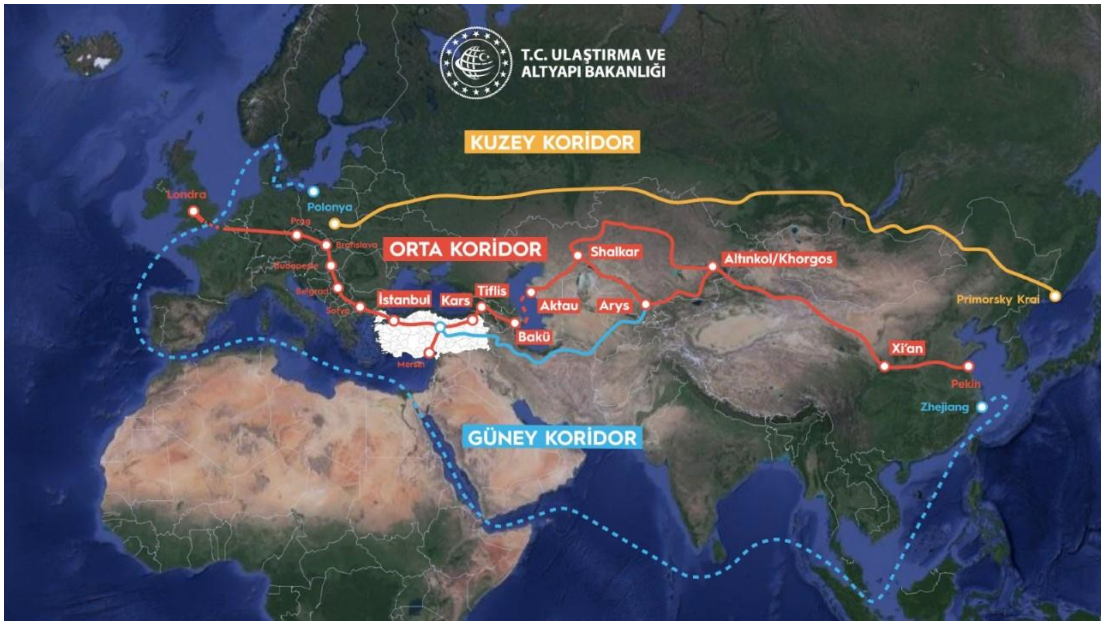
Çin'den trenlere yüklenen konteynerler Orta Koridor ve BTK üzerinden Marmaray geçişini kullanarak Avrupa'da farklı ülkelere taşınmaktadır. Özellikle yapımı devam eden Çerkezköy-Kapıkule hızlı tren standardında demiryolu hattının 2024 yılında bitirilmesi planlanmaktadır. Çerkezköy-Kapıkule arası demiryolu hattının hizmete açılması ile Bulgaristan kapılarına yüklerin demiryoluyla daha hızlı taşıması sağlanmış olacaktır.

Süveyş Kanalı üzerinde The Ever Given gemisi ile yaşanan tedarik sıkıntısı ve kuzey koridorunda devam eden Rusya-Ukrayna savaşı orta koridor üzerindeki talebi artırmaktadır. Son yıllarda artan ulaşım ve taşıma taleplerine yönelik alternatif rota eğilimleri ile birlikte uluslararası yük taşımasında Orta koridor üzerinde yaşanan tehir ve aksaklıkların giderilmesi gibi yukarıda sayılan söz konusu faktörler de göz önüne alındığında Avrupa ile Çin arasında kesintisiz ulaşımı sağlayan söz konusu güzergâh üzerinde optimizasyon çalışmalarının gerekliliği ve ihtiyacı da ortaya çıkmaktadır.

2. ULAŞTIRMA KORİDORLARI

2.1.Bir Kuşak Bir Yol Projesi – Orta Koridor

"Tek Kuşak Tek Yol" sloganı ile dile getirilen Yeni İpek Yolu Projesi; Rusya'nın içinde yer aldığı Kuzey Koridoru, ülkemizin de içinde bulunduğu Orta Koridoru ve Süveyş Kanalı üzerinden geçiş yapılan denizyolu güzergâhı olan Güney Koridorudur [5].



Şekil 2.1. Bir kuşak bir yol projesi haritası

Tez kapsamında son yıllarda giderek etkinliği artan ve milyar dolarları bulan yatırımlarla küresel demiryolu taşımacılığı ve modern ipek yolunun Türkiye parkuru ele alınmış, Çin-Avrupa arasında demiryolu ağında en uygun güzergâhın bulunması amacıyla şebeke optimizasyon çalışması yapılmıştır. İpek yolunu Demir İpek Yolu olarak yeniden canlandırmak için Avrupa'yı Türkiye, Gürcistan (Tiflis) Azerbaycan (Bakü) ve Kazakistan üzerinden Çin'e ulaştıran Kars-Tiflis-Bakü Demiryolunda; 2018 yılında 143 trenle 136 bin ton, 2019 yılında 185 trenle 194 bin ton ve 2020 yılında 288 trenle 371 bin ton olmak üzere toplam 616 trenle 701 bin ton taşıma gerçekleşmiştir [6]. İlk etapta yıllık bir milyon yolcu ve altı buçuk milyon ton yük taşıma kapasitesi ile, 2034 yılı sonunda üç milyon yolcu ve on yedi milyon ton yük taşınması planlanmaktadır. Görüldüğü üzere, Pekin ile Londra'yı demiryoluyla birbirine bağlayan doğu-batı demiryolu güzergâhının tam merkezinde yer alan hat kesiminde çalışma yapılacaktır. Çin'den yola çıkarak Orta Koridor, Bakü-Tiflis-Kars demiryolu

hattı ve Marmaray'ı kullanan yük trenleri, Çin-Türkiye arasını 12 günde, Çin- Avrupa (Çekya) arasını ise 18 günlük sürede kat etmektedir [5]. Uygulamada, Ahılkelek-Kapıkule arasında demiryolu şebekesi ele alınmış ve güzergâh üzerindeki önemli istasyonlar şebekenin düğümü olarak değerlendirilmiştir. Öncelikle düğümler arasındaki ilişkileri değerlendirmek ve en kısa yolu bulmak için yol optimizasyon problemlerinde sıklıkla tercih edilen Floyd-Warshall algoritması kullanılmış ve Python [7] programlama dilinde yazılarak çözülmüştür. Sonrasında ise maksimum akış Ford-Fulkerson [8] algoritması kullanılmıştır. Ahılkelek-Kapıkule arasında demiryolu ile yapılacak olan yük taşıması için en kısa yol algoritması kullanılarak ideal güzergâh belirlenmiştir. Belirlenen ideal güzergâhın maksimum akış yönünden günümüz ve gelecek hat kapasitesi göz önüne alınarak kıyaslaması yapılmıştır. Böylelikle Asya-Avrupa arasında Türkiye üzerinden Ahılkelek-Kapıkule hattında yapılacak demiryolu taşımacılığında en kısa yolun maksimum akış için de uygunluğu karşılaştırılmıştır. Böylesine kritik bir coğrafyada, coğrafi konumunun beraberinde getirmiş olduğu avantajı ve lojistik üs haline gelme potansiyeli ile demiryolu hatlarının verimliliği göz önüne alınarak doğu-batı ve tersi istikametinde uluslararası taşıma koridoru üzerinde en uygun güzergâh seçimi için farklı bir bakış açısı sunulmuştur.

3. LİTERATÜR TARAMASI

Bu tez çalışması kapsamında konu edilen araştırmalar ile ilgili yapılan çalışmaların özeti ve yorumlanması bu kısımda yer almaktadır. En kısa yol çalışması, maksimum akış çalışması ve atama çalışmalarının öncüleri Dantzing, Ford ve Fulkerson' dur [9, 10]. En kısa yol çalışması, maksimum akış çalışması ve atama çalışmalarında bir noktadan başka bir noktaya mümkün olan en etkin şekilde iletim sağlanması istenilmektedir. Şebeke iletim problemleri olarak adlandırılan bu çeşit çalışmalarda çözüm için matematiksel modellemeler geliştirilmiştir [10, 11, 12].

Bu çalışmada şebeke akışı çalışmalarında öncelikli bir yere sahip olan Ford ve Fulkerson [8] Flows in Network ile Ahuja ve arkadaşlarının [11] Network Flows kitapları incelenmiş ve temel kaynak olarak yararlanılmıştır.

3.1.En Kısa Yol Algoritmaları Kullanılarak Yapılan Çalışmalar

En kısa yol algoritmalarında amaç, şebeke içinde belirlenen bir başlangıç noktasından herhangi bir hedef noktaya giden en kısa yolu bulmaktır. En kısa yol problemlerinin çözümünde Floyd-Warshall Algoritması, Bellman-Ford, Dijkstra ve Johnson Algoritmaları sıkılıkla kullanılanlar arasındadır. Bu algoritmalar arasından da Floyd-Warshall Algoritması ile Dijkstra algoritması ön plana çıkmaktadır. Dijkstra algoritması, negatif kenarların bulunmadığı grafikle çalışan bir algoritmadır. Bunlarla beraber Floyd-Warshall Algoritması Dijkstra algoritmasından daha geneldir [11, 13]. Floyd-Warshall algoritması, döngü içermeyen bir şebekede, negatif veya pozitif ağırlıklı olmasına bakılmaksızın optimal sonuç vermektedir. Böylelikle şebekedeki herhangi iki düğüm arasındaki en kısa yol belirlenir [14, 15]. Ayrıca Johnson algoritması [16] da en kısa yol probleminin çözümünde kullanılan algoritmadır.

Çalışmada şebekedeki herhangi iki düğüm arasındaki en kısa yolun belirlemesi açısından Floyd-Warshall Algoritması tercih edilmiştir. En kısa yol problemlerinin çözümünde kullanılan Floyd-Warshall Algoritması literatürdeki çalışmalar incelendiğinde enerji hatları, ulaşım, telekomünikasyon, bilgisayar ağları ve sağlık başta olmak üzere birçok farklı alanda karşımıza çıkmaktadır.

Demiryolu alanında; Keskin ve Özcan, demiryolu sektöründe yaptıkları çalışma ile ulusal demiryolu ağı için toplam 33 düğümden oluşan bir şebeke oluşturmuş ve en kısa

yolun hesaplanması için Floyd-Warshall algoritması Python programlama dilinde kodlanarak çözülmüş ve toplam on dokuz adet lojistik merkez arasında en kısa yol/yollar tespit edilmiştir [17]. Wang ve Lu, Romanya - Polonya arasındaki demiryolu taşımacılığının optimize edilmesi için istasyonların teknik özelliklerini dikkate alarak doğrusal programlama modelini kullanmışlardır [18]. Pandey ve Dixit ise yaptıkları çalışmada demiryollarında daha güvenilir ve daha sürdürülebilir ulaşım sağlamak hedefiyle istasyonlar arası mesafenin optimize edilmesi uygulamasının çözümünde Dijkstra algoritmasını kullanmışlardır [19]. Liu ve arkadaşları, üç boyutlu bilgisayar modelleme sistemlerini en kısa yol algoritmaları aracılığıyla kullanarak demiryolu ulaştırmasında rota seçimi yapmışlar ve görselleştirmişlerdir [20]. Wang ve arkadaşları ise yaptıkları çalışmada yüksek hızlı demiryolu hattı inşası için yol sayısını maksimum düzeyde tutma, seyahat süresini en aza indirme ve uygun duraklar belirleme amaçlarıyla doğrusal programlama ve genetik algoritma yöntemini kullanışlardır [21]. Zhang ve arkadaşları, demiryolu ile tehlikeli madde taşımasında en uygun güzergâhın belirlenmesi amacıyla inşaat maliyeti ve güvenlik faktörlerini göz önüne alarak sezgisel tabanlı bir rota algoritması olan A* algoritmasını kullanmışlardır [22]. Kosjier ve arkadaşları, Inđija ve Novi Sad şehirleri arası demiryolun güzergâhında rota planlaması için maliyet, kapasite, fiziksel etkiler, gelişim ve yaşam ortamı üzerindeki etki kriterleri altında dört adet alternatif için çok kriterli karar verme yöntemlerinden VIKOR yöntemini kullanmışlardır [23]. Kankavi çalışmasında, büyük projeleri de dikkate alarak Türkiye geçişinde maliyet, süre, yük potansiyeli, güvenlik riski kriterlerini ele almış ve AHP yöntemi kullanarak en uygun güzergâhı belirlemiştir [24]. Saat ve Serrano ise yine çok kriterli karar verme yöntemlerini kullanarak Malezya'da yüksek hızlı trenlerin güzergâh seçim problemi için 3 kriter ve 11 alternatif ile yapmış oldukları çalışmada ELECTRE yönteminden faydalanmışlardır [25]. Özdemir çalışmasında en kısa yol algoritmalarını bir arada kullanarak Çin-Avrupa arasında yapılacak yük taşıması için demiryolunda şebeke optimizasyonu yapmış, çalışmada en kısa yol algoritmalarını Python programlama dili ile kodlayarak optimal güzergâhı hesaplamıştır [26]. Dermawan ise çalışmasında bir tren yolculuğunda en iyi yolu bulmada en kısa yol algoritmalarından Dijkstra ve Floyd-Warshall algoritmalarının karşılaştırmasını yapmıştır [27]. Özdemir ve arkadaşları, yine demiryolu alanında yapılan bu çalışmada ise ipek yolu koridorunda Pekin ile Londra arasında yer alan demiryolu ağı için toplam 26 düğümden oluşan bir şebeke

üzerinde en kısa yolun hesaplanmasını Dijkstra algoritmasını kodlayarak çözmüşlerdir [28].

Diğer alanlarda ise; Pradhan ve Mahinthakumar karayolunda yaptıkları çalışmada büyük ölçekli bir ulaşım ağında değişen trafik koşullarını dikkate alarak tüm çiftlerin en kısa yolunu bulmak için kullanılan ve iki önemli algoritma olan Floyd-Warshall ve Dijkstra yöntemlerinin performans analizini yapmışlardır [29]. Hamurcu ve Eren ise yine karayolunda yaptıkları çalışmada ÇKKV yöntemlerini kullanarak Ankara'da yeni bir ulaştırma seçeneği olan monoray sistemi için güzergâh seçimini yapmışlardır. Belirlenen sekiz alternatif monoray güzergâhı arasından en uygun güzergâhı ANP ve TOPSIS yöntemi kullanarak tespit etmişlerdir [30]. Hanzl ve arkadaşları ise en kısa yol bulma yöntemlerini tercih etmişler ve Çek Cumhuriyeti'nin Güney Bohemian bölgesinde on altı düğümlü ulaşım ağının trafik modelinin belirlenmesi konulu çalışmada Floyd algoritmasını kullanmışlardır [31]. Yanwei ve arkadaşları ise yine en kısa yol bulma yöntemlerini tercih etmişler ve yaşanılan ulaşım sorunlarından dolayı optimal bir çözüm yolu bulmak için Floyd algoritmasını kullanarak probleme çözüm getirmişlerdir [32]. Pandika ve arkadaşları etkili seyahat planlaması için bir bölgedeki tıkanıklığın ulaşımda aksamalara neden olmasından dolayı karayolu üzerinde oluşan tıkanıklığı önlemek için en uygun rotayı oluşturabilen bir uygulama geliştirmişlerdir [33]. Risald ve arkadaşları Dijkstra ve Floyd-Warshall algoritmasının bir arada kullanılması ile kaza yerinden en yakın hastaneye en kısa sürede gidilmesi için güzergâh geliştirmişlerdir [34]. Triana ve Syahputri ihtiyaca göre gerekli en yakın garajı bulmak için medya bilgilerinden yararlanılan ve Floyd-Warshall yöntemini kullanan bir uygulama ile en yakın garaj konumunu bulmuşlardır [35]. Tang ve arkadaşları yeniden üretime dayalı entegre (ileri ve tersine) lojistikte optimal yolun belirlenmesi için Floyd tabanlı bir çözüm sunarak lojistik alanında optimizasyon çalışması yapmışlardır [36]. Danışan ve arkadaşları ise bu çalışmada diğer çalışmalardan farklı olarak Floyd-Warshall algoritmasını C tabanlı bir kod yardımıyla çözümleyerek, Türkiye'deki elektrik enerjisi talebinin yaklaşık %31,9'unu karşılayan on iki Hidroelektrik Santral (HES)'lerin bakımında görevlendirilen ekiplerin santrallara en kısa yoldan ulaşmaları için Floyd-Warshall algoritmasını kullanarak optimal sonuç veren en kısa yolları bulmuşlardır [37]. Ramadhan ve arkadaşları ise Floyd-Warshall ve Prim algoritmalarının kıyaslamasını sunarak en kısa yol problemini ele almışlardır [38]. Çakır ve arkadaşları çalışmalarında CBS tabanlı rota ve güzergâh

optimizasyonu ile on mahalleye ait güzergâh tespiti için Dijkstra algoritmasını kullanmışlardır [39]. Sungkwan ve arkadaşları araçların enerji tüketiminin en aza indirilmesini için en kısa yoldan çalışma süresinin iyileştirilmesi amacıyla yaptıkları çalışmada 12 düğümden oluşan şebekede Dijkstra algoritmasını kullanmışlardır [40]. Ekmen, en kısa yol probleminde kullanılan algoritmalardan Dijkstra, Bellman-Ford, Johnson ve Floyd-Warshall algoritmalarının en kısa yolu bulma performansları için kıyaslama yapmıştır [41]. Magzhan ve Jani en kısa yolun hesaplanmasında en yaygın kullanılan algoritmalardan olan Dijkstra, Floyd-Warshall, Bellman-Ford ve Genetik algoritmasını değerlendirmiştir. Algoritmaları java yazılım diliyle kodlamış ve bu şekilde hesaplamıştır. Çalışmada en kısa yolun hesaplanmasında, daha iyi sonuca ulaşmak adına yapay zekâ, bulanık mantık ve sinir ağları gibi tekniklerin de kullanılabileceği söylenmiştir [42]. Tamimi ve Abu-Ryash ise Dijkstra algoritması, Bellman-Ford algoritması, Floyd-Warshall algoritması ve Johnson algoritması gibi en kısa yol algoritmalarını incelemişlerdir [43]. Golden, yaptığı çalışmada Dijkstra ve Bellman-Ford algoritmalarını 50-1000 düğüm arası şebeke probleminde çalışma zamanları verilerine göre değerlendirmiştir [44]. Aydın ve Alkan, çalışmalarında 3 farklı güzergâh için yol bulma algoritması test edilmiş ve elde edilen sonuçlar Dijkstra, Bellman-Ford, Floyd-Warshall, A* algoritmasında karşılaştırmışlardır [45]. Djojo ve Karyono çalışmalarında Dijkstra algoritması, A* algoritması ve Floyd-Warshall'ı süre, bellek ve hesaplama sonucu kriterleri altında karşılaştırmışlardır [46]. Wang, en kısa yol algoritmalarından Dijikstra, Bellam-Ford ve Floyd Warshall algoritmalarının temel uygulama alanlarını ile çalışma prensiplerini anlatmıştır [47]. Çam ve Sezen, Türkiye'de şehirlerarası yolcu taşıyan bir firmanın işlerinin daha iyi yönetilmesi amacıyla oluşturulan uygulamada, araçların bekleme sürelerinin en aza indirilmesi amaçlanmıştır [48]. Timor, taşıma modlarında hat uzunluğu ve erişilebilirlik hedefleri doğrultusunda maliyetlerin minimize edilmesini sağlayan bir model geliştirmişlerdir [49]. Söyler ve Fendoğlu, çalışmalarında Malatya Büyükşehir Belediyesi ilaçlama araçlarının optimal rotaları, Hierholzer & Floyd Warshall algoritmaları ve Excel-Solver ile hesaplanmış, sonuçlar karşılaştırılmıştır [50]. Nuriyeva ve Kızılateş, gezgin satıcı problemi için n adet düğüm ele alınmış "en kısa yol" ve "ekleme sezgiseli" algoritmaları kullanılarak bulunan devre tüm düğümlerden geçecek şekilde tasarlanmıştır [51]. Saçar, çalışmasında Hicaz demiryolu ile Bağdat demiryolu hattını analiz etmiştir [52]. Özdemir ve arkadaşları, lojistik merkez yatırım projesi

önceliklendirme problemi için yatırım aşamasındaki 6 adet lojistik merkezi arasından seçim yapmışlar, uygulamada AHP ve TOPSIS yöntemlerini kullanmışlardır [53].

3.2.Maksimum Akış Problemleri Kullanılarak Yapılan Çalışmalar

Ford ve Fulkerson, iki şehri bir dizi ara şehir yoluyla birbirine bağlayan bir demiryolu ağında, ağın her bağlantısının kapasitesini temsil eden bir numara atamış, sabit durum koşulu varsayarak, verilen bir şehirden diğerine maksimum akışı bulmuştur [9]. Abdullah ve Hua, trafik sıkışıklığından meydana gelen ulaşım sorununun çözümü için en kısa yol maksimum akış algoritmalarını kullanmışlardır. En kısa yol Dijkstra Algoritması ile belirlenmiş daha sonra maksimum akış ve en kısa yol problemi doğrusal programlama kullanılarak formüle edilerek Microsoft Excel'de excel çözücü kullanılarak çözülmüştür [54]. Rajalakshmi ve Vaidyanathan, Trafik Yönetim Sistemi için şebekedeki maksimum akışı bulmayı amaçlamışlardır [55]. Luo ve arkadaşları teorik bir tedarik zinciri modeli biçiminde sentezlemek için grafik teorisi mantığını kullanmışlardır [56]. Jia ve Zhang, ağ bağlantılı bir altyapı bakım planlaması ve iş gücü yönlendirmesinin ortak optimizasyonu için bütünsel amaçlı optimizasyon yaklaşımı önermişlerdir [57]. Surakhi ve arkadaşları belirlenen bir grafikte maksimum akışı bulmak için paralel bir Genetik algoritma uygulanmıştır [58]. Paithankar ve Chatterjee, maden ocakları için yeni bir üretim çizelgeleme yöntemi sunarak maden ocağı üretim programı çalışmasında maksimum akış ve genetik algoritmayı bir arada kullanan hibrit yöntem geliştirmişlerdir [59]. Kyi ve Naingsu, Myanmar'da su dağıtım boru hattı ağındaki maksimum akışı hesaplamak için grafik teorisinde iyi bilinen Ford-Fulkerson algoritması kullanılmıştır [60]. Jiang ve arkadaşları çalışmalarında Ford-Fulkerson algoritmasının simülasyon sonucu iyi bir performansa sahip olduğunu göstermişler [61]. Kyi ve arkadaşları Myanmar'ın Kyaukse bölgesi'nde elektrik dağıtım şebekesinin maksimum akışı dikkate alan bir ağ üzerinde kaynak düğüm s'den hedef düğüm t'ye mümkün olan maksimum akış için Ford-Fulkerson algoritması ile elektrik dağıtım şebekesinde matematiksel tahmin yapmışlardır [62]. Bulut ve Özcan tarafından yapılan çalışmada ise Ford-Fulkerson algoritmasını kullanılarak elektrik şebekelerinde maksimum akış ilk kez bu çalışmada ele alınmıştır [63].

4. KULLANILAN YÖNTEMLER

4.1.En Kısa Yol Algoritmaları

Şebeke, basit anlamda dallarla birbirine bağlanan düğümler kümesi olarak ifade edilebilir. En kısa yol problemleri, belirli bir şebeke üzerinde maliyet-süre gibi parametreler altında bir düğümden başka bir düğüme ulaşan en ideal yolu bulmaktır. Bir şebekede, başlangıç noktasından varış noktasına ulaşan birçok farklı alternatif olabilir. En kısa yol problemlerindeki amaç ise bu yolların içerisinde, alternatiflerine göre, mesafe ve maliyetçe en düşük olan yolları bulmaktır.

Bu amaçla ilk olarak 1959 yılında Hollandalı bilgisayar bilimcisi Edsger Dijkstra Dijkstra algoritmasını geliştirmiştir [64]. Sonrasında Floyd–Warshall, Bellman-Ford, Johnson, DAG ve A* Search gibi birçok en kısa yol algoritması geliştirilmiş ve literatürde farklı çalışmalarda bunlardan faydalanılmıştır [65]. Literatür incelendiğinde söz konusu algoritmaların başlıca kullanım alanları ulaşım, telekomünikasyon, enerji nakil hatları, bilgisayar ağları, seyahat planlaması, trafik planlama acil durum kaçış güzergâhları ve yol yapımı problemleri olduğu görülmektedir.

En kısa yol problemlerinde başlangıç düğümü ve hedef düğüm olmalıdır. En kısa yol problemlerinde çözüm için iki algoritma vardır. Bunlar Dijkstra Algoritması ve Floyd-Warshall Algoritmasıdır.

4.1.1. Floyd-Warshall algoritması

Floyd-Warshall algoritması, ağırlıklı bir grafta bulunan tüm düğüm çiftleri için en kısa yolun hesaplanmasında kullanılan bir algoritmadır. Algoritma, döngü içermeyen bir grafta, negatif veya pozitif ağırlıklı olmasına bakılmaksızın optimal sonuç vermektedir. En kısa yolun bulunmasında problem daha küçük alt problemlere bölerek, dinamik programlama yaklaşımını kullanmaktadır [66].

Dijksra algoritmasından daha genel olan Floyd algoritması şebekedeki herhangi iki düğüm arasındaki en kısa yolu belirlemede tercih edilmektedir. Algoritma, n düğümlü şebekeyi n satırlı ve n sütunlu matris olarak gösterir. Matrisin (i,j) elemanı, i. düğümden j. düğüme olan d_{ij} sonlu yoksa sonsuz olarak kabul edilir.

0.Adım: Aşağıdaki gibi D^0 uzaklık matrisini hazırlanır. Bu matrisin çapraz elemanları sıfır olup, birbirine bağlantısı olmayan düğümler (sonsuz) işareti ile gösterilir.

$$D^0 = \begin{matrix} - & d_{12} & d_{13} & \ldots_{..} & d_{1n} \\ d_{21} & & & & \\ d_{31} & & & & \\ \ldots_{..} & & & & \\ d_{n1} & .. & .. & .. & .. \end{matrix} \quad (4.1)$$

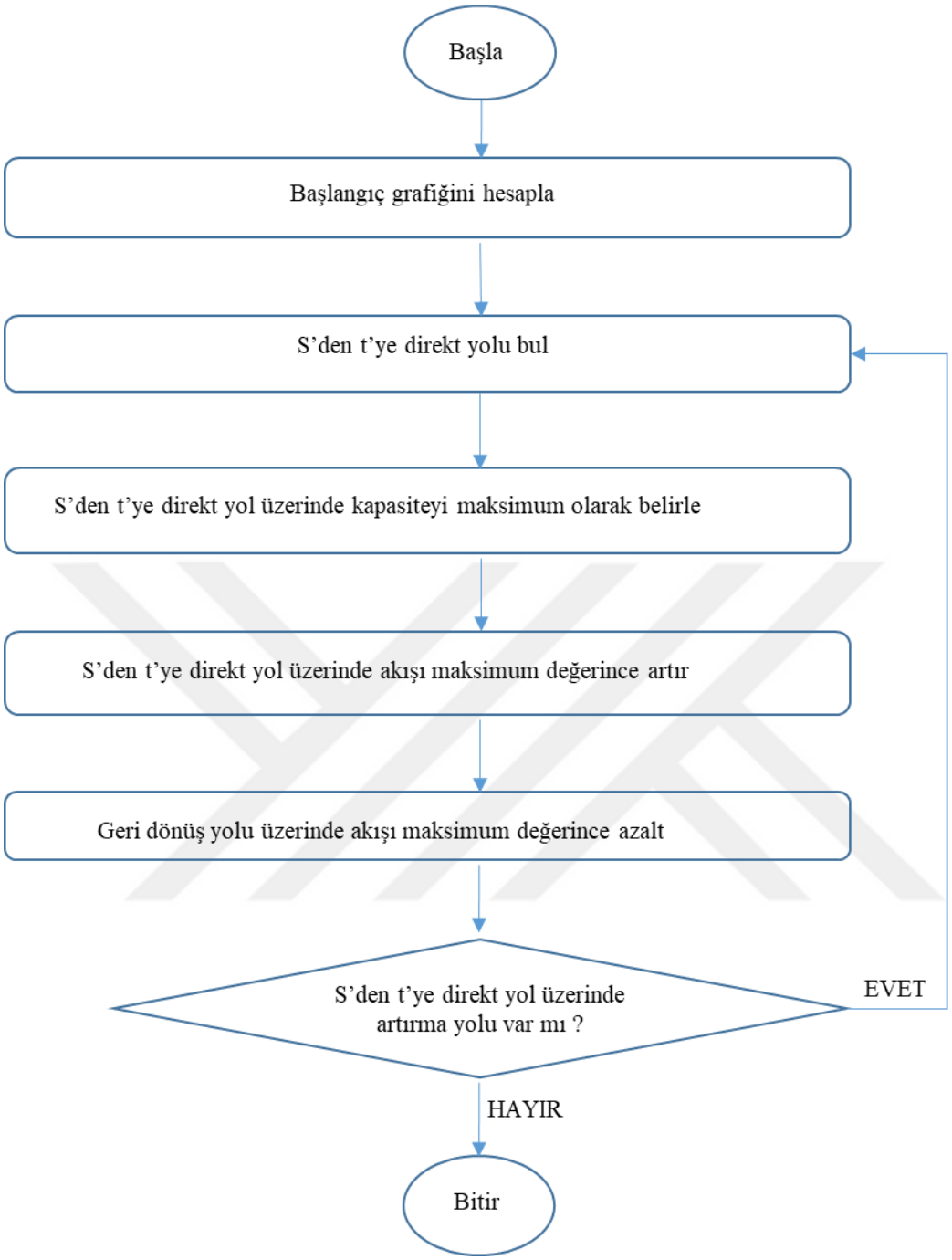
k.genel adım: k. sütunu ve k. satırı anahtar sütun ve anahtar satır olarak tanımlanır. D_k matrisine anahtar satır ve sütundaki değerleri yazılır. D_k matrisindeki diğer elemanları aşağıdaki bağıntıya göre belirlenir:

$$D_k(i,j) = \min\{ D_{k-1}(i,j), D_{k-1}(i,k) + D_{k-1}(k,j) \quad (4.2)$$

Formül tüm düğümler için tekrarlanır [66].

4.2.Maksimum Akış Problemleri

Burada "akış göndermek için kullanılabilecek alternatif yollar arasından en kısa yol hangisidir ve bu yolun diğer yollar ile maksimim akış yönünden karşılaştırması nedir?" sorusunun cevabı araştırılır. Ağ üzerinden akış sağlamak için yol arama algoritmalarına ihtiyaç vardır [67]. Ford-Fulkerson algoritması da bu yol arama algoritmalarından derinlik öncelikli arama algoritmasını kullanır. Bunu yaparken ilk akıştan başlar ve sürekli artan bir akış dizisi oluşturarak maksimum akışa ulaşır. Bu fikir, algoritma içinde s'den t'ye pozitif kullanılabilir kapasitenin altına bir artırma yolu olarak bir yol ekleme fikrine dayanmaktadır. Bu bağlamda Ford-Fulkerson algoritması iki temel adım içermektedir. İlk adım, artırma yolu ile akışı artırarak bir etiketleme işlemidir. İkinci adım, akışı buna göre düzenlemek ve değiştirmektir. Bu iki temel adım, akış artırma yolu kalmayıncaya kadar devam eder ve mümkün olan maksimum akış elde edilir. Yöntemin algoritma adımlarının akış üzerindeki gösterimi Şekil 4.1' de sunulmuştur.



Şekil 4.1. Ford-Fulkerson akış grafiği

İlk artık grafik oluşturulur (residual graph).

Artık grafikte başlangıç düğümünden bitiş düğümüne giden yol olduğu sürece belirtilen adımlar yapılır:

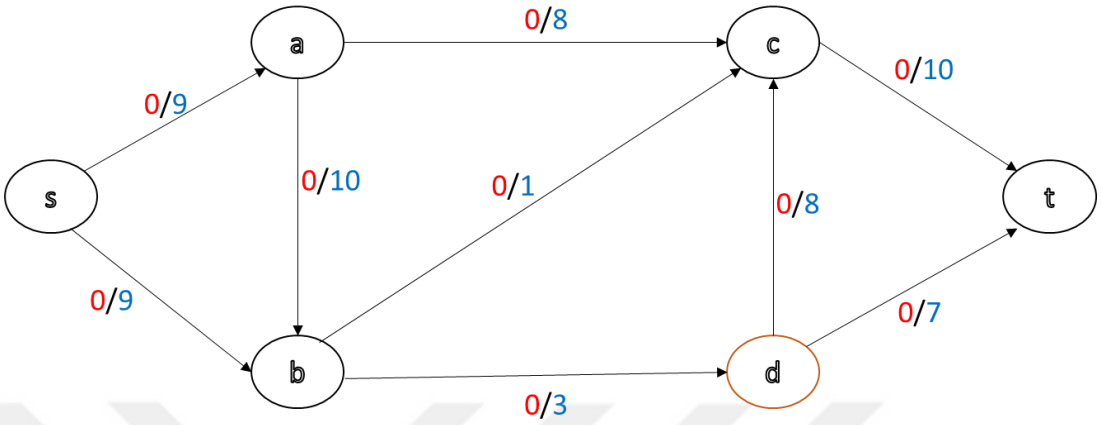
a) Başlangıç düğümünden bitiş düğümüne olan yol belirlenir,

b) Yol üzerindeki maksimum kapasite (δ) belirlenir,

c) Yol üzerindeki akış maksimum kapasite miktarınca arttırılır,

d) Geri dönüş yolundaki akış maksimum kapasite oranında azaltılır.

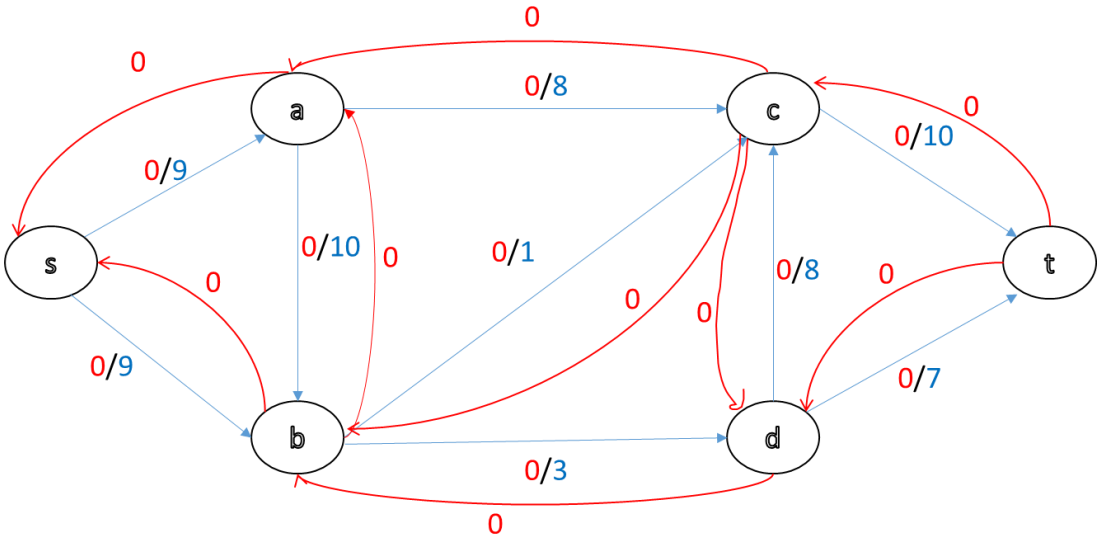
Bir örnek ile algoritmanın çalışması incelenmek istendiğinde aşağıda verilen grafikte s düğümünde bulunan depodan t düğümüne akan maksimum akış miktarı bulunur.



Şekil 4.2. Şebeke akış grafiği

Şekil 4.2.'de yer alan kırmızı renk ile ifade edilen değer hat üzerinde bulunan akışın anlık miktarını, mavi değer de hattın kapasitesini göstermektedir. s düğümünden a düğümüne maksimum 9 birimlik akış gönderilebilir. Aynı şekilde, a-c yolunun kapasitesi 8 birim olduğu görülmektedir.

Algoritmanın birinci basamağında ilk arta kalan şebeke belirlenir.

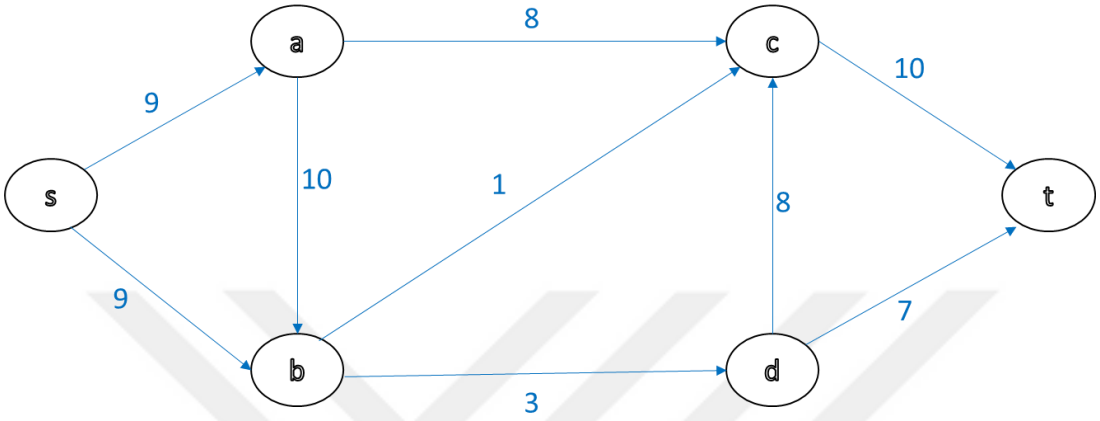


Şekil 4.3. Arta kalan şebeke

Arta kalan şebeke belirlenirken güncel şebeke grafiği değerlendirilir. Şebeke grafiğinde s-a kapasitesi 9 birim olmakla birlikte hatta anlık akış olmadığı görülmektedir. Bu itibarla s-a hattının kapasite durumu 9-0 = 9 birim ve bu yolda akış

olmadığı için a düğümünden s düğümüne dönüş 0 birim olarak görülmektedir. Bunun gibi diğer tüm düğümlerin de gidiş ile dönüş değerleri belirlenir.

Arta kalan şebeke oluşturulurken sayısal değeri olmayan hatların çizilmesine gerek yoktur. İlk arta kalan şebekeden değeri sıfır olan yollar çıkartılıp sadeleştirilme yapıldığında aşağıdaki şekil ortaya çıkar.



Şekil 4.4. Ford-Fulkerson akış grafiği sadeleştirilmiş hali

Algoritmanın bir ileriki adımında arta kalan şebekede, başlangıç düğümünden bitiş düğümüne ulaşan herhangi bir yol aranmaktadır. Başlangıç düğümünden bitiş düğümüne ulaşan yolar:

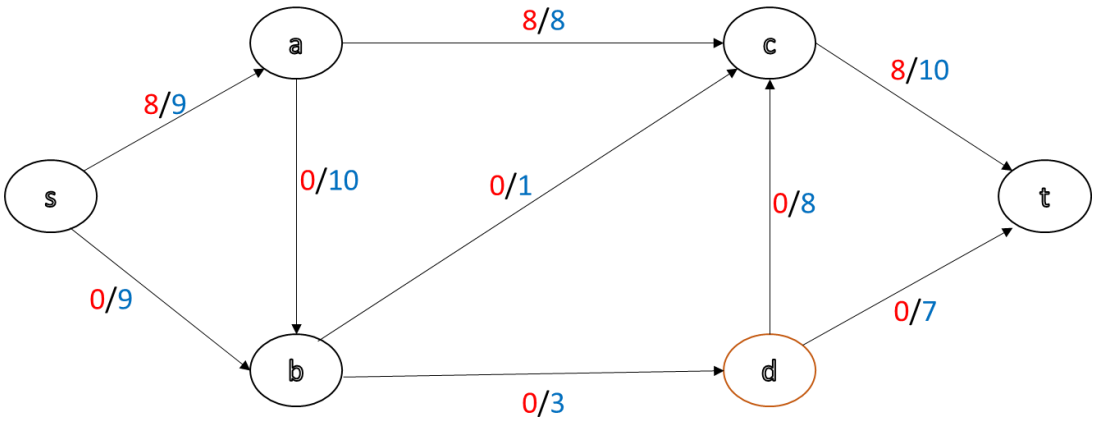
s→a→c→t

s→b→d→t

s→a→b→c→t

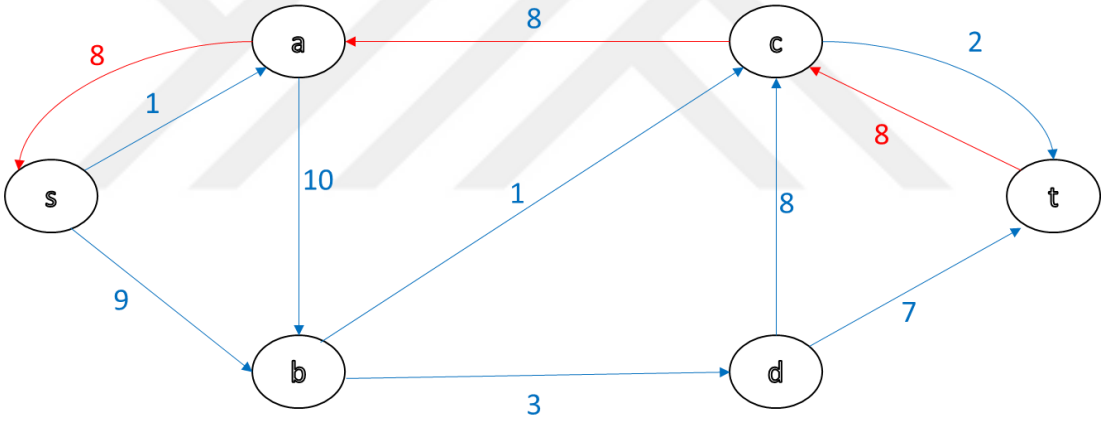
Bu noktada tercih edilen yolun maksimum kapasitesi bulunur. Tercih edilen yolun kapasitesi hat üzerindeki en düşük kapasiteye eşit olduğu bilinmelidir.

(s, a), (a, c), (c, t) δ=8



Şekil 4.5. Ford-Fulkerson akış grafiği

İlk aşamada s-a-c-t yolu tercih edildiğinde yol kapasitesi min {9, 8, 10} = 8 birim olmaktadır. Bir ileriki adım ile başlangıç grafiği yinelenir ve tercih edilen yoldaki akış güncellenir. Bir ileriki aşamadan önce yinelenen grafiğe ait arta kalan şebeke hazırlanır.



Şekil 4.6. Ford-Fulkerson arta kalan şebeke

Hazırlanan grafikte s-a-c-t hattına ait giden ve gelen sayılar güncellenir. s-a hattının kapasitesi 9 birimdi ve güzergâh üzerinde başka bir akış bulunmamaktaydı. Bir önceki iterasyonda bu güzergâh üzerinden bitiş düğümüne 8 birimlik bir akış sağlanmıştı. Bu itibarla s-a güzergâhının yeni akış değeri 8, kapasite değeri ise 9-8 = 1 birim olmuştur. Aynı şekilde a-c ve c-t güzergâhının değerleri de güncellenmelidir. Güncellenen grafikte bu güzergâha 8 birimlik akış kazandırıldığı için arta kalan şebekede a-c güzergâhının yeni kapasitesi 8-8 = 0 birim ve akış değeri 8 birim olur ve iki nokta arasındaki akış miktarı 8 birim olarak ifade edilmektedir. Nihai olarak yinelenen c-t güzergâhına ait 10-8 = 2 birimlik kapasite değeri kalmış ve 8 birimlik akış olmuştur.

Algoritma değerlendirildiğinde oluşan yeni arta kalan şebekede başlangıç düğümünden bitiş düğümüne yeni güzergâh aranır. Yeni bulunan güzergâhlar:

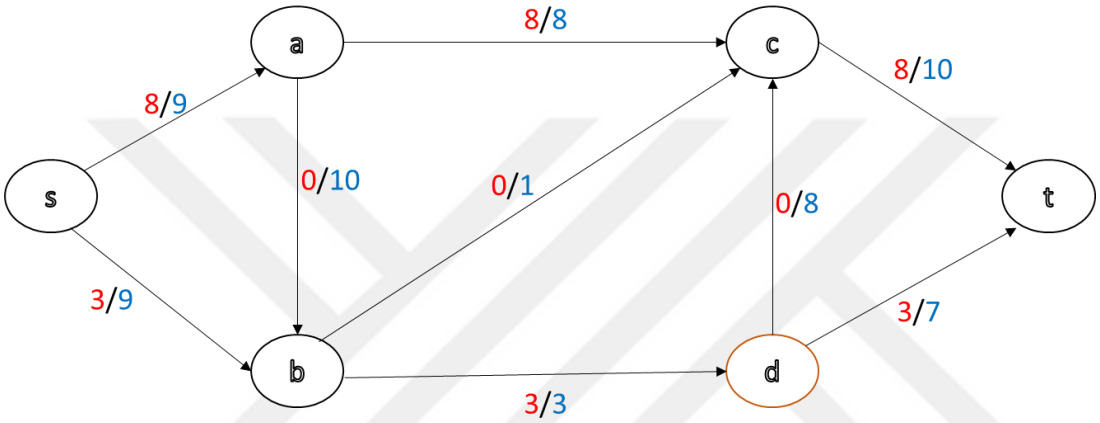
s-a-b-c-t

s-b-c-t

s-b-d-t

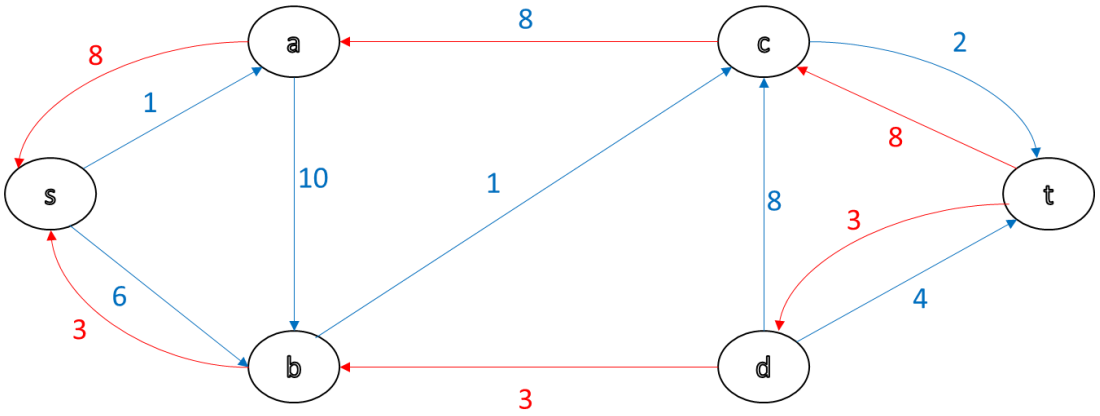
Bu güzergâhlardan bir tanesi tercih edilir ve yeni seçilen güzergâh s-b-d-t olur ise;

(s,b), (b,d), (d, t) δ=3



Şekil 4.7. Ford-Fulkerson akış grafiği

Bu güzergâhın maksimum kapasitesi min{9, 3, 7} = 3 birimdir. Bir önceki güncel grafik tekrardan yenilenir, seçilen güzergâh üzerinden 3 birimlik akış elde edilir ve hazırlanan bu yeni grafiğe ait arta kalan şebeke oluşturulur.



Şekil 4.8. Ford-Fulkerson arta kalan şebeke

Arta kalan şebekede seçilen güzergâhtaki kapasite ve akış değerleri yinelenir. s-b güzergâhının kapasitesi 9 birim olup herhangi bir akış bulunmamaktaydı. Grafikte yer

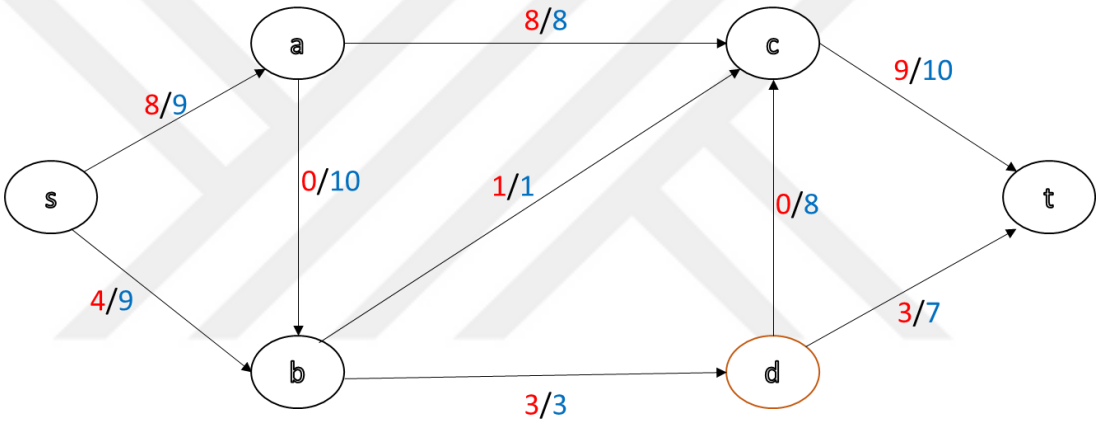
alan yeni akış ile s-b güzergâhının yeni kapasitesi 9-3 = 6 birim ve akış değeri 3 birim şeklinde hesaplanır. Yine b-d güzergâhının yeni kapasitesi 3-3 = 0 birim ve akışı 3 birim olarak hesaplanır. Sonuçta d-t güzergâhının sahip olduğu 7 birimlik kapasite 7-3 = 4 birime düşer.

Algoritmada son arta kalan şebekeye bakılarak yeni güzergâh aranır. İki farklı güzergâh olduğu görülür:

s-a-b-c-t

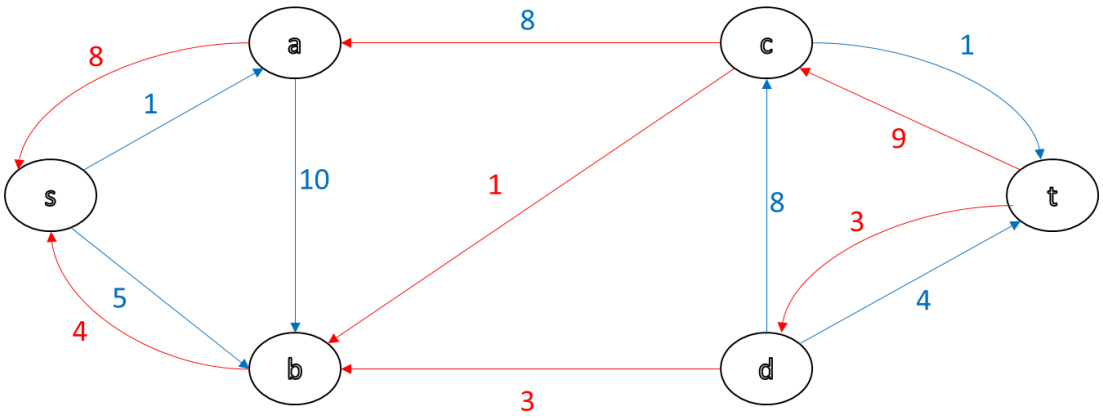
s-b-c-t

Yeni tercih edilen güzergâh s-b-c-t olduğunda (s,b), (b,c), (c, t) δ=1 olarak karşımıza çıkmaktadır.



Şekil 4.9. Ford-Fulkerson akış grafiği

Son güncel arta kalan şebekede güzergâhın maksimum kapasitesi min{6, 1, 2} = 1 birim olur ve bu güzergâh üzerinden fazladan 1 birimlik akış sağlanır. Yinelenen s-b güzergâhının yeni akış değeri 4, b-c güzergâhının 1 ve c-t güzergâhının ise 9 birim olduğu görülmektedir. Başlangıç düğümünden bitiş düğümüne yeni güzergâh için tekrardan arta kalan şebeke oluşturulur.



Şekil 4.10. Ford-Fulkerson arta kalan şebeke

Yinelenen arta kalan şebekede akış ve kapasite durumları güncellenir. 6 birimlik kapasiteye sahip s-b güzergâhının yeni kapasitesi 6-1 = 5 birime düşerken 3 birimlik olan akışı 3+1 = 4 birime yükselir. Yine b-c güzergâhının yeni kapasitesi 1-1 = 0 birim olurken akışı 0+1 = 1 birim olur ve c-t güzergâhı 2-1 = 1 birimlik kapasite ve 8+1= 9 birimlik akışa sahip olur. Yeni güzergâh bulmak için yinelenen arta kalan şebekeye bakıldığında başlangıç düğümünden yalnızca a ve b düğümlerine ulaşılabildiği görülmektedir. Varış düğümüne ulaşılamadığından dolayı algoritma tamamlanır.

Çalışma sonlandığı ve algoritma tamamlandığından maksimum akış hesaplanır. Bu akış arta kalan şebekede başlangıç düğümünden akan değerlerin toplamına ya da bitiş düğümünden çıkan değerler toplamı ile aynı olmalıdır. s düğümüne giren değerlerin tümü 8+4 = 12 birim ve t düğümünden çıkan değerlerin tümü 9+3 = 12 birimdir. Yani s düğümünde bulunan bir merkezinden t düğümüne en fazla 12 birimlik akış sağlanabilir.

5. UYGULAMA

5.1. Problemin Tanımı

Çalışmada orta koridor üzerinde yer alan Ahılkelek-Kapıkule arasında çalışacak yük treninin çıkış noktasından varış noktasına olan en kısa yolun belirlenmesi ve devamında en kısa yolun maksimum akışı karşılama durumu ele alınmış ve son olarak şebekede maksimum akış üzerine optimum bir çözüm sunulmuştur.

Trenlerin çıkış noktasından varış noktasına en kısa yoldan ulaşımı için işletmecilerin hangi güzergâhı tercih etmesi gerektiği sorusuna çözüm getirilmiştir.

Ayrıca Ford-Fulkerson algoritması ile de belirlenen en kısa yolun kapasitesinin maksimum akış yönünden yeterliliği ortaya koyularak şebeke üzerinde yer alan diğer alternatif yolların da şebekenin maksimum akışında ne kadar etkin olduğu tespit edilmiştir.

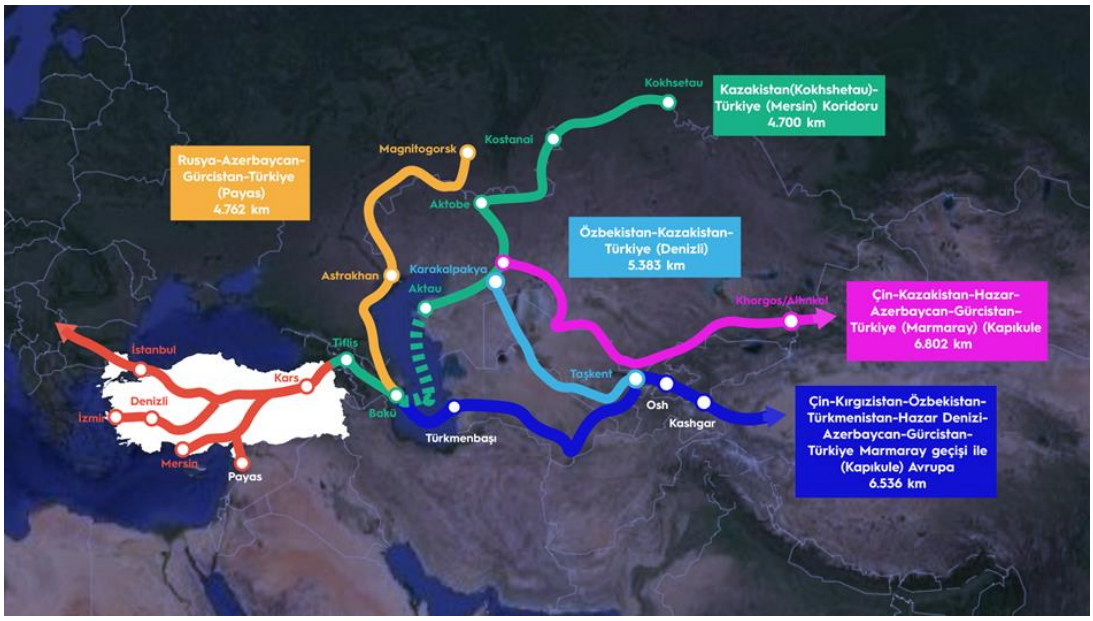
Bu tez kapsamındaki uygulamada, Ahılkelek'ten Kapıkule'ye ulaşan tüm demiryolu ağında şebeke optimizasyonu amaçlanmıştır. Elde edilen sonuçlar ile Modern İpek Yolu koridorunun avantajları ve söz konusu hat kesiminde mevcut kapasite durumu ve gelecek projeksiyonu bir arada değerlendirilmiştir. Bu amaçla Ahılkelek- Kapıkule arasındaki tüm demiryolu ağı şebeke olarak ele alınmış ve en kısa yol algoritmaları kullanılarak en kısa güzergâhın belirlenmesi amaçlanmıştır. Ayrıca Ford-Fulkerson algoritması ile de maksimum akış yönünden şebekenin diğer dalları ile kıyaslaması yapılmıştır.

Çin, Kazakistan, Hazar Denizi, Azerbaycan, Türkiye ve Avrupa güzergâhı olarak adlandırılan orta koridor esas alınmış, Ahılkelek-Kapıkule hat kesiminde 29 düğüm ve 34 daldan oluşan bir şebeke oluşturulmuştur. Problemin çözümü için literatürde sıklıkla kullanılan en kısa yol algoritması Floyd-Warshall tercih dilmiş, çözüm ise Python programlama diliyle yazılarak sağlanmıştır. Ford-Fulkerson algoritması ile de maksimum akış yönünden değerlendirme Java programlama diliyle yazılarak yapılmıştır. Böylelikle aynı çalışma içerisinde iki farklı programlama dili birlikte kullanılmıştır.

5.2.Verilerin Elde Edilmesi

Başlangıç noktası Çin olan ve ana olarak üç koridor üzerinden Avrupa'ya ulaşan Avrasya demiryolu ağı çok geniş bir coğrafyayı kapsamaktadır. Bu koridorlardan bir tanesi de orta koridor'dur. Orta koridor; Çin, Kazakistan, Hazar Denizi, Azerbaycan, Türkiye üzerinden Avrupa'ya ulaşan koridordur. Çalışmada orta koridor üzerinde yer alan Ahılkelek-Kapıkule hat kesimi üzerine yoğunlaşırmıştır.

Çin, Kazakistan, Hazar Denizi, Azerbaycan, Türkiye ve Avrupa güzergâhı olarak adlandırılan orta koridoru daha belirgin bir şekilde göstermek için Şekil 5.1 hazırlanmıştır.



Şekil 5.1. Orta koridor yol haritası

Uygulamada ele alınan şebekenin düğümleri demiryolu haritasının kavşak noktalarını Şekil 5.2'de göstermektedir.

Şekil 5.2. Şebeke düğümleri

Önemli demiryolu istasyonlarından oluşan kavşak noktalarını içeren çalışma toplam 29 düğümden oluşmaktadır. Düğümlerin şebeke üzerinde gösterilmesi Çizelge 5.1'de olduğu gibi nümerik olarak numaralandırma yöntemi seçilmiştir.

Uygulamada problem; maliyet, dal uzunluğu ve hat kapasitesi değerleri kullanılarak çözülmüştür.

Maliyet: Ulaştırma problemlerinde karar vericilerin öncelikle değerlendirdiği kriterlerin en başlarında maliyet gelmektedir. Firmalar kar amacıyla hizmet veya üretim yapan kuruluşlar olduğundan maliyet kriterinin ulaştırma sektöründeki problemlerde etkisi çok yüksektir. Demir yolu ulaştırma maliyeti yaygın olarak ton/km birimi kullanılmaktadır. Diğer bir deyişle, demiryolu fiyatlandırmasında bir ton yükün bir km deki taşıma ücreti şeklinde ifade edilir. Uygulamadaki dallar arasındaki maliyet bilgileri, demiryolu taşımacılığı hizmeti veren bir kuruluşun pazarlama departmanından elde edilmiştir.

Kapasite: Ülkelerin lojistik alanlarında yaptıkları yatırımların asıl amaçları bu alandaki kapasitelerini iyileştirmektir. Taşıma kapasitelerinin yüksek olması artan dünya ticaretinden daha fazla pay almada büyük etkendir. Bu amaçla, uygulamada her bir dalın taşıma kapasitesi gerçek değerlere göre yazılmıştır. Kullanılan veriler TCDD 2021 yılı şebeke bildiriminin Ek-3.3'de yer alan Şebekenin Teknik Özellikleri bölümünden alınmıştır [68].

Hat Uzunlukları: Uygulamadaki dallar arasındaki mesafe bilgileri, demiryolu taşımacılığı hizmeti veren bir kuruluşun pazarlama departmanından elde edilmiştir [69].

5.2.1. Düğümlerin Belirlenmesi

Çalışmada ele alınan şebeke toplam 29 düğüm ve 34 dal'dan meydana gelmektedir. Düğümlerin şebeke üzerinde ifade edilmesi ve algoritmalardaki kodlamada kullanım kolaylığı sağlaması amacıyla Çizelge 5.1'de gösterildiği üzere nümerik numaralandırma yöntemi seçilmiştir.

Çizelge 5.1. Şebeke düğümleri

Numerik	Alfabetik	Düğüm Adı	Numerik	Alfabetik	Düğüm Adı
0	A	Ahılkelek	15	P	Ulukışla
1	B	Kars	16	Q	Boğazköprü
2	C	Çetinkaya	17	R	Irmak
3	D	Malatya	18	S	Zonguldak
4	E	Bitlis	19	T	Afyon
5	F	Bostankaya	20	U	Denizli
6	G	Kalın	21	V	İzmir
7	H	Samsun	22	W	Manisa
8	I	Hanlı	23	X	Balıkesir
9	J	Toprakkale	24	Y	Alavunt
10	K	İskenderun	25	Z	Enveriye
11	L	Yenice	26	AA	Köseköv
12	M	Mersin	27	BB	Halkalı
13	N	Gömec	28	CC	Kapıkule
14	O	Kayseri			

En kısa yol algoritmalarının çalıştırılması, Python ve Java programlama dili ile yazılması sırasında kolaylık sağlanması amacıyla 29 adet düğüm Çizelge 5.1'de gösterildiği üzere alfabetik ve numerik karakterler atanmıştır. Bu sayede algoritmaların kod yazım aşamalarında basitlik sağlanmış, sade hale gelen düğüm isimlerinin kullanılmasıyla hata payı azaltılmıştır.

5.2.2. Dalların Belirlenmesi

Gürcistan (Ahılkelek)-Kapıkule arasındaki demiryolu ağından yola çıkılarak 29 düğümden oluşan şebekede, birbirleriyle direkt demiryolu ulaşımı olan 34 adet dal belirlenmiştir. Düğümlerde olduğu gibi en kısa yol algoritmalarının çalıştırılmasında kolaylık sağlanması adına her bir dala numerik değer atanmış ve Çizelge 5.2'de bu değerlere yer verilmiştir. Düğümler arası bağlantılar Şekil 5.2'de verilen harita

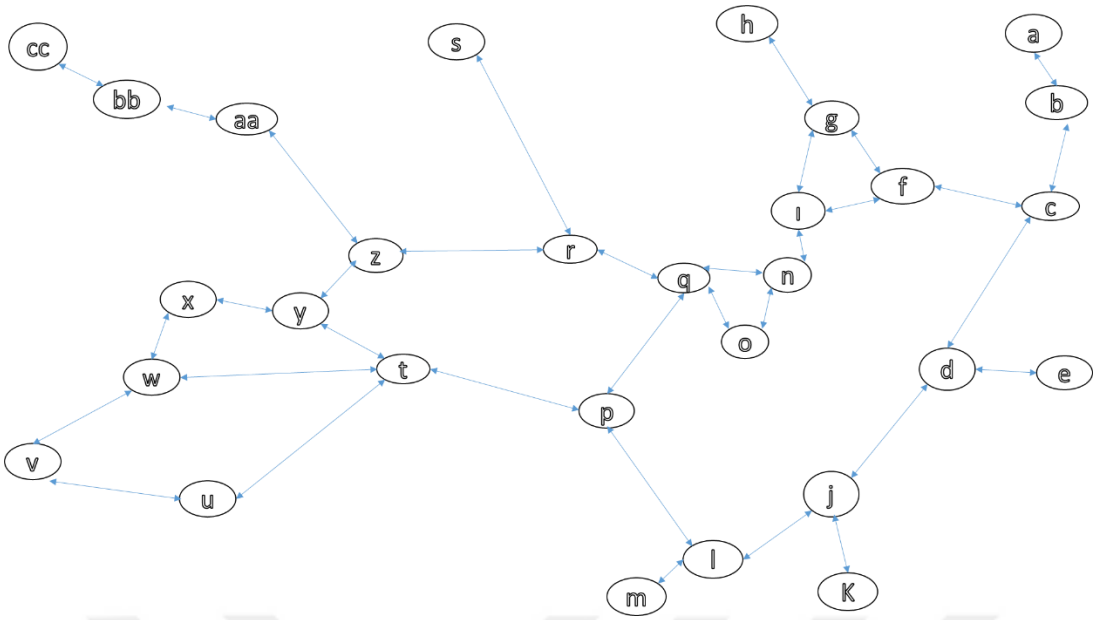
üzerinden oluşturulmuştur. Her bir düğümü ifade eden geçiş düğümlerinin istasyonlar arasındaki mesafe ise kilometre cinsinden hesaplanmıştır. Mesafe hesaplamada kaynak olarak bir firmanın demiryolu mesafe hesaplama aracı kullanılmıştır [68].

Çizelge 5.2. Şebeke dalları

Nu	Dal Adı	Nu	Dal Adı	Nu	Dal Adı
1	Ahılkelek-Kars	13	Yenice-Mersin	25	Denizli-İzmir
2	Kars-Çetinkaya	14	Yenice-Ulukışla	26	İzmir-Manisa
3	Çetinkaya-Malatya	15	Hanlı-Gömeç	27	Afyon-Manisa
4	Malatya-Bitlis	16	Gömeç-Kayseri	28	Manisa-Balıkesir
5	Çetinkaya-Bostankaya	17	Gömeç-Boğazköprü	29	Afyon-Alayunt
6	Bostankaya-Kalın	18	Kayseri-Boğazköprü	30	Balıkesir-Alayunt
7	Kalın-Samsun	19	Ulukışla-Boğazköprü	31	Alayunt-Enveriye
8	Hanlı-Kalın	20	Ulukışla-Afyon	32	Enveriye-Köseköy
9	Hanlı-Bostankaya	21	Boğazköprü-Irmak	33	Köseköy-Halkalı
10	Malatya-Toprakkale	22	Irmak-Zonguldak	34	Halkalı-Kapıkule
11	Torakkale-İskenderun	23	Irmak-Enveriye		
12	Toprakkale-Yenice	24	Afyon-Denizli		

5.3.Şebekenin En Kısa Yol Algoritması ile Çözülmesi

Şebeke dallarının bir önceki kısımda belirlenmesinin ardından problemin çözümünün ikinci kısmına geçilmiştir. Geleneksel en kısa yol problemlerinden farklı olarak bu tez çalışmasında en kısa yol algoritmalarının çözümünde ayrıca dal ağırlıkları uzunluk ve maliyet değerlendirilerek en kısa yol algoritmaları bu veriler ile çalıştırılmış ve sonuçlar yorumlanmıştır. Problemin çözümünde en kısa yol algoritmalarından sıklıkla kullanılan ve optimal sonuç veren Floyd-Warshal tercih edilerek bu algoritmalar Pycharm yazılımı ara yüzünde Python programlama dilinde kodlanarak çözüm üretilmiştir.



Şekil 5.3. Düğümler arasındaki şebeke ağı

Temelini Şekil 5.2'teki demiryolu haritasının oluşturduğu şebeke düğümlerinin birbirleriyle olan ilişkileri şebeke üzerinde gösterilmiş ve Şekil 5.3'de verilmiştir.

5.3.1. Floyd Warshall Algoritması İle Çözümü

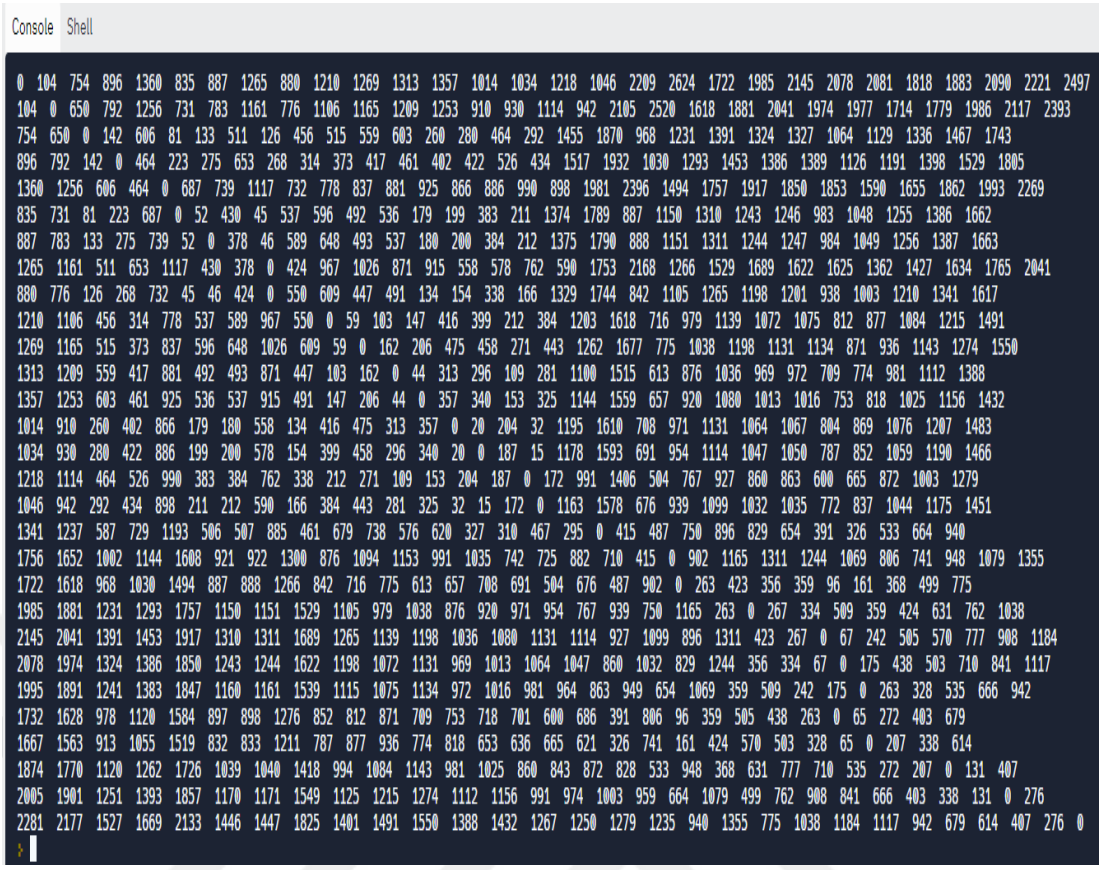
En kısa yolun tespiti ve tespit edilen yolda ton/km başına düşen maliyeti bulmak için şebeke çözümünde esas alınan mesafe ve maliyet değerleri ile Floyd Warshall algoritması kodlanmış ve elde edilen sonuçlar aşağıda verilmiştir.

Floyd-Warshall algoritması pseudo (sözde) kod:

Floyd-Warshall algoritması problemin karakteristik özelliklerine uygun olmasından dolayı elde edilen gerçek verilere göre Python programlama dilinde kodlanmıştır.

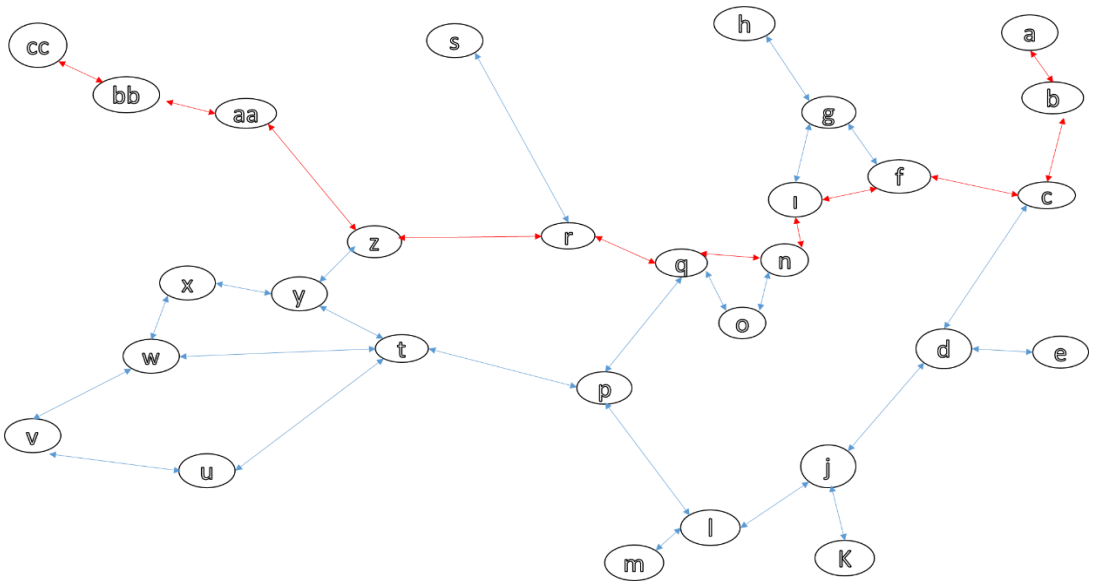
Floyd Warshall algoritması Pseudo (sözde) kod:

1: **for** $k \leftarrow 1$ **to** n
2: **do for** $i \leftarrow 1$ **to** n
3: **do for** $j \leftarrow 1$ **to** n
4: **if** cij > cik + ckj
5: **then** $c_{ij} \leftarrow c_{ik} + c_{kj}$



Şekil 5.4. Floyd Warshall algoritması mesafe değerleri ile çözüm sonuçları

Dalların birbirlerine olan km cinsinden uzaklık değerleri kullanılarak Floyd Warshall algoritması çalıştırılmış, başlangıç düğümünden bitiş düğümüne olan en kısa yol 2.497 km olarak hesaplanmıştır. En kısa yol değerlerinden yola çıkarak şebeke a-b-c-f-ı-n-q-r-z-aa-bb-cc şeklinde karşımıza çıkmaktadır.



Şekil 5.5. Başlangıç ve bitiş düğümler arasında en kısa yol

Dallar arasında yapılan taşıma maliyet değerleri kullanılarak Floyd Warshall algoritması çalıştırılmış, başlangıç düğümünden bitiş düğümüne en avantajlı olan a-b-c-f-ı-n-q-r-z-aa-bb-cc yolun toplam bir ton başına km maliyeti 672 tl olarak hesaplanmıştır.

Şekil 5.6. Floyd Warshall algoritması maliyet değerleri ile çözüm sonuçları

5.4.Şebekenin Maksimum Akış Ford-Fulkerson Algoritması ile Çözülmesi (Mevcut Durum Çözümü)

Ford Fulkerson algoritması Pseudo (sözde) kod:

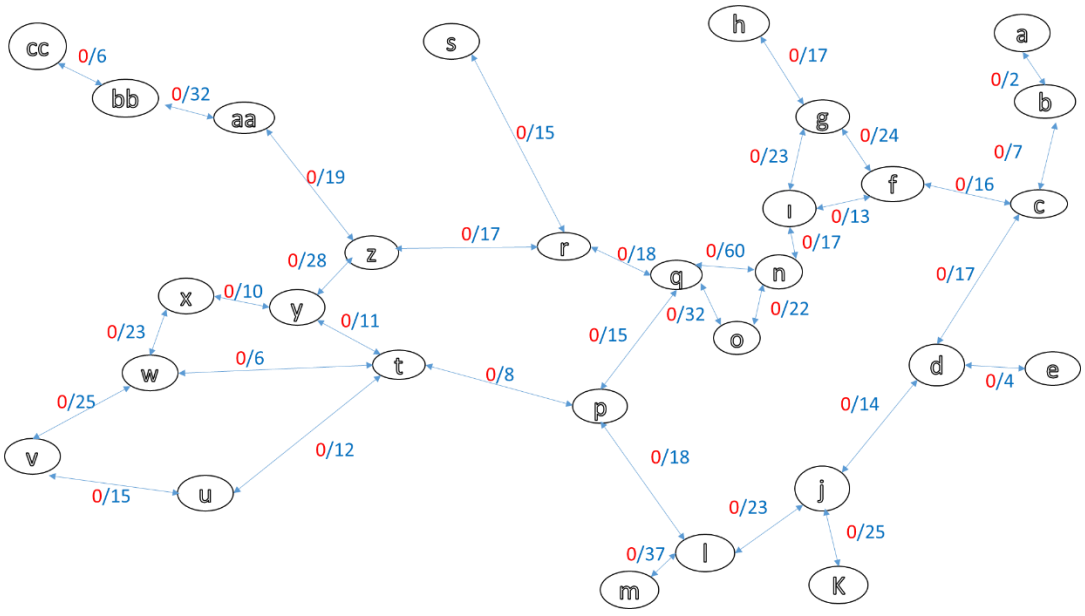
Entry specified a network G = (Y, O) with flow capacity
c, a source node s, and a sink node t
Output calculate a flow f from s to t of the maximum amount
1 **for** each vertex (x,y) ∈ O [G]
2 **do** f [x, y] = 0
3 f [y, x] = 0
4 **return** there exists a path p from s to t
in the residual graph G_f
5 **do** $c_f(p) = \min\{c_f(x, y) \mid (x, y) \in p\}$
6 **for** each vertex (u, y) in p
7 **do** f [x, y] = f [x, y] + c_f (p)
8 f [y, x] = −f [x, y]

Bir başlangıç düğümünden hedef düğümüne olan maksimum akışı tespit etmek için kullanılan Ford-Fulkerson Algoritması, çalışmada en kısa yol hesaplamada kullanılan Python programlama dilinden farklı olarak hem de çalışmaya zenginlik kazandırması açısından Java programlama dilinde kodlanarak çözümlenmiştir.

Çizelge 5.3. Dalların mevcut kapasite durumları

Nu	Dal	Kapasite	Nu	Dal	Kapasite	Nu	Dal	Kapasite
1	a-b	2	13	l-m	37	25	u-v	15
2	b-c	7	14	l-p	18	26	v-w	25
3	c-d	17	15	I-n	17	27	t-w	6
4	d-e	4	16	n-o	22	28	w-x	23
5	c-f	16	17	n-q	60	29	t-y	11
6	f-g	24	18	o-q	32	30	x-y	10
7	g-h	17	19	p-q	15	31	y-z	28
8	g-ı	23	20	p-t	8	32	z-aa	19
9	f-ı	13	21	q-r	18	33	aa-bb	32
10	d-j	14	22	r-s	15	34	bb-cc	6
11	j-k	25	23	r-z	17			
12	j-l	23	24	t-u	12			

Şekil 5.7'de yer alan kırmızı değerler anlık hat üzerinde yer alan tren sayısını, mavi değerler ise demiryolunun kapasitesini ifade etmektedir. Yani a düğümünden b düğümüne maksimum 2 birim tren gönderilebilir ancak anlık olarak hatta akış gözlenmemektdir. Yine b-c hattının kapasitesi 7 birim trendir ve bu güzergâh üzerinde de henüz bir tren akışı bulunmamaktadır.



Şekil 5.7. İlk arta kalan şebeke grafiği

Arta kalan şebeke belirlenirken güncel grafik dikkate alınır. Şuan ki güncel grafikte a-b hat kapasitesinin 2 tren olduğu henüz bir tren akışı gözlenmemektedir. Bu itibarla a-b hattının kapasitesi 2-0 = 2 (kapasite – tren akışı) ve herhangi bir tren yolda olmadığı için b düğümünden a düğümüne dönüş yolu 0 birimdir. Yine b düğümünden c düğümüne herhangi bir tren akışı yoktur ve hattın kapasitesi 7 birim trendir. Bu sebepten dolayı b düğümünden c düğümüne giden hattın değeri 7-0 = 7 tren, c düğümünden b düğümüne dönen hattın değeri 0 birim trendir. Aynı şekilde c-d hattı incelendiğinde hattın kapasitesinin 17 tren olduğu ve hat üzerinde bir akış olmadığı sonucuna varılır. Bu itibarla c-d değeri 17 tren, d-c değeri 0 birimdir. Bu şekilde tüm düğümlerin gidiş ve dönüşüne ait akış ile kapasite değerleri belirlenir.

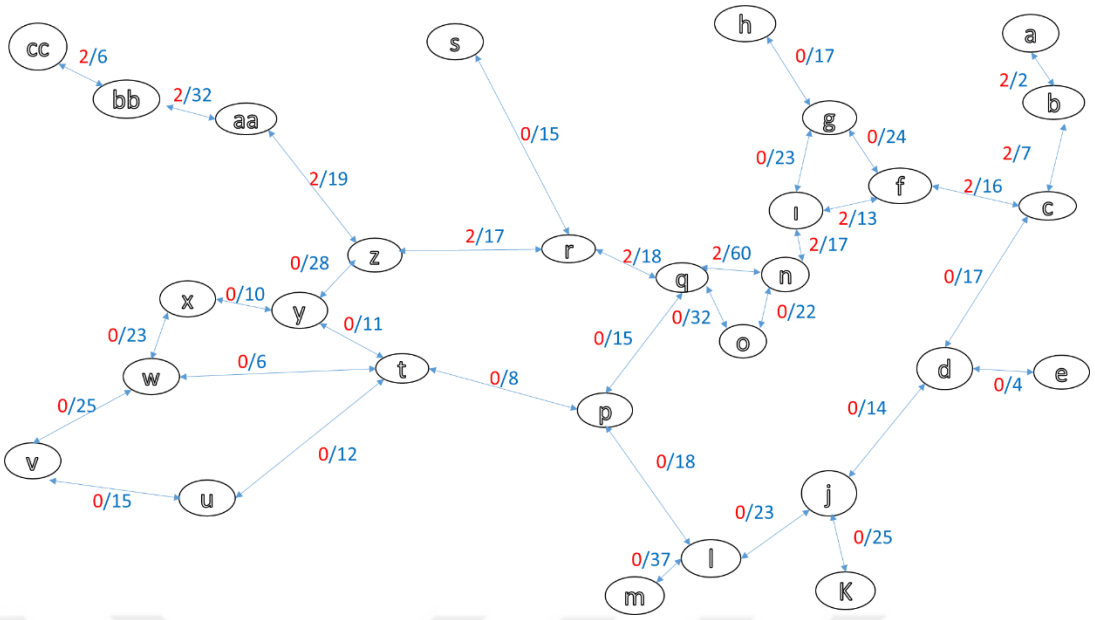
Algoritmanın ileriki aşamasında arta kalan grafik üzerinde, "a" başlangıç düğümünden "cc" bitiş düğümünden ulaşan bir hat aranır. Başlangıç düğümden varış düğümüne giden hatlar:

Çizelge 5.4. a 'dan cc'ye giden hatlar

a→b→c→f→g→ı→n→q→r→z→aa→bb→cc
a→b→c→f→ı→n→q→r→z→aa→bb→cc
a→b→c→f→ı→n→o→q→r→z→aa→bb→cc
a→b→c→f→g→ı→n→q→p→t→y→z→aa→bb→cc
a→b→c→f→g→ı→n→q→p→t→w→x→y→z→aa→bb→cc
a→b→c→f→g→ı→n→q→p→t→u→v→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→q→r→z→aa→bb→cc
a→b→c→d→j→l→p→t→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→u→w→x→y→z→aa→bb→cc

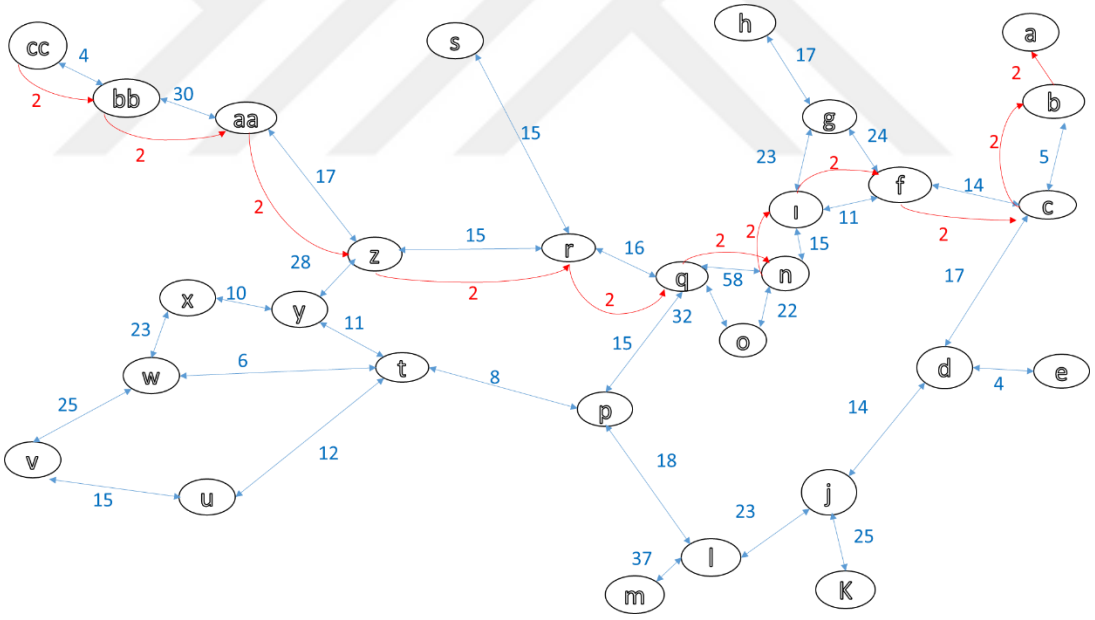
Bu yollardan bir hat tercih edilir ve seçilen hattın maksimum kapasitesi tespit edilir. Seçilen bu hattın kapasitesi hat üzerindeki en düşük kapasiteye eşittir. Seçilecek hattı en kısa yol olarak bulduğumuz a→b→c→f→ı→n→q→r→z→aa→bb→cc olacak şekilde öncelikli olarak seçtiğimizde ise;

(a,b), (b,c), (c, f), (f,ı), (ı,n), (n,q), (q,r), (r,z), (z,aa), (aa,bb), (bb,cc) δ=2



Şekil 5.8. a-b-c-f-ı-n-q-r-z-aa-bb-cc yolu için arta kalan şebeke

Kaynak düğümden batak düğüme giden yolun da kapasitesi min{2, 7, 16, 13, 17, 60, 18, 17, 19, 32, 6} = 2 tren olarak bulunur.



Şekil 5.9. Tren sevki için hesaplanan maksimum akış ağ modeli

Artık grafik incelendiğinde kaynak düğümünün kapasitesinin en kısa yol olarak bulduğumuz a→b→c→f→ı→n→q→r→z→aa→bb→cc yolunun sağladığı görülebilmektedir. Böylece söz konusu şebeke üzerinde yer alan kapasiteler için algoritma tamamlanır. Başlangıçta bulunan en kısa yol ile maksimum akış çözümü birbiriyle örtüşmekte olduğu görülmüştür. Diğer bir ifadeyle en kısa yol üzerinden maksimum akışı sağlayan taşıma gerçekleştirilebilir diyebiliriz. En kısa yol

değerlerinden yola çıkarak şebeke incelendiğinde karşımıza a→b→c→f→ı→n→q→r→z→aa→bb→cc güzergâhı en kısa yol olarak karşımıza çıkmaktaydı. Şebekeyi maksimum akış yönünden incelediğimizde ise en kısa yolun yine maksimum akış yönünden de uygun olduğu görülmektedir. Maksimum akış kapasitesi en iyi olan şebekenin mevcut durum analizinde Şekil 5.9'da gösterildiği üzere; a→b→c→f→ı→n→q→r→z→aa→bb→cc güzergâhı olarak bulunmuştur.

Şebeke çözümünde grafik çözümünden ayrı olarak, Ford-Fulkerson algoritması Java programında kodlanmış ve elde edilen sonuçlar Şekil 5.10'da verilmiştir.

```java
import java.util.LinkedList;
import java.util.Queue;

public class MaxFlow_Ford_Fulkerson {
    static class Graph {
        int vertices;
        int graph[][];

        public Graph(int vertex, int[][] graph) {
            this.vertices = vertex;
            this.graph = graph;
        }

        public int findMaxFlow(int source, int sink) {
            //residual graph
            int[][] residualGraph = new int[vertices][vertices];

            //initialize residual graph same as original graph
            for (int i = 0; i <vertices ; i++) {
                for (int j = 0; j <vertices ; j++) {
                    residualGraph[i][j] = graph[i][j];
                }
            }

            //initialize parent [] to store the path Source to destination
            int [] parent = new int[vertices];

```

Şekil 5.10. Ford-Fulkerson algoritması ile çözüm sonuçları

```
            int max_flow = 0; //initialize the max flow

            while(isPathExist_BFS(residualGraph, source, sink, parent)){
                //if here means still path exist from source to destination

                //parent [] will have the path from source to destination
                //find the capacity which can be passed though the path (in parent[])

                int flow_capacity = Integer.MAX_VALUE;

                int t = sink;
                while(t!=source){
                    int s = parent[t];
                    flow_capacity = Math.min(flow_capacity, residualGraph[s][t]);
                    t = s;
                }

                //update the residual graph
                //reduce the capacity on fwd edge by flow_capacity
                //add the capacity on back edge by flow_capacity
                t = sink;
                while(t!=source){
                    int s = parent[t];
                    residualGraph[s][t]-=flow_capacity;
                    residualGraph[t][s]+=flow_capacity;
                    t = s;
                }

        //add flow_capacity to max value
        max_flow+=flow_capacity;
    }
    return max_flow;
}

public boolean isPathExist_BFS(int [][] residualGraph, int src, int dest, int [] parent){
    boolean pathFound = false;

    //create visited array [] to
    //keep track of visited vertices
    boolean [] visited = new boolean[vertices];

    //Create a queue for BFS
    Queue<Integer> queue = new LinkedList<>();

    //insert the source vertex, mark it visited
    queue.add(src);
    parent[src] = -1;
    visited[src] = true;

    while(queue.isEmpty()==false){
        int u = queue.poll();

        //visit all the adjacent vertices
        for (int v = 0; v <vertices ; v++) {
                //if vertex is not already visited and u-v edge weight >0
                if(visited[v]==false && residualGraph[u][v]>0) {
                    queue.add(v);
                    parent[v] = u;
                    visited[v] = true;
                }
            }
        }
        //check if dest is reached during BFS
        pathFound = visited[dest];
        return pathFound;
    }
}
```

Şekil 5.10. Ford-Fulkerson algoritması ile çözüm sonuçları (devamı)

```
public static void main(String[] args) {
    int vertices = 29;
    int graph[][] =
{ {0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{2, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 7, 0, 17, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 17, 0, 4, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 16, 0, 0, 0, 24, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 24, 0, 17, 23, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 13, 23, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 25, 23, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 0, 0, 37, 0, 0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 37, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 22, 0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 0, 0, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 0, 0, 0, 0, 15, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 32, 15, 0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 0, 15, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 12, 0, 6, 0, 11, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 15, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 25, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 25, 0, 23, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 0, 10, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0, 0, 10, 0, 28, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 28, 0, 19, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 0, 32, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32, 0, 6},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0}
    };
    Graph g = new Graph(vertices, graph);
    int source = 0;
    int destination = 28;
    int max_flow = g.findMaxFlow(source,destination);
    System.out.println("Maximum flow from source: " + source + " to destination: " + destina
            + " is: " + max_flow);
}
```

Şekil 5.10. Ford-Fulkerson algoritması ile çözüm sonuçları (devamı)

Ford-Fulkerson algoritması Java programında kodlanarak çözüldüğünde Şekil 5.10'da görüldüğü üzere 29 farklı destinasyonun yer aldığı şebeke için maksimum akış aynı şekilde 2 tren gün olarak bulunmuştur.

Output:

```
Maximum flow from source: 0 to destination: 28 is: 2
```

Time Taken: 0.08 seconds

Memory Used: 24312 KB

Şekil 5.11. Ford-Fulkerson algoritması çıktıları

Bu işlem Java uygulamasında Şekil 5.11'de görüldüğü üzere 0.08 saniyede çözüme kavuşturulmuş olup bellekte 24312 KB yer aldığı görülmektedir.

Ancak bu tezin diğer çalışmalardan farkı en kısa yol ve maksimum akışı aynı çalışmada kullanmak, karşılaştırmak ve bununla beraber gelecekte artan kapasiteyle güncel ve gelecek projeksiyonunu analiz ederek demiryolu sektörü ve en kısa yol/maksimum akış algoritmalarının karşılaştırması konularında literatüre katkı sağlamak olduğundan bir sonraki bölümde gelecek projeksiyonda en kısa yolun kapasitesinin maksimum akışın sağlanması için yetersiz kaldığı durum incelenerek şebekenin diğer dallarının maksimum akışı nasıl sağladığı durumu tespit edilmiştir.

5.5.Olası Durum Çözümü

Şebeke de görüldüğü üzere tren hat kapasitesi yönünden dar boğaz oluşturan a-b, b-c ve bb-cc dalları üzerinde planlanan alt yapı ve sinyalizasyon çalışmaların tamamlanmasıyla gelecek dönem hat kapasiteleri çalışmaya dahil edilerek şebekenin güncel durumu ile gelecek projeksiyonu arasında yani en kısa yolun kapasitesinin maksimum akışı sağlaması için yetersiz kaldığı durum incelenerek kıyaslama yapılmıştır.

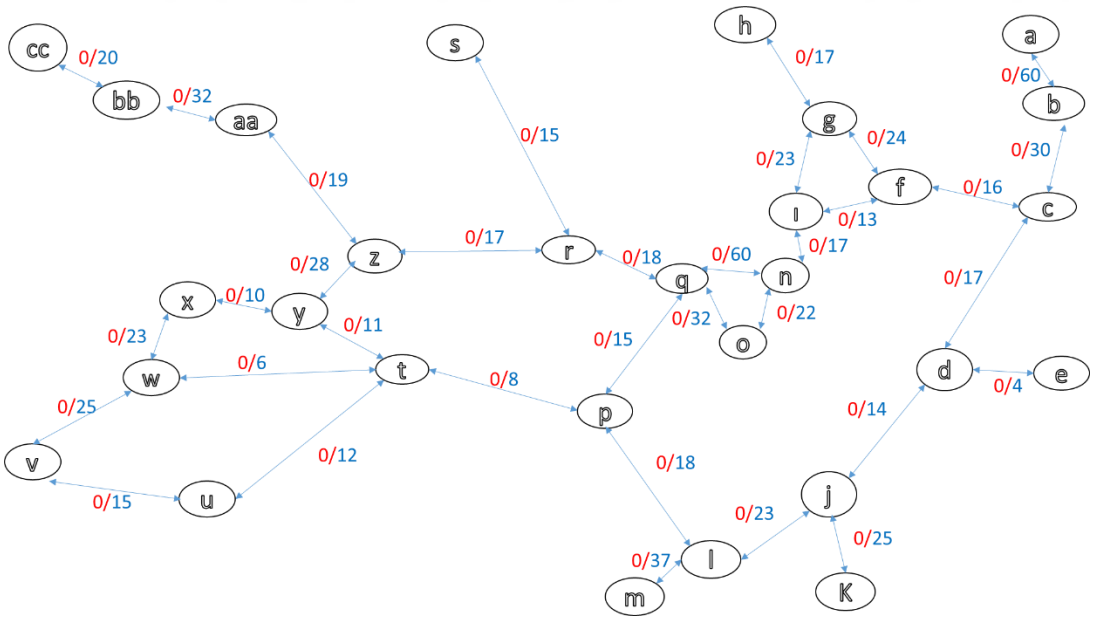
bb-cc hat kesiminde devam eden yol yapım çalışması ve sinyalizasyon çalışması tamamlandığında 6 tren olan hat kapasitesinin 20 tren/gün olması, a-b arası 2 tren olan hat kapasitesinin de hattın sinyalizasyon ve çift hat projesi tamamlanmasıyla birlikte hat kapasitesinin 60 tren/gün olması ayrıca b-c arası 7 tren olan hat kapasitesinin de hattın sinyalizasyon projesi tamamlanmasıyla birlikte hat kapasitesinin 30 tren/gün olması planlanmaktadır.

Çalışmanın bu bölümünde mevcut durum ile gelecek projeksiyon karşılaştırması yapılacak olup literatüre katkı sunulacaktır.

Çizelge 5.5. Şebeke dal kapasiteleri gelecek projeksiyonu

Nu	Dal	Kapasite	Nu	Dal	Kapasite	Nu	Dal	Kapasite
1	a-b	60	13	l-m	37	25	u-v	15
2	b-c	30	14	l-p	18	26	v-w	25
3	c-d	17	15	I-n	17	27	t-w	6
4	d-e	4	16	n-o	22	28	w-x	23
5	c-f	16	17	n-q	60	29	t-y	11
6	f-g	24	18	o-q	32	30	x-y	10
7	g-h	17	19	p-q	15	31	y-z	28
8	g-ı	23	20	p-t	8	32	z-aa	19
9	f-ı	13	21	q-r	18	33	aa-bb	32
10	d-j	14	22	r-s	15	34	bb-cc	20
11	j-k	25	23	r-z	17			
12	j-l	23	24	t-u	12			

bb-cc hat kesiminde devam eden yol yapım çalışması ve sinyalizasyon çalışması tamamlandığında 6 tren olan hat kapasitesinin 20 tren/gün olması, a-b arası 2 tren olan hat kapasitesinin de hattın sinyalizasyon ve çift hat projesi tamamlanmasıyla birlikte hat kapasitesinin 60 tren/gün olması ayrıca b-c arası 7 tren olan hat kapasitesinin de hattın sinyalizasyon projesi tamamlanmasıyla birlikte hat kapasitesinin 30 tren/gün olarak kabul edilmiş ve şebekenin dal kapasiteleri Çizelge 5.5 te olduğu gibi yeniden hesaplanmıştır.



Şekil 5.12. İlk arta kalan grafik

Şekilde hat üzerinde yer alan kırmızı değer o anki hat üzerindeki tren miktarını, mavi değer ise hattın kapasitesini ifade etmektedir. İlk arta kalan şebeke oluşturulurken güncel grafik göz önüne alınır. Şuan ki güncel grafikte a-b hat kapasitesinin 60 tren

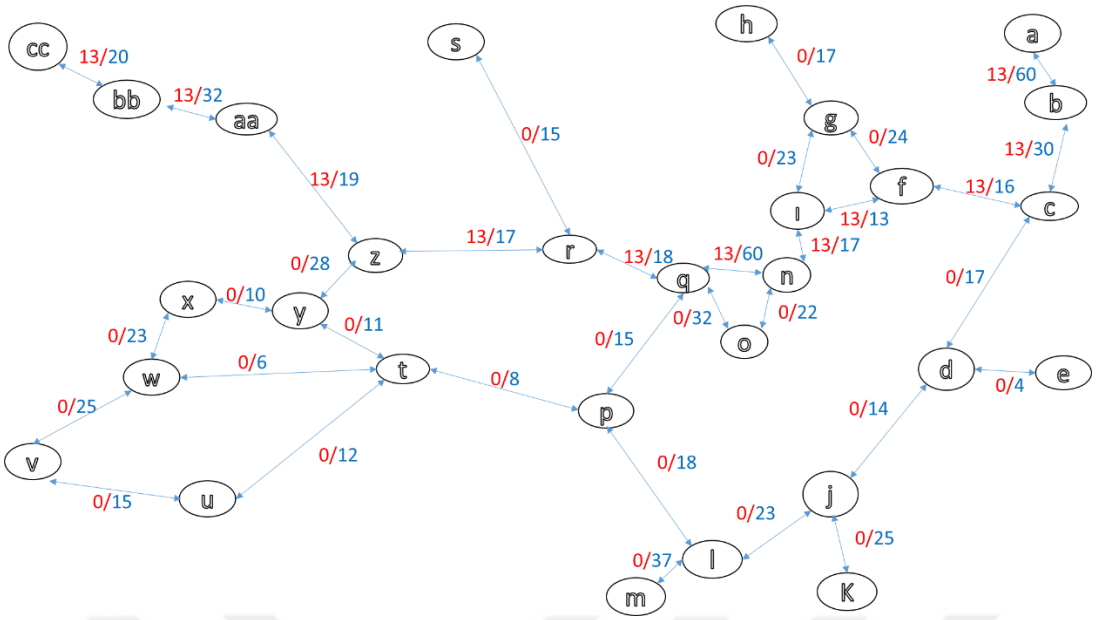
olduğu henüz bir tren akışı olmadığı görülmektedir. Bu itibarla a-b hattının kapasitesi 60-0 = 60 (kapasite – çalışan tren) ve b düğümünden a düğümünden dönüşte herhangi bir tren olmadığı için b-a 0 trendir. Yine b düğümünden c düğümünden akış yoktur ve hattın kapasitesi 30 trendir. Bu itibarla b düğümünden c düğümüne giden hattın değeri 30-0 = 30, c düğümünden b düğümüne dönen yolun değeri 0 birimdir. Yine c-d hattına bakıldığında hattın kapasitesinin 17 tren olduğu ve hat üzerinde başka bir akış olmadığı tespit edilir. Bu itibarla ötürü c-d değeri 17 tren ve d-c değeri 0 tren olarak bulunur. Böyle tüm düğümlerin gidiş ve dönüş değerleri bulunur.

Algoritmanın ileriki adımında yine arta kalan şebeke üzerinde, "a" düğümünden "cc" düğümüne ulaşan bir hat aranmaktadır. Bunu hatlar:

Çizelge 5.6. Başlangıç iterasyonu için a 'dan cc'ye giden hatlar

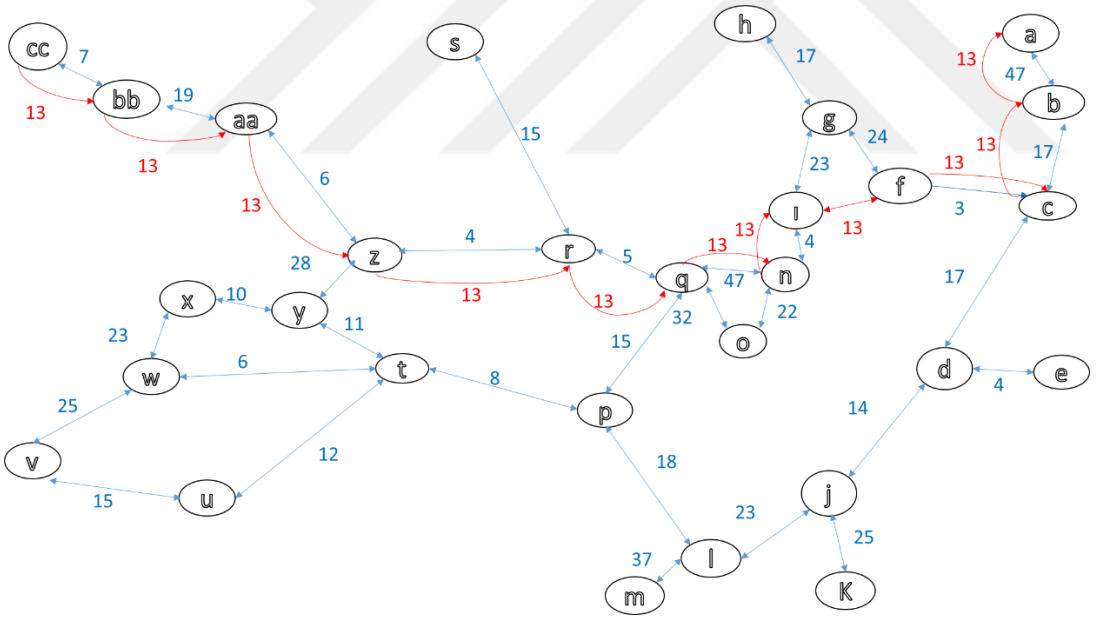
a→b→c→f→ı→n→q→r→z→aa→bb→cc
a→b→c→f→g→ı→n→q→r→z→aa→bb→cc
a→b→c→f→ı→n→o→q→r→z→aa→bb→cc
a→b→c→f→g→ı→n→q→p→t→y→z→aa→bb→cc
a→b→c→f→g→ı→n→q→p→t→w→x→y→z→aa→bb→cc
a→b→c→f→g→ı→n→q→p→t→u→v→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→q→r→z→aa→bb→cc
a→b→c→d→j→l→p→t→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→u→w→x→y→z→aa→bb→cc

Bu hatlardan öncelikle başlangıç düğümünden varış düğümüne en kısa hat olan a→b→c→f→ı→n→q→r→z→aa→bb→cc hattı seçilir ve tercih edilen hattın maksimum tren kapasitesi bulunur. Tercih edilen bu hattın kapasitesi hat üzerindeki en düşük kapasiteye eşittir. (a,b), (b,c), (c, f), (f,ı), (ı,n), (n,q), (q,r), (r,z), (z,aa), (aa,bb), (bb,cc) δ=13



Şekil 5.13. a-b-c-f-ı-n-q-r-z-aa-bb-cc yolu için ilk arta kalan şebeke

Başlangıç düğümünden varış düğümüne giden hattın kapasitesi min{60, 30, 16, 13, 17, 60, 18, 17, 19, 32, 20} = 13 tren olarak bulunur.



Şekil 5.14. Maksimum akış ağ modeli (a-b-c-f-ı-n-q-r-z-aa-bb-cc yolu için)

Bu hattın kapasitesi 13 birim tren olduğundan bir ileriki adımda başlangıç grafiği yinelenir ve tercih edilen hattaki akış güncellenir. İlerleyen adıma geçmeden yinelenen grafiğe ait yeni bir artık grafik hazırlanır.

Meydana gelen grafikte a→b→c→f→ı→n→q→r→z→aa→bb→cc hattına ait gidiş ve dönüş akış miktarları güncellenir. a-b hattının kapasitesi 60 birim trendi ve hat üzerinde herhangi bir tren akışı yoktu. Bundan önce hat üzerinden bitiş düğümüne 13

birim trenlik akış sağlanmıştı. Bundan dolayı a-b hattının yeni akışı 13 tren, kapasitesi 60-13 = 47 birim tren olmuştur. Yine (b,c), (c, f), (f, ı), (ı,n), (n,q), (p,r), (r,z), (z,aa), (aa,bb) ve (bb,cc) hatlarının değerleri de güncellenmelidir. b-c hattının yeni akışı 13 tren, kapasitesi 30-13 = 17 birim tren olmuştur. Yinelenen grafikte bu hatta 13 birim trenlik akış sağlandığından ilk arta kalan grafikte de c-f hattının yeni kapasitesi 16-13 = 3 birim tren ve akış değeri 16 birim trendir. f- ı hattının yeni akışı 13 tren, kapasite değeri ise 13-13 = 0 birim tren olmuştur. İki düğüm arasındaki akış 13 birim tren olarak gösterilmiştir. ı-n hattının yeni akışı 13 tren, kapasitesi ise 17-13 = 4 birim tren olmuştur. n-q hattının yeni akışı 13 tren, kapasitesi ise 60-13 = 47 birim tren olmuştur. q-r hattının akış değeri 13 tren, kapasitesi ise 18-13 = 5 birim tren olmuştur. r-z hattının akış değeri 13 tren, kapasitesi ise 17-13 = 1 birim tren olmuştur. z-aa hattının akış değeri 13 tren, kapasitesi ise 19-13 = 6 birim tren olmuştur. aa-bb hattının akış değeri 13 tren, kapasitesi ise 32-13 = 19 birim tren olmuştur. Son olarak, yinelenen bb-cc hattının yeni değerleri ise 20-13 = 7 birim trenlik kapasite ve 13 birim trenlik akıştır.

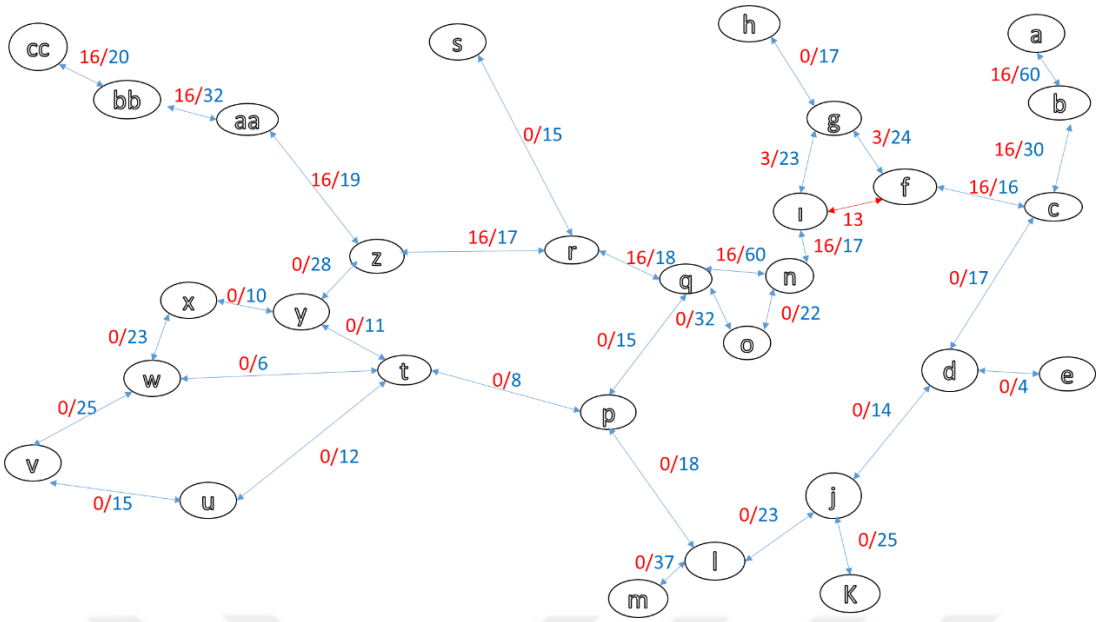
Algoritma adımları takip edilerek oluşan yeni arta kalan şebekede başlangıç düğümünden varış düğümüne yeni hat aranır.

Çizelge 5.7. İkinci iterasyon için a ’dan cc’ye giden hatlar

a→b→c→f→g→ı→n→q→r→z→aa→bb→cc
a→b→c→f→g→ı→n→o→q→r→z→aa→bb→cc
a→b→c→f→g→ı→n→o→q→p→t→y→z→ aa→bb→cc
a→b→c→f→g→ı→n→o→q→p→t→w→x→y→z→ aa→bb→cc
a→b→c→f→g→ı→n→o→q→p→t→u→v→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→q→r→z→aa→bb→cc
a→b→c→d→j→l→p→t→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→u→v→w→x→y→z→aa→bb→cc

Bu hatlar içerisinden bir tanesi seçilir.

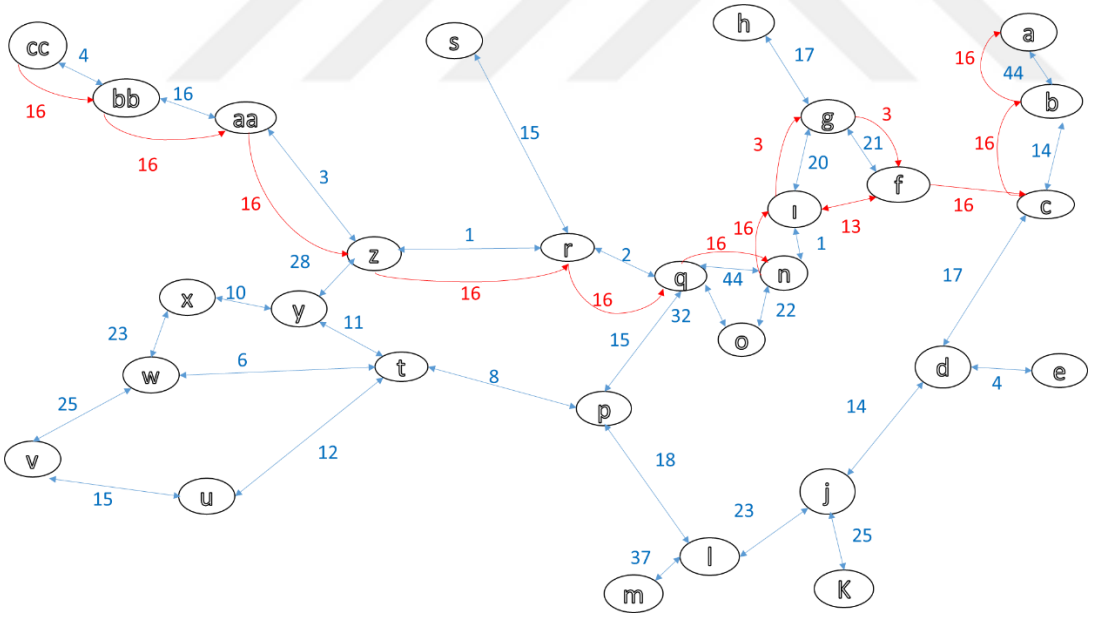
Yeni seçilen hat a→b→c→f→g→ı→n→q→r→z→aa→bb→cc olsun. Bu hattın kapasitesi min{47, 17, 3, 8, 23, 7, 4, 47, 5, 4, 6, 19, 7} = 3 birimdir.



Şekil 5.15. a-b-c-f-g-ı-n-q-r-z-aa-bb-cc yolu için arta kalan şebeke

(a,b), (b,c), (c, f), (f,g), (g,ı), (ı,n), (n,q), (q,r), (r,z), (z,aa), (aa,bb), (bb,cc) δ=3

Bir sonraki aşamada başlangıç grafiği yenilenir ve seçilen hattaki akış değerlerinde güncelleme yapılarak güncellenen grafiğe ait yeni bir arta kalan grafik hazırlanır.



Şekil 5.16. Maksimum akış ağ modeli (a-b-c-f-g-ı-n-q-r-z-aa-bb-cc yolu için)

Bir önceki artık grafikte olduğu üzere seçilen hat üzerindeki kapasite ve akış değerleri yenilenir. Grafiğe kazandırılan yeni akış sayesinde 47 birimlik kapasiteye sahip olan a-b hattının kapasitesi 47-3 = 44 birim trene düşerken 13 birimlik akış değeri 13+3 =16 birim trene yükselir. Aynı şekilde 17 birim trenlik kapasite b-c hattı için hesaplanan yeni değerde 17-3 = 14 birim trene düşerken akış değeri 13+3 =16 birim

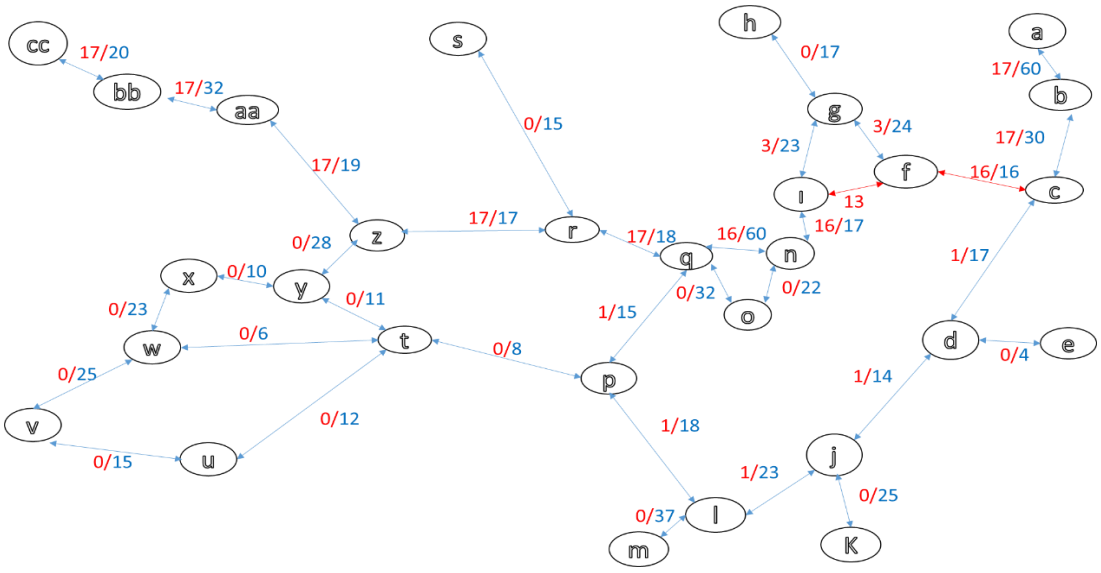
trene yükselir. c-f hattının kapasitesi 3-3 = 0 birim tren ve akışı 13+3 =16 birim tren olarak bulunur. f-g hattında kapasite 8-3 = 5 birim tren ve akış değeri 0+3 =3 birim tren olarak bulunur. g-ı hattında kapasite 7-3 = 4 birim tren ve akış değeri 0+3 =3 birim tren olarak bulunur. ı-n hattında kapasite 4-3 = 1 birim tren ve akış değeri 13+3 =16 birim tren olarak bulunur. n-p hattında kapasite 47-3 = 44 birim tren ve akış değeri 13+3 =16 birim tren olarak bulunur. q-r hattında kapasite 5-3 = 2 birim tren ve akış değeri 13+3 =16 birim tren olarak bulunur. r-z hattında kapasite 4-3 = 1 birim tren ve akış değeri 13+3 =16 birim tren olarak bulunur. z-aa hattında kapasite 6-3 = 3 birim trene düşerken akış değeri 13+3 =16 birim trene yükselir. Benzer şekilde aa-bb hattının yeni kapasite değeri 19-3 =16 birim tren olurken akış değeri 13+3 =16 birim tren olur ve son olarak bb-cc hattının 7 birim trenlik kapasitesi 7-3 =4 birim trene düşer ve akış değeri 13+3 =16 birim tren olarak bulunur. Algoritma adımları takip edilerek oluşan yeni arta kalan şebekede başlangıç düğümünden varış düğümüne yeni hat aranır.

Çizelge 5.8. Üçüncü iterasyon için a 'dan cc'ye giden hatlar

a→b→c→d→j→l→p→q→r→z→aa→bb→cc
a→b→c→d→j→l→p→t→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→u→v→w→x→y→z→aa→bb→cc

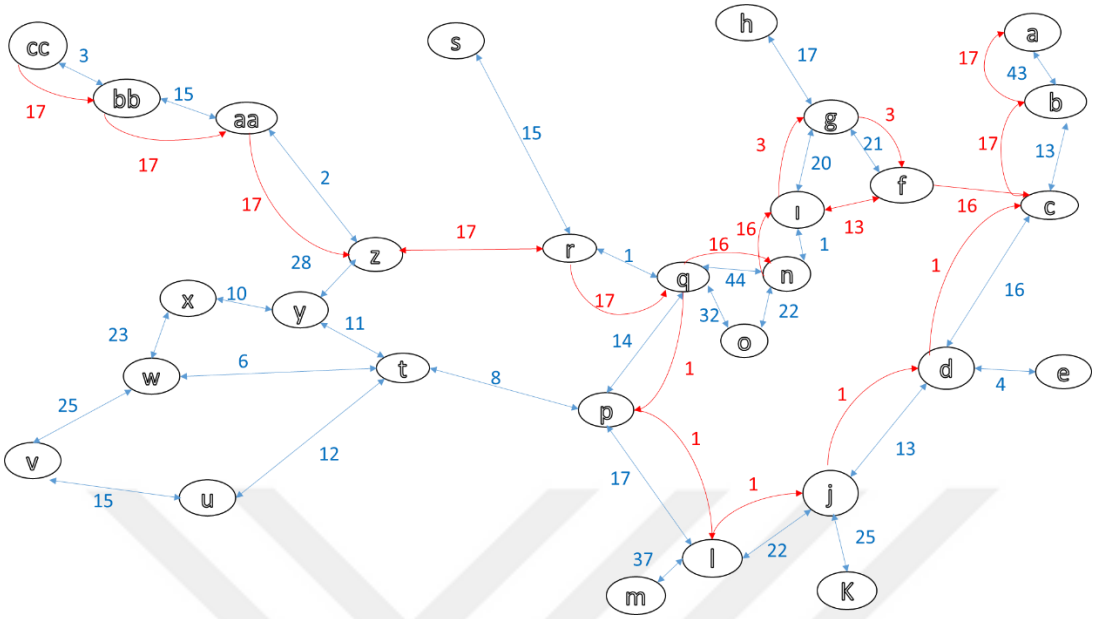
Bu hatlar arasından tekrar bir hat seçilir.

Yeni seçilen hat a→b→c→d→j→l→p→q→r→z→aa→bb→cc olsun. Bu yolun kapasitesi (a,b), (b,c), (c, d), (d,j), (j,l), (l,p), (p,q), (q,r), (r,z), (z,aa), (aa,bb), (bb,cc) δ=1 min{44, 14, 17, 14, 23, 18, 15, 2, 1, 3, 16, 4} = 1 birimdir.



Şekil 5.17. a-b-c-d-j-l-p-q-r-z-aa-bb-cc yolu için arta kalan şebeke

Bir ileriki adımda başlangıç grafiği yinelenir ve seçilen hattaki akış değerleri güncellenerek güncellenen grafiğe ait yeni bir arta kalan grafik hazırlanır.



Şekil 5.18. Maksimum akış ağ modeli (a-b-c-d-j-l-p-q-r-z-aa-bb-cc yolu için)

Bir önceki arta kalan şebeke gibi seçilen hat üzerindeki kapasite ve akış değerleri yenilenir. Grafiğe kazandırılan yeni akış sayesinde 44 birim trenlik kapasiteye sahip olan a-b hattının kapasitesi 44-1 = 43 birim trene düşerken 16 birim trenlik akış değeri 16+1 =17 birim trene yükselir. Aynı şekilde 14 birim trenlik kapasite b-c hattı için hesaplanan yeni değerde 14-1 = 13 birim trene düşerken akış değeri 16+1 =17 birim trene yükselir. c-d hattının kapasitesi 17-1 = 16 birim tren ve akış değeri 0+1 =1 birim tren olarak bulunur. d-j hattının kapasitesi 14-1 = 13 birim tren ve akışı 0+1 =1 birim tren olarak bulunur. j-l hattının kapasitesi 23-1 = 22 birim tren ve akışı 0+1 =1 birim tren olarak bulunur. l-p hattının kapasitesi 18-1 = 17 birim tren ve akış değeri 0+1 =1 birim tren olarak bulunur. p-q hattının kapasitesi 15-1 = 14 birim tren ve akış değeri 0+1 =1 birim tren olarak bulunur. q-r hattının kapasitesi 2-1 = 1 birim tren ve akış değeri 16+1 =17 birim tren olarak bulunur. r-z hattının kapasitesi 1-1 = 0 birim tren ve akışı 16+1 =17 birim tren olarak bulunur. z-aa hattının kapasitesi 3-1 = 2 birim trene düşerken akış değeri 16+1 =17 birim trene yükselir. Benzer şekilde aa-bb hattının kapasitesi 16-1 =15 birim tren olurken akış değeri 16+1 =17 birim tren olur ve son aşamada bb-cc hattının sahip olduğu 4 birim trenlik kapasite 4-1 =3 birim trene düşer ve akış değeri 16+1 =17 birim tren olarak bulunur.

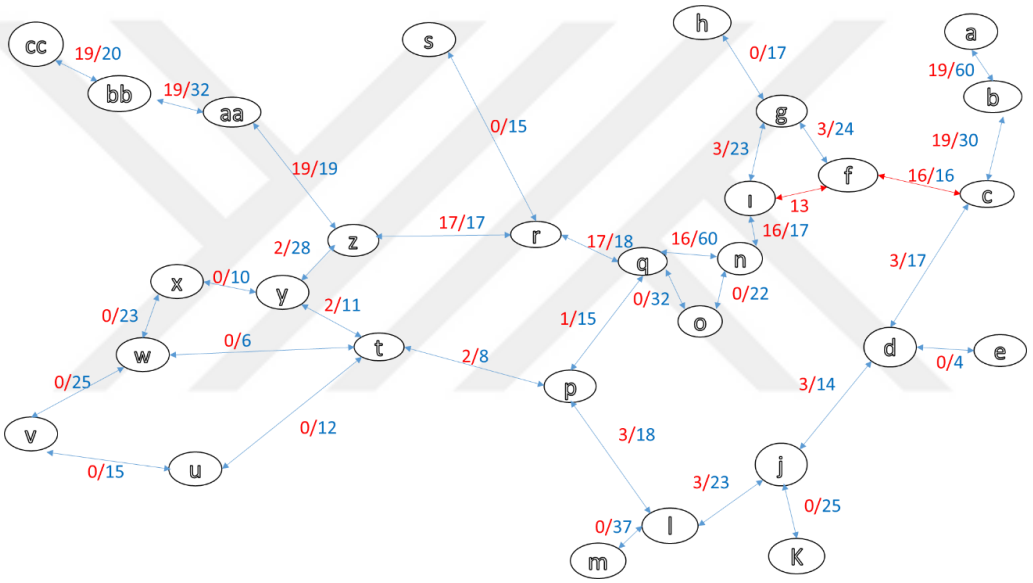
Algoritma adımları takip edilerek oluşan yeni arta kalan şebekede başlangıç düğümünden varış düğümüne yeni hatlar aranır.

Çizelge 5.9. Son iterasyon için a 'dan cc'ye giden hatlar

a→b→c→d→j→l→p→t→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→w→x→y→z→aa→bb→cc
a→b→c→d→j→l→p→t→u→v→w→x→y→z→aa→bb→cc

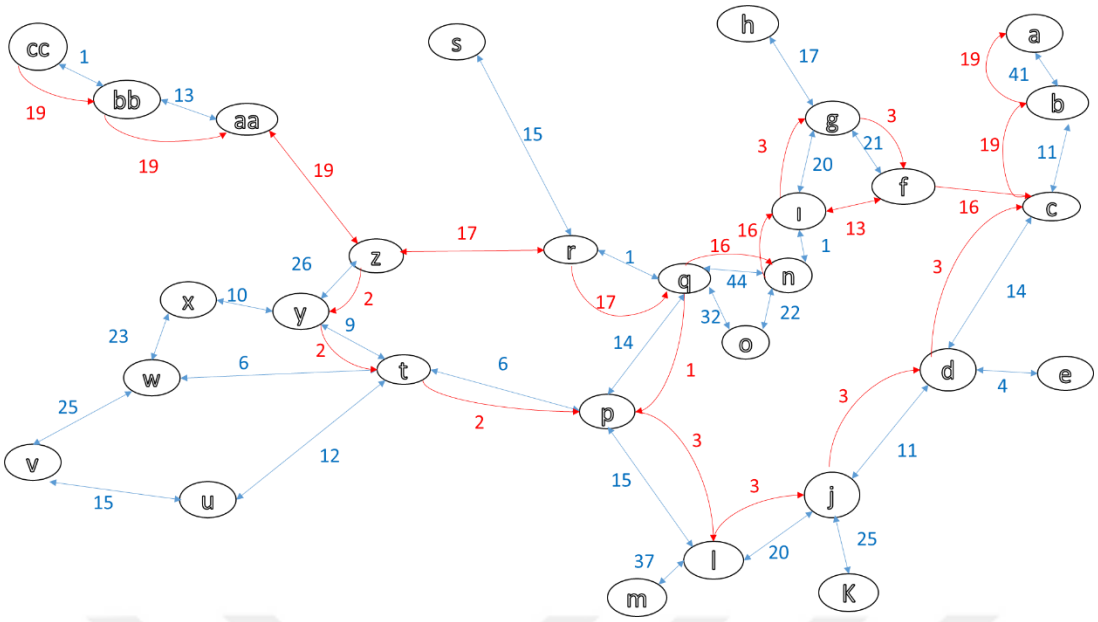
Bu hatlar arasından yine bir tanesi seçilir.

Yeni seçilen hat a→b→c→d→j→l→p→t→y→z→aa→bb→cc olsun. Bu hattın kapasitesi (a,b), (b,c), (c, d), (d,j), (j,l), (l,p), (p,t), (t,y), (y,z), (z,aa), (aa,bb), (bb,cc) δ=2 min{43, 13, 16, 13, 22, 17, 8, 11, 28, 2, 15, 3} = 2 birimdir.



Şekil 5.19. a-b-c-d-j-l-p-t-y-z-aa-bb-cc yolu için arta kalan şebeke

Bir ileriki adımda başlangıç grafiği yinelenir ve seçilen hattaki akış değerleri güncellenerek güncellenen grafiğe ait yeni bir arta kalan grafik hazırlanır.



Şekil 5.20. Trenlerin sevkinde hesaplanan maksimum akış ağ modeli

Bir önceki arta kalan şebeke gibi seçilen hat üzerindeki kapasite ve akış değerleri yenilenir. Şebekeye kazandırılan yeni akış sayesinde 43 birim trenlik kapasiteye sahip olan a-b hattının kapasitesi 43-2 = 41 birim trene düşerken 17 birim trenlik akış değeri 17+2 =19 birim trene yükselir. Benzer şekilde 13 birim trenlik kapasite b-c hattı için hesaplanan yeni değerde 13-2 = 11 birim trene düşerken akış değeri 17+2 =19 birim trene yükselir. c-d hattında kapasite 16-2 = 14 birim tren ve akış değeri 1+2 =3 birim tren olarak bulunur. d-j hattında kapasite 13-2 = 11 birim tren ve akış değeri 1+2 =3 birim tren olarak bulunur. j-l hattında kapasite 22-2 = 20 birim tren ve akış değeri 1+2 =3 birim tren olarak bulunur. l-p hattında kapasite 17-2 = 15 birim tren ve akış değeri 1+2 =3 birim tren olarak bulunur. p-t hattında kapasite 8-2 = 6 birim tren ve akış değeri 0+2 =2 birim tren olarak bulunur. t-y hattında kapasite 11-2 = 9 birim tren ve akış değeri 0+2 =2 birim tren olarak bulunur. y-z hattında kapasite 28-2 = 26 birim tren ve akış değeri 0+2 =2 birim tren olarak bulunur. z-aa hattında kapasite 2-2 = 0 birim trene düşerken akış değeri 17+2 =19 birim trene yükselir. Aynı şekilde aa-bb hattının yeni kapasite değeri 15-2 =13 birim tren olurken akış değeri 17+2 =19 birim tren olur ve son olarak bb-cc hattının sahip olduğu 3 birim trenlik kapasite 3-2 =1 birim trene düşer ve akış değeri 17+2 =19 birim tren olarak bulunur.

Yeni hat bulmak için son artık grafik incelenir ve başlangıç düğümünden bitiş düğümüne yeni hat ile ulaşılamadığı için algoritma bu noktada tamamlanır.

Algoritmanın tamamlandığı bu noktada maksimum akış hesaplanır. Bu değer ağ modelinde başlangıç düğümünden akan değerler toplamına ya da bitiş düğümünden çıkan değer toplamına eşittir. a düğümüne giren değerler toplamı 13+3+1+2 =19 birim tren cc düğümünden çıkan değerlerin toplamı 13+3+1+2 =19 birim trendir.

Şebekedeki yollardan birinci adımda ilk iterasyonda en kısa yol olan a→b→c→f→ı→n→q→r→z→aa→bb→cc yolu seçilmiş ve seçilen yolun maksimum kapasitesi olan 13 tren bu güzergâhtan işletilmiştir. Devamında ikinci iterasyonda a→b→c→f→g→ı→n→q→r→z→aa→bb→cc seçilen bu yoldan 3 trenlik bir akış sağlanmıştır. Üçüncü iterasyonda seçilen a→b→c→d→j→l→p→q→r→z→ aa→bb→cc yoldan 1 trenlik akış sağlanmış olup dördüncü ve son iterasyonda ise a→b→c→d→j→l→p→t→y→z→aa→bb→cc yolundan 2 trenlik bir akış sağlanmış ve son güncel artık grafik incelendiğinde başlangıç noktasından bitiş noktasına ulaşılamadığı için algoritma son iterasyonda tamamlanmıştır.

Sonuç olarak demiryolu tren işletmecisi kaynak a noktasından varış/batak cc noktasına günde en fazla 19 tren çalıştırabilme imkânına sahip olacağı sonucuna varılmıştır. Yani mevcut durumda bir tren işletmecisi kaynak a noktasından varış/batak cc noktasına günde en fazla 2 tren çalıştırabiliyorken gelecek projeksiyonunda ise 19 tren çalıştırabilme kabiliyetine sahip olacaktır. Böylece yapılan çalışma ile tren sefer sayılarında mevcut duruma göre yaklaşık 10 kat artış sağlanmış olacaktır.

Ford-Fulkerson algoritması Java programında kodlanmış ve elde edilen sonuçlar Şekil 5.21'de verilmiştir.

```java
import java.util.LinkedList;
import java.util.Queue;

public class MaxFlow_Ford_Fulkerson {
    static class Graph {
        int vertices;
        int graph[][];

        public Graph(int vertex, int[][] graph) {
            this.vertices = vertex;
            this.graph = graph;
        }

        public int findMaxFlow(int source, int sink) {
            //residual graph
            int[][] residualGraph = new int[vertices][vertices];

            //initialize residual graph same as original graph
            for (int i = 0; i <vertices ; i++) {
                for (int j = 0; j <vertices ; j++) {
                    residualGraph[i][j] = graph[i][j];
                }
            }

            //initialize parent [] to store the path Source to destination
            int [] parent = new int[vertices];

          int max_flow = 0; //initialize the max flow

          while(isPathExist_BFS(residualGraph, source, sink, parent)){
              //if here means still path exist from source to destination

              //parent [] will have the path from source to destination
              //find the capacity which can be passed though the path (in parent[])

              int flow_capacity = Integer.MAX_VALUE;

              int t = sink;
              while(t!=source){
                  int s = parent[t];
                  flow_capacity = Math.min(flow_capacity, residualGraph[s][t]);
                  t = s;
              }

              //update the residual graph
              //reduce the capacity on fwd edge by flow_capacity
              //add the capacity on back edge by flow_capacity
              t = sink;
              while(t!=source){
                  int s = parent[t];
                  residualGraph[s][t]-=flow_capacity;
                  residualGraph[t][s]+=flow_capacity;
                  t = s;
              }
```

Şekil 5.21. Ford-Fulkerson algoritması ile çözüm sonuçları

```
55
56          //add flow_capacity to max value
57          max_flow+=flow_capacity;
58      }
59      return max_flow;
60  }
61
62  public boolean isPathExist_BFS(int [][] residualGraph, int src, int dest, int [] parent){
63      boolean pathFound = false;
64
65      //create visited array [] to
66      //keep track of visited vertices
67      boolean [] visited = new boolean[vertices];
68
69      //Create a queue for BFS
70      Queue<Integer> queue = new LinkedList<>();
71
72      //insert the source vertex, mark it visited
73      queue.add(src);
74      parent[src] = -1;
75      visited[src] = true;
76
77      while(queue.isEmpty()==false){
78          int u = queue.poll();
79
80          //visit all the adjacent vertices
81          for (int v = 0; v <vertices ; v++) {
82                  //if vertex is not already visited and u-v edge weight >0
83                  if(visited[v]==false && residualGraph[u][v]>0) {
84                      queue.add(v);
85                      parent[v] = u;
86                      visited[v] = true;
87                  }
88              }
89          }
90          //check if dest is reached during BFS
91          pathFound = visited[dest];
92          return pathFound;
93      }
94  }
95
```

Şekil 5.21. Ford-Fulkerson algoritması ile çözüm sonuçları (devamı)

```
public static void main(String[] args) {
    int vertices = 29;
    int graph[][] =
{ {0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{60, 0, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 30, 0, 17, 0, 16, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 17, 0, 4, 0, 0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 16, 0, 0, 0, 24, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 24, 0, 17, 23, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 13, 23, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 14, 0, 0, 0, 0, 0, 0, 25, 23, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 0, 0, 37, 0, 0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 37, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 22, 0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 0, 0, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 0, 0, 0, 0, 15, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 60, 32, 15, 0, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 0, 15, 0, 0, 0, 0, 0, 17, 0, 0, 0},
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 0, 0, 0, 0, 12, 0, 6, 0, 11, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 0, 15, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 0, 25, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 25, 0, 23, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 0, 10, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 23, 0, 10, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 0, 0, 0, 10, 0, 28, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0, 28, 0, 19, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 0, 32, 0},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32, 0, 20},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20, 0}
    };
    Graph g = new Graph(vertices, graph);
    int source = 0;
    int destination = 28;
    int max_flow = g.findMaxFlow(source,destination);
    System.out.println("Maximum flow from source: " + source + " to destination: " + destin
            + " is: " + max_flow);
}
```

Şekil 5.21. Ford-Fulkerson algoritması ile çözüm sonuçları (devamı)

Ford-Fulkerson algoritması Şekil 5.21'de görüldüğü gibi Java kodlama dilinde gelecek projeksiyonu ele alınarak kodlanarak çözüldüğünde 29 farklı destinasyonun yer aldığı şebeke için maksimum akış 19 tren gün olarak bulunmuştur.

Output:

```
Maximum flow from source: 0 to destination: 28 is: 19
```

Time Taken: 0.09 seconds

Memory Used: 24392 KB

Şekil 5.22. Ford-Fulkerson algoritması çıktıları

Bu işlem Şekil 5.22'de yer aldığı üzere Java uygulamasında 0.09 saniyede çözüme kavuşturulmuş olup bellekte ise 24392 kilobayt yer aldığı görülmektedir.

Java uygulama sonuçlarına bakıldığında 0.08 saniye olan işlem süresinin 0.09 saniye olduğu ayrıca bellekte ise 24312 kilobaytlık bellek yerine 24392 kilobayt bellekte yer kapladığı görülmektedir. Programsal olarak 0,01 saniyelik işlem ve 80 kilobayt bellek yoğunluğunda artış gözlenmiştir. Şebekenin maksimum akış durumu incelendiğinde iki olan sefer sayısında on katlık bir artış sağlandığı görülmüştür.

6. SONUÇ, DEĞERLENDİRME ve ÖNERİLER

Asya'dan Avrupa'ya uzanan üç büyük ticaret yolu göz önüne alınarak bir değerlendirme yapıldığında güneyde Evergreen gemisinin Süveyş Kanalında karaya oturması dünya deniz ticaretine bir haftalık süre içerisinde milyarlarca dolar zarar verdiği bilinmektedir. Kuzeyde ise Rusya-Ukrayna savaşı ve Rusya AB ilişkileri gibi nedenlerden dolayı Asya-Avrupa arasında alternatif rota bulmanın önemi daha da artmaktadır. Ayrıca alternatif koridorlardan olan; Rusya üzerinden geçen Kuzey koridoru üzerinde yaşanan savaşın belirsizliği gibi olumsuz gelişmeler de orta koridorun avantajını gün yüzüne çıkartmıştır. Özellikle savaş gibi nedenlerle Asya-Avrupa arasındaki taşımacılığın başka hat üzerinden sağlanması alternatifi ulaştırma sektöründe talep görmektedir. Burada iki konunun ön plana çıktığını görmekteyiz. İlki bütün bir hat kesiminin sadece ufak bir bölümünde yaşanan aksamanın tüm hat kesimini olumsuz etkilediğidir. İkincisi ise günümüzde zaman ve emniyet kavramlarının öneminin çok daha fazla hissedilmeye başlandığı süreçte kullanılmakta olan güzergâh üzerinde yaşanan bir kriz nedeniyle alternatif güzergâhların ön plana çıkmaya başlamasıdır. Çünkü tedarik sürecinde yaşanan aksaklığı sıfır stok anlayışıyla çalışan sektörlerin bir krizi tahammül edecek durumu bulunmamaktadır. Bu itibarla orta koridorun bir parçası olan Ahilkelek-Kapıkule hat kesimi üzerine optimizasyon çalışmalarının yapılması kaçınılmaz bir hale gelmektedir. Tam bu noktada yüklerin taşındığı demiryolu hat kapasitesinin maksimum sevide kullanılması gerek taşıma maliyetlerinde iyileştirme gerekse de yüklerin gideceği noktaya en kısa sürede ulaşmasını sağlayarak toplam taşıma zamanının iyileştirilmesinde en kısa yol algoritması ile maksimum akış optimizasyonunun birlikte kullanılarak çalışmanın literatüre zenginlik katması hedeflenmiştir.

Literatürde yük taşımacılığına ilişkin optimizasyon çalışmaları olmasına rağmen, demiryolu yük taşımacılığında en kısa yol ve maksimum akış durumları beraber dikkate alınmamıştır. Bu bağlamda, maksimum akış problemlerinin uygulamalarında avantajları nedeniyle sıklıkla tercih edilen ancak Kars-Kapıkule hat kesiminde ilk kez bu çalışmada kullanılan en kısa yol Floyd-Warshall ve maksimum akış Ford-Fulkerson algoritması birlikte kullanılarak yenilikçi bir çözüm yaklaşımı sunulmuştur. Ele alınan hat kesimi için en kısa yol Şekil 5.4' te ayrıca en kısa yola ait maliyet değerleri ise Şekil 5.6'da yer almaktadır. Programlama dillerindeki algoritmaların zaman

karmaşıklık durumu göz önüne alınarak, hedef düğüm için başlangıçta konumlandırılmış mümkün olan en yüksek yük akış miktarı mevcut durum analizi Şekil 5.10'da yer alan çözümden yola çıkılarak nihai sonucuna Şekil 5.11'de yer verilmiştir. Ayrıca Şekil 5.21'de çözümü yer alan ve sonucu Şekil 5.22'de görüldüğü üzere gelecek projeksiyonunda maksimum akışın ne kadar olduğu hesaplanmıştır. Özellikle uygulamada tercih edilen programlama dili sayesinde 0,09 saniyelik gibi kısa bir işlem süreci sonunda çözüme ulaşıldığı da unutulmamalıdır. Bu durum literatürde adı geçen diğer çalışmalardan farkı açıkça ortaya koymaktadır.

Demiryolu şebeke optimizasyonu konusunda yapılan bu çalışmalar gelecekte yapılması düşünülen veya hali hazırda ihtiyaç duyulan çalışmalara yönelik bir öngörü oluşturmaktadır. Belirlenen düğümler ve dallar çalışma özelliğine ve ilgilinin hedeflerine göre farklılık gösterebilmektedir. Bu itibarla ileride yapılacak ulaştırma ve lojistik projeleri için farklı şebekeler göz önüne alınarak bu gibi optimizasyon çalışmaları yapılması önerilmektedir.

KAYNAKLAR

[1] Altay, N. ve Green, W. G. (2006). OR/MS research in disaster operations management. European Journal of Operational Research. 175(1), 475–493.

[2] Akkaymak, M. (2009). Avrupa-Asya Ulaştırma Koridorları ve Yeniden Canlanan İpek Yolu. Yüksek Lisans Tezi, İstanbul.

[3] Türkiye Cumhuriyeti Devlet Demiryoları İşletmesi Genel Müdürlüğü (2021). Demiryolu Sektör Raporu 2021 https://static.tcdd.gov.tr/ İndirilme Tarihi: 24.11.2022

[4] Uğur, A. (2019). Investigation of the world railway sector development prospects and Turkey's status. The Journal of Operations Research, Statistics, Econometrics and Management Information Systems, 7(2), 369-398.

[5] Nash, C. and Rivera-Trujillo, C. (2004). Principles and practice. Rail regulatory reform in Europe. Institute for Transport Studies, 4.

[6] Anonim (2021). 12.Ulaştırma ve Haberleşme Şurası Sektör Öngörü Raporu, Ulaştırma ve Altyapı Bakanlığı, Ankara

[7] Einar, S. (2020). Introduction to the tools of scientific computing. S. Einar (Edit.), Python, the fundamentals (pp. 19–50). Germany: Springer.

[8] Ford, L.R. and Fulkerson, D.R. (1962). *Flows in network.* New Jersey: Princeton University Press.

[9] Ford, L.R. and Fulkerson, D.R. (1956). Maximal flow through a network, Canadian Journal of Mathematics, 8, 399-404.

[10] Fulkerson, D.R. and Dantzing G.B. (1955). Computation on maximal flow in networks, Naval Research Logistic Quarterly, 2, 277-283.

[11] Ahuja, R.K., Magnanti, T.J. and Orlin, J.B. (1993). *Network flows*, New Jersey: Prentice Hall Press.

[12] Bazaraa, M.S., Jarvis J.J. and Sherali, H.D. (2005) Linear programming and network flows, New Jersey: A John Wiley & Sons, Inc., Publication.

[13] Bellman, R.E. (1958). On a routing problem., The Quarterly of Applied Mathematics, 16, 87–90.

[14] Warshall, S. (1962). A theorem on Boolean matrices. Journal of the ACM, 9(1), 11–12.

[15] Floyd, R. W. (1956). Algorithm 97: shortest path. Communications of the ACM, 5, 345.

[16] Johnson, D. B. (1977). Efficient algorithms for shortest paths in sparse networks. Journal of the Association for Computing Machinery, 24, 1-13.

[17] Keskin, B. and Özcan, E. (Basımda). En Kısa Yol Optimizasyonlarında Floyd-Warshall Algoritması: Lojistik Merkezler Örneği. Demiryolu Mühendisliği Dergisi, doi: 10.47072/demiryolu.1187884

[18] Wang, Y. and Lu, C. (2019). Railway route optimisation from Romania to Poland based on an analysis of China' s ' One Belt and One Road ' initiative. International Journal of Internet Manufacturing and Services, 6(1), 1-18.

[19] Pandey, S.D. (2014). Railway route optimization system using Dijkstra Method. Int. J. Recent Innov. Trends Comput. Commun., 2(3), 435-440.

[20] Liu, L., Xia, Y. and Han, Y. (2010). Research on three-dimensional modelling of railway route in railway route selection. Int. Conf. Mech. Autom. Control Eng. 1, 2907-2911.

[21] Wang, L., Jia, L. M., Qin, Y., Xu, J., and Mo, W.T. (2011). A two-layer optimization model for high-speed railway line planning. J. Zhejiang Univ. Sci. A, 12(12), 902-912.

[22] Zhang, H., Yuan, M., Liang, Y., Wang, B., Zhang, W., and Zheng, (2018). J. A risk assessment based optimization method for route selection of hazardous liquid railway network. Railway safety, 110, 217–229.

[23] Kosijer, M., Ivic, M., Markovic, M., and Belosovic, I. (2012). Multicriteria decision-making in railway route planning and design. Građevinar, 64, 195-205.

[24] M.Y. Kankavi, (2019). Demir ipekyolunda Türkiye geçişi için en uygun güzergâh seçimi. Yüksek Lisans Tezi. Maltepe Üniversitesi, İstanbul.

[25] Saat, M.R., and Serrano, J.A. (2015). Multicriteria high-speed rail route selection: application to Malaysia's high-speed rail corridor prioritization. Transp. Plan. Technol., 38(2), 200–213.

[26] Özdemir, S. (2021). Modern ipek yolu koridorlarında rota optimizasyonu için hibrit model önerisi. Yüksek Lisans Tezi. Kırıkkale Üniversitesi, Fen Bilimleri Enstitüsü, Kırıkkale.

[27] Dermawan, T.S. (2019). Comparison of dijkstra dan floyd-warshall algorithm to determine the best route of train. IJID (International Journal on Informatics for Development), 7(2), 54-58.

[28] Özdemir, S., Sacar, Ö. ve Özcan, E. (2021). Dijkstra algoritması kullanılarak ipek yolu koridorları arasında en kısa ulaştırma güzergâhının belirlenmesi. Demiryolu Mühendisliği Dergisi, 13, 97-105.

[29] Pradhan, A. and Mahinthakumar, G. (2013). Finding all-pairs shortest path for a large-scale transportation network using parallel Floyd-Warshall and parallel

Dijkstra algorithms. Journal of computing in civil engineering, 27(3), 263-273.

[30] Hamurcu, M. ve Eren, T. (2019). An Application of multicriteria decision-making for the evaluation of alternative monorail routes. Mathematics, 7(1), 16.

[31] Hanzl, J., Bartuška, L., Rozhanskaya, E. and Průša, P. (2016). Application of floyd's algorithm on transport network of south bohemian region. Komunikácie: Communications (Scientific Letters of the University of Žilina), 18(2).

[32] Yanwei, Z., Gengyu, W., Fangzhi, G,. Chen, X., R. Shedong, Zhiwei, R. and Zhiwei, X. (2019). Optimal coordination path selecting method for conduction transformation based on floyd algorithm. Procedia computer science, 162, 227-234.

[33] Pandika, I.K.L.D., Irawan, B. and Setianingsih, C., (2018). Application of optimization heavy traffic path with floyd-warshall algorithm. In 2018 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), 57-62.

[34] Risald, A.S. (2017). Best routes selection using dijkstra and floyd-warshall algorithm. 11th International Conference on Information & Communication Technology and System (ICTS), Indonesia.

[35] Triana, Y.S. and Syahputri, I. (2018). Implementation floyd-warshall algorithm for the shortest path of garage. International journal of innovative science and research technology, 3(2), 871-878.

[36] Tang, K., Pan, C. and Qian, M. (2019). Manufacturing/remanufacturing logistics network optimization based on floyd algorithm. In journal of physics: conference series, 1288(1), 012026.

[37] Danışan, T., Özcan, E. ve Eren, T. (2021). Bakım ekiplerinin en kısa yoldan santrallara ulaşımı: hidroelektrik santral örneği. Journal of Turkish Operations Management, 576-587.

[38] Ramadhan, Z., Siahaan, A.P.U. and Mesran, M. (2018). Prim and floyd-warshall comparative algorithms in shortest path problem. In proceedings of the joint workshop ko2pi and the 1st ınternational conference on advance & scientific innovation, 47-58.

[39] Çakır, M.E., Yetiş, A.D., Yeşilnacar, M.İ. ve M. Ulukavak, (2019). Katı atıklar için optimum güzergâh tespiti ve alansal dağılım haritalarının cbs ortamında oluşturulması: Suruç (Şanlıurfa) örneği. BEÜ Fen Bilimleri Dergisi BEU Journal of Science, 8(2), 595-603.

[40] Sungkwan, K., Hojun, J., Minah, S. and Dongsoo, H. (2019). Optimal path planning of automated guided vehicle using dijkstra algorithm under dynamic conditions. 1th International Conference on Robot Intelligence Technology and Applications (RiTA) Robot Intelligence Technology and Applications Robot Intelligence Technology and Applications, Daejeon, Korea.

[41] E.D. Ekmen,. (2020). A Study On Performance Evaluatıon Of Optımızatıon Algorıthms In The Shortest Path Problem. Yüksek Lisans Tezi. Yıldırım Beyazıt Üniversitesi, Ankara.

[42] Magzhan, K. and Jani, H. (2013). A Review and evaluations of shortest path algorithms. Int. J. Sci. Technol. Res., 2(6), 99–104.

[43] Tamimi, A. A. (2015). Comparison studies for different shortest path algorithms. Int. J. Comput. Technol., 14(8), 5979–5986.

[44] Golden, B. (1976). Shortest-Path Algorithms: A Comparison." Operations Research, 24(6).

[45] Alkan, M. ve Aydin,M. (2019) Simulation and comparison of pathfinding algorithms using real Turkey data. Int. Conf. Artif. Intell. Data Process. (IDAP 2018).

[46] Djojo, M. A. and Karyono, K. (2013). Computational load analysis of Dijkstra, A*, and Floyd-Warshall algorithms in mesh network. Proc. 2013 Int. Conf. Robot. Biomimetics, Intelligent Computational Systems. Tokyo, Japan, 01 November

[47] Wang, X. Z. (2018). The comparison of three algorithms in shortest path issue. Journal of Physics: Conference Series 1087(2), 1-6.

[48] Çam, Ö. N. and Sezen, H. K. (2018). Toplam bekleme süresini enküçükleme amaçlı bir araç rotalama problemi, 11(2), 47–60.

[49] Timör, M. (2005). Medyan en kısa yol problemi: maliyet erişilebilirlik hedeflerine yönelik bir çok amaçlı taşımacılık problemi uygulaması. İstanbul Üniversitesi İşletme Fakültesi Dergisi, 34(2), 31–56.

[50] Fendoğlu, E. and Söyler, H. (2017). Route optimization of Malatya metropolitan municipality pesticide vehicles. Alphanumeric Journal, 6(1), 13-24.

[51] Nuriyeva, F. ve Kizilates, G. (2016). Gezgin satıcı problemi için merkezden kenarlara hipersezgisel yöntem. Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 20(2), 319–323.

[52] Ö. Saçar,. (2018). İpek Yolu Güzergâhında Yapılan Lojistik Etkinliklerin Günümüz Lojistik Faaliyetleri İle Karşılaştırılması. Yüksek Lisans Tezi. Balıkesir Üniversitesi, Balıkesir.

[53] Özdemir, S., Keskin, B. Eren, T. ve Özcan. E. (2020). Türkiye'deki lojistik merkezleri yatırım önceliklerinin değerlendirilmesinde çok kriterli karar modeli önerisi. Demiryolu Mühendisliği Dergisi, 12, 83–94.

[54] Abdullah, N. and Hua, T.K. (2017). The application of the shortest path and maximum flow with bottleneck in traffic flow of kota Kinabalu. Journal of

Computer Science & Computational Mathematics, 7 (2) 37–43.

[55] Rajalakshmi, V. and Ganesh Vaidyanathan, S. (2019). Efficient traffic management on road network using Edmonds–Karp algorithm. Progress in Advanced Computing and Intelligent Engineering, 577–583.

[56] Luo, Z., Dubey, R., Papadopoulos, T., Hazen, B. and Roubaud, D. (2018). Explaining environmental sustainability in supply chains using graph theory. Computational Economics 52, 1257–1275.

[57] Jia, C. and Zhang, C. (2019). Joint optimization of maintenance planning and workforce routing for a geographically distributed networked infrastructure. IISE Transactions, 52(2), 732–750.

[58] Surakhi, O.M., Qatawneh, M. and Al Ofeishat, H.A. (2017). A parallel genetic algorithm for maximum flow problem. International Journal of Advanced Computer Science and Applications, 8(6), 159-164.

[59] Paithankar, A. and Chatterjee, S. (2019). Open pit mine production schedule optimization using a hybrid of maximum-flow and genetic algorithms. Applied Soft Computing, 81, 1-16.

[60] Than Kyi, M. and Naing, L.L. (2018). Application of Ford-Fulkerson algorithm to maximum flow in water distribution pipeline network. International Journal of Scientific and Research Publications, 8(12), 306–310.

[61] Jiang, Z., Hu, X. and Gao, S. (2013). A parallel Ford-Fulkerson Algorithm for maximum flow problem. Chinese Academy of Sciences, 70-73.

[62] Than Kyi, M., Maw, S.S. and Naing, L.L. (2019). Mathematical estimation for maximum flow in electricity distribution network by Ford-Fulkerson iteration algorithm. International Journal of Scientific and Research Publications, 9(8), 192–196.

[63] Bulut, M. and Ozcan, E. (2021). Optimization of electricity transmission by Ford-Fulkerson algorithm, Sustainable Energy, Grids and Networks, 28.

[64] T. Önder,. (2007). Türkiye' de taşımacılık sektörünün lojistik olgusu içerisinde incelenmesi. Yüksek Lisans Tezi. Kadir Has Üniversitesi, İstanbul.

[65] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs, Numerische Mathematik, 1(1), 269–271.

[66] Zieyel, E. R. (1988). Operations Research: Applications and Algorithms. Technometrics, 30(3), 361–362.

[67] Even, S. (2011). *Graph Algorithms(2nd edition).* Cambridge University Press.

[68] Türkiye Cumhuriyeti Devlet Demiryoları İşletmesi Genel Müdürlüğü Şebeke Bildirimi 2021. https://static.tcdd.gov.tr/ İndirilme Tarihi: 24.11.2021.

[69] Türkiye Cumhuriyeti Devlet Demiryoları İşletmesi Genel Müdürlüğü Yük Taşıma Fiyatları 2021. https://portal1.tcddtasimacilik.gov.tr/ İndirilme Tarihi: 14.10.2021.

ÖZGEÇMİŞ

Adı Soyadı :Bekir KESKİN

Doğum Tarihi : 1986

Yabancı Dil :

Eğitim Durumu :

(Kurum ve Yıl)Lisans : İstanbul Üniversitesi: Ulaştırma ve Lojistik Hoca Ahmet Yesevi Ulus. Türk-Kazak Üniversitesi: Endüstri Mühendisliği

Çalıştığı Kurum/Kurumlar ve Yıl/Yıllar : Türkiye Cumhuriyeti Devlet Demiryolları Genel Müdürlüğü (2011-Halen devam etmekte)

Yayınları :

Özdemir, S., Keskin, B., Eren, T. ve Özcan. E. (2020). Türkiye'deki lojistik merkezleri yatırım önceliklerinin değerlendirilmesinde çok kriterli karar modeli önerisi. Demiryolu Mühendisliği Dergisi, 12, 83–94.

Keskin, B. and Özcan, E. (Basımda). En Kısa Yol Optimizasyonlarında Floyd-Warshall Algoritması: Lojistik Merkezler Örneği. Demiryolu Mühendisliği Dergisi, doi: 10.47072/demiryolu.1187884

Araştırma Alanları: Demiryollarında şebeke çalışmaları