T.C.
MANİSA CELAL BAYAR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ
YAZILIM MÜHENDİSLİĞİ ANABİLİM DALI
YAZILIM MÜHENDİSLİĞİ BİLİM DALI

AĞLARDA ZEDELENEBİLİRLİK VE AYRIT PARAMETRELERİ ÜZERİNE

Ramazan SÜRÜCÜ

Danışman
Doç. Dr. Ersin ASLAN

MANİSA–2022

TEZ ONAYI

Ramazan SÜRÜCÜ tarafından hazırlanan **"Ağlarda Zedelenebilirlik ve Ayrıt Parametreleri Üzerine"** adlı tez çalışması 10/08/2022 tarihinde aşağıdaki jüri üyeleri önünde Manisa Celal Bayar Üniversitesi Fen Bilimleri Enstitüsü **Yazılım Mühendisliği Anabilim Dalı**'nda **YÜKSEK LİSANS TEZİ** olarak savunulmuş ve **oybirliği** ile başarılı olarak kabul edilmiştir.

Danışman	**Doç. Dr. Ersin ASLAN** Manisa Celal Bayar Üniversitesi	.....
Jüri Üyesi	**Prof. Dr. Vecdi AYTAÇ** İzmir Ege Üniversitesi	.....
Jüri Üyesi	**Dr. Öğr. Üyesi Yusuf ÖZÇEVİK** Manisa Celal Bayar Üniversitesi	.....

TAAHHÜTNAME

Bu tezin Manisa Celal Bayar Üniversitesi Hasan Ferdi Turgutlu Teknoloji Fakültesi Yazılım Mühendisliği'nde, akademik ve etik kurallara uygun olarak yazıldığını ve kullanılan tüm literatür bilgilerinin referans gösterilerek tezde yer aldığını beyan ederim.

Ramazan SÜRÜCÜ

İÇİNDEKİLER

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

G	Çizge
E(G)	Bir G çizgesinin ayrıtlar kümesi
V(G)	Bir G çizgesinin tepeler kümesi
n	Bir G çizgesinin tepe sayısına bağlı seviyesi
V	Bir çizgenin tepesi
E	Bir çizgenin ayrıtı
deg(V)	V tepesinin derecesi
es(G)	G çizgesinin ayrıt saçılma sayısı
r'(G)	G çizgesinin ayrıt kopma derecesi
I'(G)	Ayrıt bütünlüğü
m(G-S)	En büyük bileşenin tepe sayısı
ω(G-S)	Bileşen sayısını
|S|	Çıkarılan tepe kümesi
$\Delta(\boldsymbol{G})$	G çizgesine ait maksimum tepe derecesi
$\boldsymbol{\delta}$**(G)**	G çizgesine ait minimum tepe derecesi
α(G)	Bağımsızlık sayısı
λ(G)	Ayrıt bağlantı sayısı
$\boldsymbol{P_{n-1}}$	n-1 adet tepeye sahip yol çizge
$\mathbf{K_n}$	n adet tepeye sahip tam çizge
$\mathbf{K_{1,n}}$	Yıldız çizge
$\mathbf{C_{n,m}}$	Kuyruklu yıldız
$\mathbf{K_{r,m}}$	Double Star

$\mathbf{B_{r,m}}$	Muz ağacı (Banana Tree)
$\boldsymbol{\beta}$**(G)**	Bağımsızlık sayısı (Indepence number)

TABLO DİZİNİ

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

TEŞEKKÜR

Lisans üstü eğitimim boyunca ve çalışmamın her aşamasında bana yardımcı olan, bilgi ve tecrübeleri ile yol gösteren, desteğini hiç eksik etmeyen danışman hocam Sayın Doç. Dr. Ersin ASLAN'a ve öğrenim hayatımda maddi manevi destek veren ve her zaman yanımda olan aileme teşekkür ederim.

Ramazan SÜRÜCÜ
Manisa, 2022

ÖZET

Yüksek Lisans Tezi

Ağlarda Zedelenebilirlik ve Ayrıt Parametreleri Üzerine

Ramazan SÜRÜCÜ

Manisa Celal Bayar Üniversitesi Fen Bilimleri Enstitüsü
Yazılım Mühendisliği Anabilim Dalı

Danışman: Doç. Dr. Ersin ASLAN

Günlük yaşantımızda iletişim ve veri aktarımının önemi giderek artmaktadır. Bu iletişim ve veri aktarımını sağlayabilmenin bir yolu ise bilgisayarlar arası iletişim yani bilgisayar ağlarıdır. İletişimin verimli olabilmesi için kesintisiz, güvenilir ve hızlı olması istenir. Günümüzde karmaşık bir hal almış birçok problem modelleme yolu ile daha sade ve anlaşılır bir hale getirilebilir. Bu modeller çizge teorisinde önemli bir yer alan zedelenebilirlik parametreleri ile incelelenebilmekte ve irdelenebilmektedir. Bu tezde çizgelerdeki ayrıt parametreleri; Ayrıt Saçılma Sayısı (Edge Scattering Number), Ayrıt Kopma Derecesi (Edge Rupture Degree), Ayrıt Bütünlüğü (Edge Integrity) üzerine çalışılmıştır. Belirtilen parametrelerin her biri için algoritmalar oluşturulmuştur. Oluşturulan algoritmalar detaylıca açıklanmıştır.

Anahtar Kelimeler: Ayrıt Saçılma Sayısı, Ayrıt Bütünlüğü, Ayrıt Kopma Derecesi, Tepe, Ayrıt, Çizge Teorisi.

2022, 62 sayfa

ABSTRACT

M.Sc. Thesis

On Vulnerability and Edge Parameters in Networks

Ramazan SÜRÜCÜ

Manisa Celal Bayar University
Graduate School of Applied and Natural Sciences
Department of Software Engineering

Supervisor: Doç. Dr. Ersin ASLAN

The importance of communication and data transfer is gradually increasing in our daily life. In order for communication to be efficient, it is required to be uninterrupted, reliable and fast. Today, many problems that have become complex can be made simpler and more understandable by modeling. These models can be examined and studied with vulnerability parameters, which have an important place in graph theory. In this thesis, the edge parameters in graphs; It has been studied on Edge Scattering Number, Edge Rupture Degree, Edge Integrity. Algorithms were designed for each of the specified parameters. The algorithms designed are explained in detail.

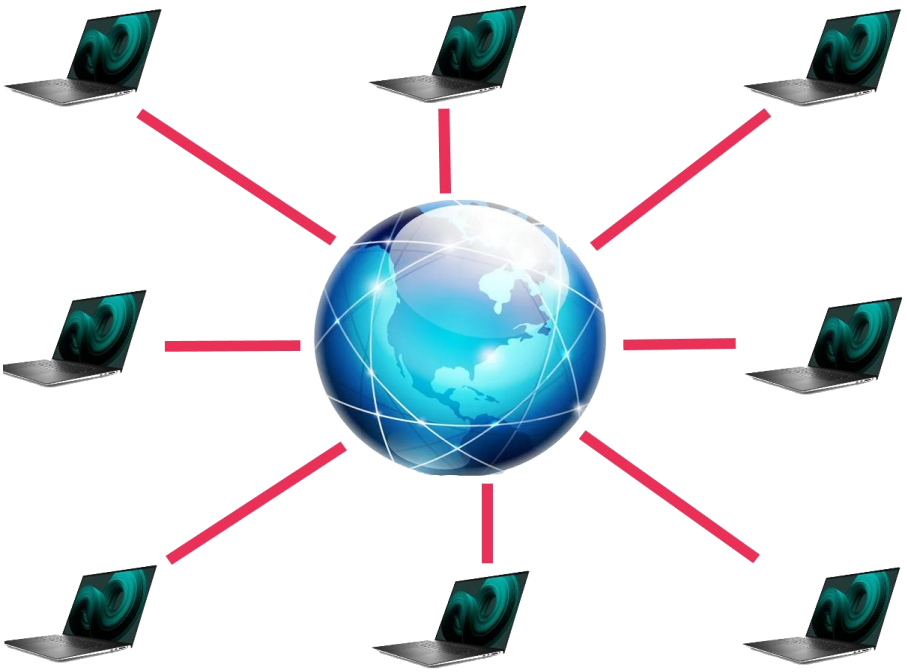
Keywords: Edge Scattering Number, Edge Integrity, Edge Rupture Degree, Vertex, Edge, Graph Theory.

2022, 62 pages

1. GİRİŞ

1.1 Günümüzde İletişim Ağı ve Önemi

İletişim, kişiler arasında, duygu, düşünce, bilgi, haber alışverişi, bilgi ve haberlerin, akla gelebilecek her türlü biçim ve yolla kişiden kişiye karşılıklı olarak aktarılmasıdır. İletişim, çoğunlukla uzak mesafeler arası veri iletimi ya da haberleşme olarak görülmektedir. Günümüzde ise telefon hatları ile sesin iletimi değil, Bilişim Teknolojileri ile bir yerden başka bir yere sayısal veri iletimi olarak anlaşılmaktadır. Bilgisayar ağlarının ilk uygulamaları 1960'ların sonlarında başlamıştır. Ancak özellikle kişisel bilgisayarların ortaya çıkması ve hızla gelişmesiyle 1980'li yıllarda yerel ağ uygulamaları önemli ölçüde gözlemlenmiştir. Bu dönemlerde Bilişim Teknolojileri de gelişmiştir. Böyle bir durum bilgisayar ağlarına büyük etki etmiştir. Günümüzde ise en basit bir bilgisayar dahi ağ bağlantılarını sağlayabilecek kapasitededir.



Şekil 1.1. İletişim ağı

1.2. Çizge Teorisi Tanımı ve Uygulama Alanları

Çizge teorisi, bir problemin ayrıt ve tepeler ile modellenip bu modelin çizge ile gösterilmesi ve bu çizgeleri inceleyen matematik dalıdır. Günümüzde karmaşık bir hal almış birçok problem modelleme yolu ile daha sade ve anlaşılır bir hale getirilebilir. Bu sayede çizge teorisi uygulanabilir. Çizge teorisi modellemesi yapılabilmesi için öncelikle çizgenin tanımı yapılmalıdır. Çizge, tepeler ve bu tepeleri birbirine bağlayan ayrıtlardan oluşan bir tür ağ yapısıdır. Bir nokta, bağlantı, ayrıt veya yay kümesinin basit kavramının genelleştirilmesidir.

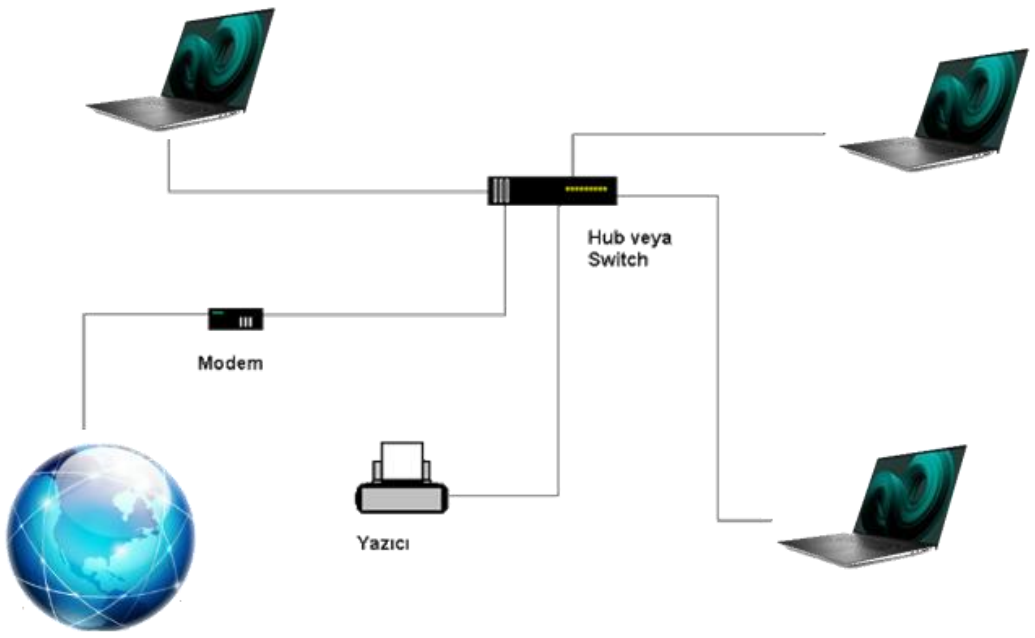
Çizge;

- G = (V, E) şeklinde gösterilir.
    - V, vertices yani tepeler,
    - E, edge yani ayrıtlar olarak ifade edilir.

Günlük yaşantımızdaki problemlere uygulanabilmektedir.

Örneğin:

- Karayolları ulaşım ile ilgili, şehirleri tepelerle, yolları ise ayrıtlarla,
- Bilgisayar ağlarında, bilgisayarları tepelerle, kablolar ise ayrıtlarla ifade edilebilmektedir.



Şekil 1.2. Bilgisayar iletişim ağı

Ağ yapılarının merkezleri tepeler, bağlantıları ise ayrıt olan çizgelerle modellenmiştir. Ağ tasarımı yapılırken, güvenlik açıkları bulunabilir ve çizge modellemesi yapıldığında sayısal bir hesaplama yapılabilir. Bu durumlar göz önünde bulundurularak güvenilir ve düşük maliyetli ağ yapıları için zedelenebilirlik parametreleri bulunmaktadır.

Bir ağ yapısında merkezler arası bağlantı hatları hasar gördüğünde iletişim sorunları ortaya çıkar. Çizge teorisi, bu iletişim sorunlarının çözülmesi için yapının tepe ve ayrıt ile çizge olarak gösterilip modellenmesidir. Gerçek dünyadan bir problem çizge olarak modellenebilir ve bu model çözülerek gerçek dünyada uygulanması sağlanabilir. Bu tarz modelleme ile başarıya ulaşmak çizge teorisinin başarılı olduğunu gösterir.

1.3. Zedelenebilirlik (Vulnerability)

İletişim ağı, merkezler ve bu merkezler arası bağlantı hatlarından oluşur. Bazı olumsuz durumlarda, ağ yapısının ne kadar ve nasıl dayanacağı önemli bir sorundur. İşte tam burada “zedelenebilirlik” kavramı ortaya çıkar.

İletişim halinde olan ağ yapısında tepelerin veya ayrıtların hasar görmesinden sonra iletişim kaybı olana kadar geçen sürede gösterdiği ağın dayanma gücünün ölçümüne zedelenebilirlik denir [10].

Zedelenebilirlik, bir sistemde o sistemin belirli bir konuda işlevini yitirmesine, azalmasına ya da kaybolmasına neden olabilecek hasar karşısında gösterdiği dayanıklılık ölçümüdür.

Bir ağ analizinde değerlendirilmesi gereken konu, ağın güvenliği ve kesintisiz olmasıdır.

Şekil 1.2'deki örnekten düşünecek olursak, bilgisayarlar tepeleri, kablolar ise ayrıtları oluşturur. Bir ağ yapısında bulunan bağlantılar arasında bozulma veya yok olma gibi durumlar oluştuğunda, ağın genelinde oluşan hasarın derecesi araştırılırken aşağıdaki sorular sorulabilir:

- Hasar gören tepe sayısı,
- Hasar gören ayrıt sayısı,
- En fazla zarar görmeden devam eden tepe sayısı,
- Hasar gören tepelerde, birbirinden bağlantısız tepe kümelerinin sayısı,
- Hasar gören tepeler ile bağlantılı olan tepe sayısı,
- Hasar gören ayrıtlar ile bağlantılı olan tepelerin diğer bağlantılarının sayısı,
- Hasar görenler haricinde iletişimi devam eden tepe sayısını da araştırmamız gerekmektedir.

2. GENEL TANIMLAR

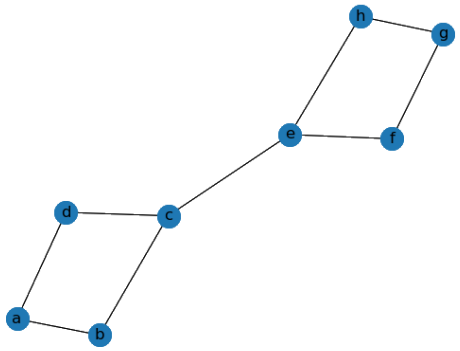
Tanım 2.1. [11] Bir çizgenin bir tepesine bağlı ayrıtlarının sayısına o tepenin derecesi denir. deg(v) ile gösterilir.

Tanım 2.2. [11] Bir çizgenin bütün tepe dereceleri içinde en küçük tepe derecesine minimum tepe derecesi denir. $\delta(G)$ ile gösterilir.

Tanım 2.3. [11] Bir çizgenin bütün tepe dereceleri içinde en büyük tepe derecesine maksimum tepe derecesi denir. $\Delta(G)$ ile gösterilir.

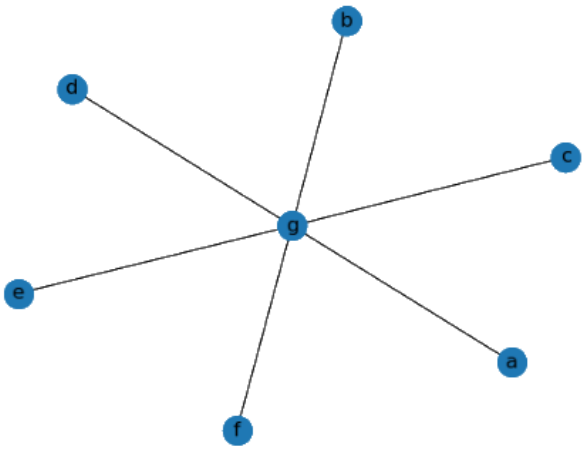
Tanım 2.4. [11] Bir çizgede, çıkarılan tepe veya ayrıtların kümesine kesim kümesi denir. $|S|$ ile gösterilir.

Tanım 2.5. [11] Bir çizgenin herhangi bir tepesinden başka herhangi bir tepesine yol var ise bu çizgeye bağlantılı (connected) çizge denir.



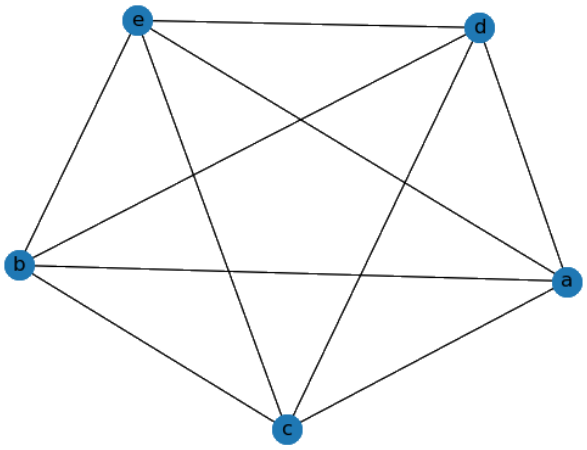
Şekil 2.1. Bağlantılı Çizge (Connected Graph)

Tanım 2.6. [11] n+1 tepeye sahip bir çizgenin n adet tepesinin derecesi bir olup merkezindeki tepesinin derecesi n olmak üzere n+1 tepeli çizgeye yıldız çizge denir. Şekil 2.1'de yıldız çizge örneği gösterilmiştir. $K_{1,n}$ ile gösterilir.



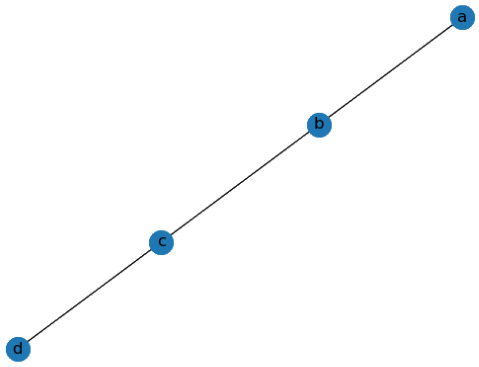
Şekil 2.2. Yıldız Çizge (Star Graph)

Tanım 2.7. [11] Çizgenin tüm tepeleri, diğer tepelerine komşu ise bu çizgeye tam çizge denir ve *n* tepeli bir tam çizge K_n ile gösterilir.
Şekil 2.3'de tam çizge örneği gösterilmiştir.



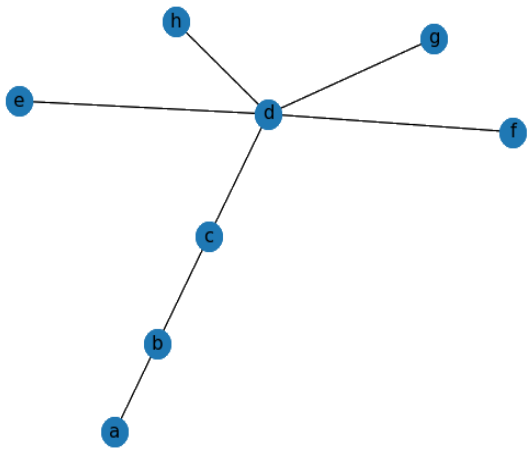
Şekil 2.3. Tam Çizge (Complete Graph)

Tanım 2.8. [13] Baştaki ve sondaki tepesi tek dereceli, geriye kalan tepeleri ise iki dereceli olan çizgeye yol çizge denir. Şekil 2.4'de yol çizgesi örneği gösterilmiştir.



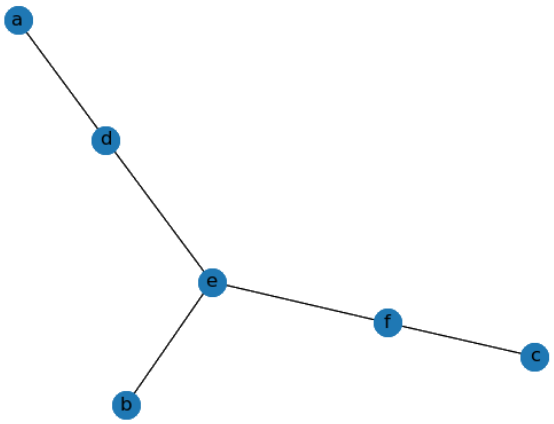
Şekil 2.4. Yol Çizge (Path Graph)

Tanım 2.9. [2] Yıldız çizgenin merkez tepesine, n-1 tepeli P_{n-1} yol çizgesinin eklenmesi ile oluşan çizgeye kuyruklu yıldız (Comet Graph) çizgesi denir.
Şekil 2.5'de kuyruklu yıldız çizgesi gösterilmiştir. $C_{n,m}$ ile gösterilir.



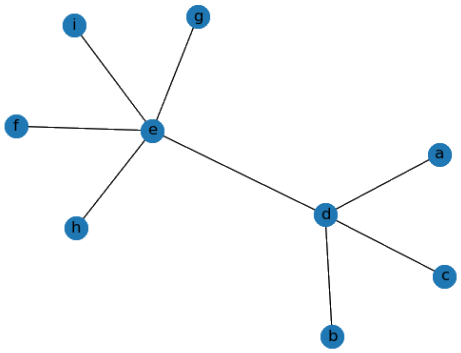
Şekil 2.5. Kuyruklu Yıldız Çizge (Comet Graph)

Tanım 2.10. [14] $V(G) = n$ olan bir çizge olsun. $p_1, p_2, \dots, p_n$ olan diken çizgesi G çizgesinin $i = 1, 2, \dots, n$ eklenmesi ile elde edilen çizgeye Dikenli Çizge (Thorn) denir. G'nin vi tepesine pi pendent tepeleri eklenerek elde edilen çizgedir. Bu tepeye bağlı p_i asılı tepeler v_i' nin dikenleri olarak adlandırılır.



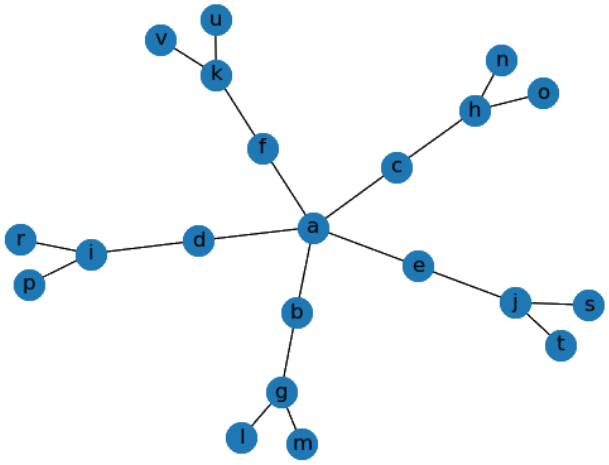
Şekil 2.6. Dikenli Çizge (Thorn Graph)

Tanım 2.11. Sırasıyla 2 farklı dereceye sahip merkezi tepeleri olan bir ağaçtır. Tam olarak iki sarkık olmayan tepe içeren çizgeye Çift Yıldız çizge denir. $K_{r,m}$ ile gösterilir.



Şekil 2.7. Çift Yıldız Çizge (Double Star Graph)

Tanım 2.12. Bir kopyanın her bir kopyasının bir yaprağının bağlanmasıyla elde edilen çizgedir. $B_{r,m}$ ile gösterilir.



Şekil 2.8. Muz Ağacı Çizgesi (Banana Tree Graph)

Tanım 2.13. [2] Bağımsızlık sayısı (Indepence number), G çizgesinin maksimum tepe sayısı olarak bilinir ve β(G) ile gösterilir.

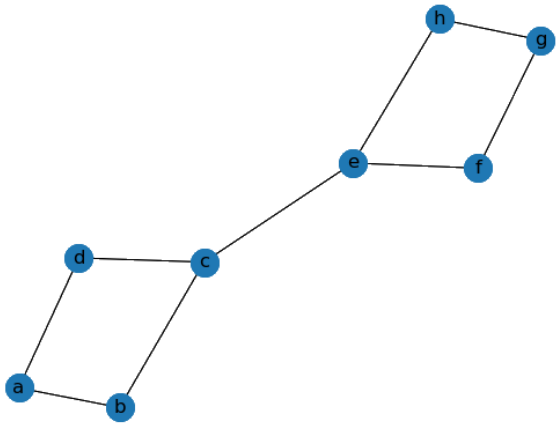
Tanım 2.14. [9] Bir G çizgesinin saçılma sayısı es(G) ile gösterilir. $\mathrm{es}(\mathrm{G}) = max\{\omega(G-S) - |S| : S \subseteq V(G), \omega(G-S) > 1\}$'dir. Burada $|S|$ çıkarılacak olan ayrıtlar kümesini, $\omega(G-S)$; $G-S$ ise ayrıtlar çıkarıldıktan sonra G çizgesinde geriye kalan bileşenlerinin sayısını gösterir.

Tanım 2.15. [4] Bir G çizgesinin ayrıt kopma derecesi (rupture) $\mathrm{r}'(\mathrm{G})$ ile gösterilir ve, $\mathrm{r}'(\mathrm{G}) = \max\{\omega(\mathrm{G}-\mathrm{S}) - |\mathrm{S}| - \mathrm{m}(\mathrm{G}-\mathrm{S}) : \mathrm{S} \subseteq \mathrm{E}(\mathrm{G}), \omega(\mathrm{G}-\mathrm{S}) > 1\}$ dir.
Burada $G-S$ içinde $\omega(G-S)$ bileşen sayısını ve $\mathrm{m}(\mathrm{G}-\mathrm{S})$ en büyük bileşenin tepe sayısını gösterir.

Tanım 2.16. [7] Bir G çizgesinin ayrıt bütünlüğü (integrity) $\mathrm{I}'(\mathrm{G})$ ile gösterilir ve $\mathrm{I}'(\mathrm{G}) = \min\{|\mathrm{S}| + \mathrm{m}(\mathrm{G}-\mathrm{S}) : \mathrm{S} \subseteq \mathrm{E}(\mathrm{G})\}$'dir. Burada G-S içinde $\mathrm{m}(\mathrm{G}-\mathrm{S})$ en büyük bileşenin tepe sayısını gösterir.

3. AYRIT BAĞLANTISI (EDGE CONNECTIVITY)

Bu bölümde, ayrıt bağlantısının tanımı ve bazı örnekler verilmiştir. Connectivity, çizge teorisinin temel bir kavramıdır. Bir çizgenin bağlı olup olmadığını tanımlar. Bağlanabilirlik olmadan, bir çizgede bir tepe noktasından başka bir tepe noktasına geçmek mümkün değildir.
Bağlı bir çizge olan G'nin ayrıt bağlantısı $\lambda(\mathrm{G})$ çıkarılması ise G'nin bağlantısını kestiği en küçük ayrıt sayısıdır. $\lambda(\mathrm{G}) \geq \mathrm{k}$ olduğunda, G çizgesinin k ayrıtına bağlı olduğu söylenir.
Bir çizgenin G'den silinmesi G'nin bağlantısını kestiği minimum ayrıt $\lambda(G)$ sayısı, hat bağlantısı olarak da adlandırılır.
Bağlantısı kesilmiş bir çizgenin ayrıt bağlantısı 0, çizge köprüsü olan bağlı bir çizge ise 1'dir.

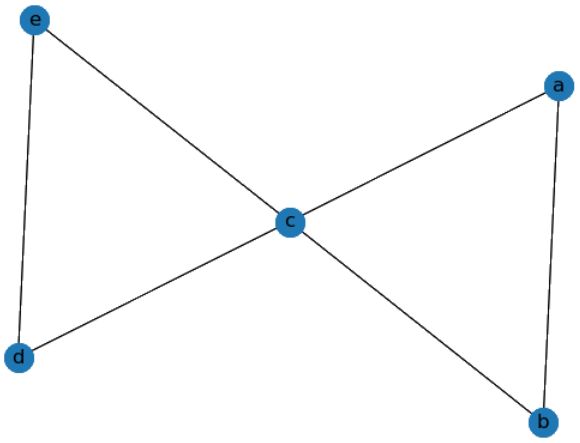


Şekil 3.1. Tek ayrıt bağlı bağlantılı çizge

Yukarıdaki örnekte bir tepe noktasından başka bir tepe noktasına ayrıtlar ile iletişim mümkündür. A tepe noktasından G tepe noktasına geçilebilir.

C tepe noktası ve E tepe noktası arasındaki ayrıt kesilmesi durumunda bağlantısı kesilmiş bir çizge olacaktır.

Örnekte belirtilen G_1 çizgesinde CE köprü görevi görmektedir. Tek bir ayrıtın kaldırılması durumunda bağlantı kesileceğinden dolayı edge connectivity (ayrıt bağlılık) sayısı, $\lambda(G) = 1$'dir.



Şekil 3.2. Çift ayrıt bağlı bağlantılı çizge

Yukarıda belirtilen çizge örneğinde tek bir ayrıtı kaldırmak bağlantının kopması için yeterli değildir. AC ve BC gibi 2 ayrıtı kaldırmak bağlantının kopması için yeterli olacaktır.

Bu 2 ayrıtı kaldırdıktan sonra 2 ayrı bağlantısız çizge oluşacaktır.

O halde $\lambda(G) = 2$'dir.

Bir iletişim ağında, güvenlik açığı, belirli istasyonların veya iletişim bağlantılarının arızalanmasından sonra ağın iletişiminin kesintiye uğramasına karşı direncini ölçer. Güvenlik açığını ölçmek için bağlantı, bütünlük, saçılma sayısı ve kopma derecesi olan bazı parametrelerimiz var.

4. AYRIT SAÇILMA SAYISI (EDGE SCATTERING NUMBER)

Ayrıt saçılma sayısı, çizgenin tepelerini, ayrıtlarını ve komşuluklarını anlamak için önemli bir ölçümdür. Çizgenin yapısını iyi çözümleyebilmemizi ve en iyi sonuca ulaşmamızı sağlar.

Ayrıt saçılma sayısı**,** ağa zarar vermek için yapılan iş miktarı ile ağın ne kadar kötü hasar gördüğü arasındaki dengeyi temsil eder. Bağlantısı kesilen ağ daha fazla bileşen içeriyorsa bozulmanın daha başarılı olduğunu söyleyebiliriz.

Bu bölümde, ayrıt saçılma sayısının tanımı ve bazı teoremler verilmiştir. Sonrasında özel çizgelerden biri olan yıldız çizgenin ayrıt saçılma sayısı hesaplanmıştır.

4.1. Tanım ve Teoremler

Tanım 4.1.1 [9] Tamamlanmamış bir bağlı çizge G'nin ayrıt saçılma sayısı,

$$\mathrm{es(G)} = \max\{\omega(\mathrm{G} - \mathrm{S}) - |\mathrm{S}| : \mathrm{S} \subseteq \mathrm{E(G)}, \omega(\mathrm{G} - \mathrm{S}) > 1\}$$

burada $\omega(\mathrm{G} - \mathrm{S})$, G – S'deki bileşen sayısını gösterir.

Eğer $\mathrm{es(G)} = \omega(\mathrm{G} - \mathrm{S}) - |\mathrm{S}|$ ise bir set $\mathrm{S} \subseteq \mathrm{E(G)}$, G'nin es-setini belirtir.

İyi bilinen çizge parametreleri açısından ayrıt saçılma sayısı için bazı alt ve üst sınırlar veririz.

Teorem 4.1.2. [9] G bağlı bir çizge olsun,

$$es(G) \leq 1$$

Teorem 4.1.3. [9] G bağlı çizgesinde n ve m sırasıyla tepe ve ayırt sayısını gösterir. O halde;

$$es(G) \geq n - m$$

Teorem 4.1.4. [9] G bir λ-ayrıt bağlı çizge olsun,

$$es(G) \geq 2 - \lambda$$

Teorem 4.1.5. [9] G bir bağlı çizge olsun ve δ(G) minimum derece olsun. O halde,

$$es(G) \geq 2 - \delta(G)$$

4.2.Ayrıt Saçılma Sayısı Kod Yapısı

Ayrıt Saçılma Sayısı kod yapısında ilk olarak kullanılacak olan kütüphaneler dahil edilmiştir. Bunlar: Combinations, array, networkx, copy ve pyplot kütüphaneleridir.

Combinations: Tüm iterasyonları bulmayı sağlar.

Array: Dizi kullanımı için gereklidir.

Networkx: Karmaşık ağ yapısı ile çalışmak için tasarlanmış python kütüphanesidir.

Matplotlib.Pyplot: Veri görselleştirmesinde kullanılan python kütüphanesidir.

```
from itertools import combinations
import array as arr
import networkx as nx
import matplotlib.pyplot as plt
import copy
```

Çizge üzerinde işlem yapmak ve tanımlı çizge üzerindeki bilgileri elde edebilmek için oluşturulan "Graph" sınıfı ve bu sınıfın fonksiyonları aşağıdaki gibidir.

```
class Graph(object):
    def __init__(self, cizgesoz=None):
        if cizgesoz == None:
            cizgesoz = {}
        self.__cizgesoz = cizgesoz
    def Tepeler(self):
        return list(self.__cizgesoz.keys())
    def Ayritlar(self):
```

```
        return self.AyritlariOlustur()
    def TepeEkle(self, tepe):
        if tepe not in self.__cizgesoz:
            self.__cizgesoz[tepe] = []
    def AyritEkle(self, ayrit):
        ayrit = set(ayrit)
        (tepe1, tepe2) = tuple(ayrit)
        if tepe1 in self.__cizgesoz:
            self.__cizgesoz[tepe1].append(tepe2)
        else:
            self.__cizgesoz[tepe1] = [tepe2]
    def AyritlariOlustur(self):
        ayritdizi = []
        for tepe in self.__cizgesoz:
            for komsu in self.__cizgesoz[tepe]:
                if {komsu, tepe} not in ayritdizi:
                    ayritdizi.append({tepe, komsu})
        return ayritdizi
```

"__init__ (self, cizgesoz = None)" fonksiyonu, sınıf içerisinden türetmiş olduğumuz nesnelere ulaşmamızı sağlar.

"Tepeler (self)" fonksiyonu, fonksiyona gönderilen çizgenin tepe noktalarını elde etmemizi sağlar.

"Ayritlar (self)" fonksiyonu, fonksiyona gönderilen çizgenin ayrıtlarını elde etmemizi sağlar.

"TepeEkle (self, tepe)" fonksiyonu, fonksiyona gönderilen çizgenin tepe noktası eklememizi sağlar.

"AyritEkle (self, ayrit)" fonksiyonu, fonksiyona gönderilen çizgenin ayrıt eklememizi sağlar.

"AyritlariOlustur (self)" fonksiyonu, fonksiyona gönderilen çizgenin tepe-ayrıt komşuluk yapılarını tanımlamamızı sağlar.

$\omega\,(G-S),$ yani çizgedeki bileşen sayısını elde etmemizi sağlayan "KokAl" fonksiyonu aşağıdaki gibidir.

```
def KokAl(aKoms):
    def KokBul(aTepe,aKok):
        while aTepe != aKok[aTepe][0]:
            aTepe = aKok[aTepe][0]
        return (aTepe,aKok[aTepe][1])
    kokdizi = {}
    for TepeNok in aKoms.keys():
        kokdizi[TepeNok] = (TepeNok,0)
```

```
    for i in aKoms:
        for j in aKoms[i]:
            (kokdizi_i,derinliki) = KokBul(i,kokdizi)
            (kokdizi_j,derinlikj) = KokBul(j,kokdizi)
            if kokdizi_i != kokdizi_j:
                Mindeger = kokdizi_i
                Maksdeger = kokdizi_j
                if  derinliki > derinlikj:
                    Mindeger = kokdizi_j
                    Maksdeger = kokdizi_i
                kokdizi[Maksdeger] =
(Maksdeger,max(kokdizi[Mindeger][1]+1,kokdizi[Maksdeger][1]))
                kokdizi[Mindeger] = (kokdizi[Maksdeger][0],-1)
    yenidizi = {}
    for i in aKoms:
        if kokdizi[i][0] == i:
            yenidizi[i] = []
    for i in aKoms:
        yenidizi[KokBul(i,kokdizi)[0]].append(i)
    return yenidizi
```

Dizi içerisine tanımlanmış olan çizge, "KokAl" fonksiyonuna gönderilerek $\omega\ (G - S)$ sayısı elde edilir.

"KokAl" fonksiyonu içerisine, içerisindeki işlemlerde kullanılacak olan ve gönderilen verinin kökünü bulan "KokBul" fonksiyonu tanımlanmıştır.

Çizge "KokAl" fonksiyonuna gönderilir ve burada her döngü yardımı ile çizgenin her bir ayrıtı ve tepe komşulukları gezilerek $\omega\ (G - S)$ sayısı elde edilir.

Çizge sınıfından çizge oluşturmayı sağlayan "createGraphClass" fonksiyonu aşağıdaki gibidir.

```
def createGraphClass(othergraph):
    graph = Graph(othergraph)
    return graph
```

Çizge yapısında izole tepeleri bulmamızı sağlayan "find_isolated_nodes" fonksiyonu aşağıdaki gibidir.

```
def find_isolated_nodes(graph):
    isolated = []
    for node in graph:
        if not graph[node]:
            isolated += node
    return isolated
```

Çizge yapısında çıkarılacak ayrıt ya da ayrıtlar sonucunda elde edilen verileri gösteren "edgesnumber" fonksiyonu olarak tanımlanmıştır.

```
def edgesnumber(graph, edge,edges):
    list   = []
    list2 = []
    graf = graph
    s = len(edge)
    sonuc = 0
    for n in edge:
        list= []
        for ns in n:
            list.append(ns)
        list2.append(list)
    for l in list2:
        dzlist = []
        say =0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say+1]) != "" and str(dzlist[say+1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say+1]))
        if str(dzlist[say]) != ""  and str(dzlist[say]) in graf[str(dzlist[say+1])]:
            graf[str(dzlist[say+1])].remove(str(dzlist[say]))
    yenigraph = graf
    w = len(KokAl(yenigraph))
    sonuc = w - s
    wgs = "w(G-S)= " + str(w)
    ss = "|S|= " + str(s)
    esg = "es(G) = w(G-S) - |S| = " + str(sonuc)
    goster = wgs + " " + ss + " " + esg
    veri = str(sonuc) + "*" + goster +"*" + str(yenigraph)
    if w > 1:
        return veri
    else:
        return 'none'
```

Çizge, çıkarılacak olan ayrıt ve tüm ayrıtlar fonksiyona gönderilir. Liste oluşturularak çıkarılacak ayrıt sayısı kadar döngü oluşturularak ayrıtlar listeye eklenir.

Liste döngü içerisinde kontrol edilerek boş ve çizge dizisi içerisinde olup olmadığı kontrol edilerek “remove()” komutu ile diziden çıkarılır.

Ayrıtları çıkarılmış olan yeni çizge dizisi “KokAl()” fonksiyonuna gönderilerek “len()” komutu ile dönen sonuç dizisinin uzunluğu elde edilir.

“len(edge)” komutu ile çıkarılacak olan ayrıt sayısı “s” değişkenine atandıktan sonra “w-s“ ile aradaki fark bulunur ve sözel olarak ekrana yazdırılır.

Sonucu sayı olarak geri döndüren “sonucal” fonksiyonu aşağıdaki gibidir.

```
def sonucal(graph, edge,edges):
    list  = []
    list2 = []
    graf  = graph
    s     = len(edge)
    sonuc = 0
    for n in edge:
        list= []
        for ns in n:
            list.append(ns)
        list2.append(list)
    for l in list2:
        dzlist = []
        say =0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say+1]) != "" and str(dzlist[say+1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say+1]))
        if str(dzlist[say]) != ""  and str(dzlist[say]) in graf[str(dzlist[say+1])]:
            graf[str(dzlist[say+1])].remove(str(dzlist[say]))
    yenigraph = graf
    w = len(KokAl(yenigraph))
    sonuc = w - s
    return str(sonuc)
```

Bu fonksiyon “edgesnumber” fonksiyonunun sayısal olarak kullanmak amaçlı veri döndüren halidir.

Çizge üzerinde çıkarılmış ayrıtları bulan “removedEdge” fonksiyonu aşağıdaki gibidir.

```
def removedEdge(cgraf, edge2,edges):
    list1 = []
    list3 = []
    graf2 = cgraf
    for nk in edge2:
        list1 = []
        for nsy in nk:
            list1.append(nsy)
        list3.append(list1)
    for lx in list3:
        dzlist2 = []
        say2 = 0
        for tx in lx:
            dzlist2.append(tx)
        if str(dzlist2[say2 + 1]) != "" and str(dzlist2[say2 + 1]) in
graf2[str(dzlist2[say2])]:
            graf2[str(dzlist2[say2])].remove(str(dzlist2[say2 + 1]))
        if str(dzlist2[say2]) != "" and str(dzlist2[say2]) in graf2[str(dzlist2[say2 + 1])]:
            graf2[str(dzlist2[say2 + 1])].remove(str(dzlist2[say2]))
```

```
    yenigraph2 = graf2
    return yenigraph2
```

Ayrıt Saçılma Sayısı yönteminde ana fonksiyonumuz olan “EdgeScatteringNumber” fonksiyonu aşağıdaki gibidir.

```
def EdgeScatteringNumber(othergraph):
    crg      = copy.deepcopy(othergraph)
    graph    = createGraphClass(crg)
    edges    = graph.edges()
    vertices = graph.vertices()
    cg       = copy.deepcopy(othergraph)
    graph    = createGraph(cg)
    G        = nx.Graph()
    for vert in vertices:
        G.add_node(vert)
    for gr in graph:
        for g in graph[gr]:
            G.add_edge(str(gr),g)
    nx.draw_networkx(G)
    plt.show()

    for i in range(1, len(edges)+1):
        iBoyluKomb = [list(x) for x in combinations(edges, i)]
        komb.extend(iBoyluKomb)

    xyz = []
    for x in komb:
         xyz.append(x)

    for x in xyz:
        copy_list = copy.deepcopy(othergraph)
        graf = createGraph(copy_list)
        i = 0
        sonliste.append(x)
              for say in x:
            sayac = sayac + str(say)
            i = i + 1
            if(i != len(x)):
                sayac = sayac + ","
        snc = edgesnumber(graf,x,edges)
        if snc != 'none':
            sonuclar.append(sonucal(graf, x, edges))
            dizi      = snc.split("*")
            esdeger   = dizi[0]
            maxbul.append(dizi[0])
            maxlist.append([sayac])
            dizis = [maxbul,sayac]
            print("Çizgeden S|"+sayac+"| çıkarılınca sonuç= "+dizi[1])
        sayac = ""
```

```
    nums          = [int(s) for s in maxbul]
    maksdeger = max(nums)
    maksindex = nums.index(max(nums))
    formul = ""
    i = 1
    for sn in sonuclar:
        if i!= len(sonuclar):
            formul = formul + sn + ","
        else:
            formul = formul + sn
        i = i+1
    print("\nes(G) = max{"+formul+"} = "+str(maksdeger))
    print("\nes(G) maximum değeri = "+ str(maksdeger)+", çıkarılan edge listesi S|"+
str(maxlist[maksindex])+"|")

    cgc          = copy.deepcopy(othergraph)
    cgraph2    = createGraphClass(cgc)
    edges2      = cgraph2.edges()
    vertices2   = cgraph2.vertices()
    cgg          = copy.deepcopy(othergraph)
    cgraph2    = createGraph(cgg)

    G2 = nx.Graph()
    for vt in vertices2:
        G2.add_node(vt)
    sonucgrafi = removedEdge(cgraph2,sonliste[maksindex],edges)
    for gr2 in sonucgrafi:
        for g2 in sonucgrafi[gr2]:
            G2.add_edge(str(gr2),g2)
    nx.draw_networkx(G2)
    plt.show()
```

Bu fonksiyon çizgenin tüm iterasyonlarını alıp, yönteme uygun bir biçimde tek tek hesaplayarak ekrana yazdıran, görsel olarakta ilk ve son hallerini ekranda gösteren bir fonksiyondur.

Tüm iterasyonları dolaştığımız döngünün içerisinde bulunan “sonucal” ve “edgesnumber” fonksiyonlarının döngülerinin zaman karmaşıklık değeri $O(n^2)$’dir. En dışta bulunan döngünün ise maliyeti $O(n^3)$’tür. Böylece algoritmanın zaman karmaşıklık değeri $O(n^3)$’tür.

Fonksiyon içerisinde çizge oluşturan ve oluşan çizgenin ayrıt ve tepelerini elde ettiğimiz kod aşağıdaki gibidir. “deepcopy” kodu çizge dizisini kopyalamayı sağlar.

Gönderilen çizge ilk olarak “createGraphClass()” ve “createGraph()” fonksiyonları çağırılarak değişkene atılır. Networkx kütüphanesi yardımı ile oluşturulan “G” değişkeninin içerisine döngü ile eklenir. Çizgenin ilk halini ekranda göstermek için nx.draw_networkx(G) ve plt.show() komutları ile çizgenin ilk hali ekrana getirilir.

Çizgede, “combinations” komut yardımı ile çizgenin tüm iterasyonlarını bulup “komb” dizisi içerisine aktarılır.

Döngü 1’den başlatılıp ayrıt sayısı kadar döndürülerek tüm ayrıtların iterasyonları bulunur.

Burada “xyz” dizisi tanımlanarak iterasyonlar bu dizi içerisine “append” yardımı ile ekler.

Tüm iterasyonların bulunduğu “xyz” dizisi döngü yardımı ile tek tek dolaşılmaktadır.

Maksimum değer ve maksimum değerin indeksi alınarak değişkenlere aktarılır.

Bulunan sonucun formül ve değerleri “print()” komutu ile ekrana yazdırılır.

“othergraph” çizgesi “deepcopy()” ile farklı bir değişkene atılıp, “createGraphClass()” ve “createGraph” fonksiyonları ile yeni çizge sınıfı ve çizge oluşturulmuştur.

“nx.Graph()” ile G2 çizgesi tanımlanıp sonrasında döngü içerisinde “add_node()” ile tepeleri eklenir.

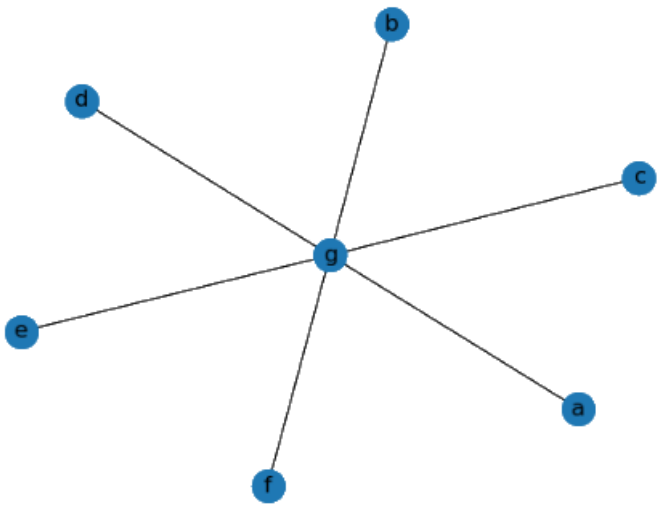
“removedEdge” fonksiyonu ile çıkarılan ayrıtlar değişkene atılıp, G2 çizgesi içerisine eklenir.

Eklenen ayrıtlar son olarak gösterilmek üzere “nx.draw_networkx()” ve “plt.show()” komutları ile görsel olarak ekrana yazdırılmaktadır.

4.3. Ayrıt Saçılma Sayısı Algoritma Uygulaması

Ayrıt saçılma sayısı algoritması özel çizgelerde uygulanmıştır.

4.3.1. Yıldız Çizge Ayrıt Saçılma Sayısı



Komşuluklar

a tepesi için {g},

b tepesi için {g},

c tepesi için {g},

d tepesi için {g},

e tepesi için {g},

f tepesi için {g},

g tepesi için {a,b,c,d,e,f} tepeleri komşu tepelerdir.

Ayrıt Saçılma Sayısı tanımından yola çıkarsak,

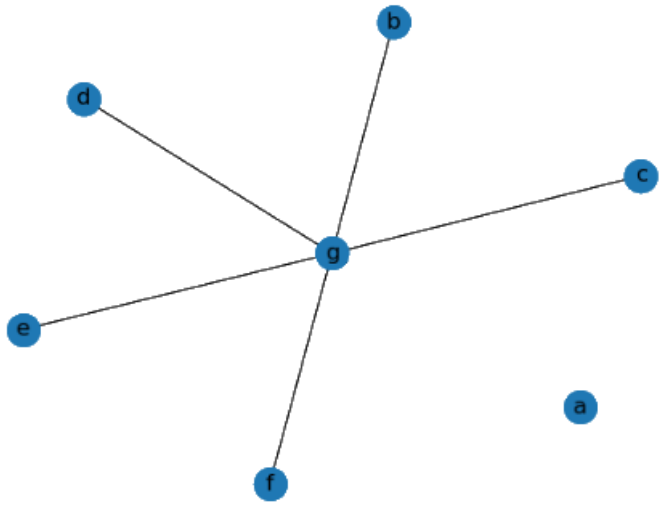
$$es(G) = \max\{\omega(G-S) - |S| : S \subseteq E(G), \omega(G-S) > 1\}$$

$\omega(G-S)$, $G-S$'deki bileşen sayısını,

$|S|$ ise çıkarılan tepeleri gösterir.

Tüm iterasyonlar denenerek en uygun sonuç bulunur. Bu iterasyonlardan bazıları aşağıda Yıldız Çizge için uygulanmıştır.

$|S_1| = \{\ [a, g]\ \}$ ayrıtı çıkarılması durumu için,



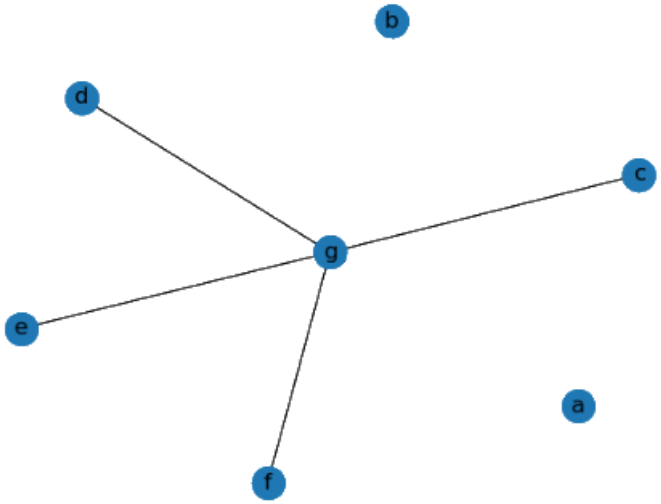
a ile g tepeleri arasındaki ayrıt çıkarıldığında a tepesi ile g tepesi arasındaki bağlantı kesilmiş olur. Kesilen bağlantı sonucunda bileşen sayısı ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısından, çizgeden çıkarılan ayrıt sayısını çıkararak ayrıt saçılma sayısını elde ederiz. O halde,

$\omega(G-S) = 2,\ |S| = 1, es(G) = \omega(G-S) - |S| = 1$ dir.

Bileşen sayısı 2, çıkarılan ayrıt sayısı 1 ve bulunan sonuç 2 - 1 = 1 dir.

Şimdi farklı iki ayrıt çıkarıp deneyelim.

$|S_2| = \{ [a, g], [b, g] \}$ ayrıtı çıkarılması durumu için,



a ile g arasındaki ve b ile g arasındaki ayrıtlar çıkarıldığında a tepesi ile g tepesi arasındaki ve b tepesi ile g tepesi arasındaki bağlantı kesilmiş olur. Kesilen bağlantı sonucunda bileşen sayısı ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısından, çizgeden çıkarılan ayrıt sayısını çıkararak ayrıt saçılma sayısını elde ederiz. O halde,

$\omega(G - S) = 3,\ |S| = 2,\ es(G) = \omega(G - S) - |S| = 1$'dir.

Bileşen sayısı 3, çıkarılan ayrıt sayısı 2 ve bulunan sonuç 3 - 2 = 1 dir.

Bu şekilde tek tek çıkarılabilecek tüm iterasyonları deneyip en verimli sonuca ulaşılmaktadır.

Tüm iterasyonların denenmesi ile çıkan sonuç ise,

$$es(G) = \max\{es_1, es_2, \dots\} = \max\{1,1,1,1,1,1, \dots\} = 1'\text{dir.}$$

5. AYRIT KOPMA DERECESİ (EDGE RUPTURE DEGREE)

Bu bölümde, ayrıt kopma derecesinin tanımı, hesaplanan bazı teoremler verilmiştir. Daha sonra bazı özel çizgelerin ayrıt kopma derecesi hesaplanmıştır.

5.1 Tanım ve Teoremler

Tanım 5.1.1 [4] Bağlantılı bir G çizgesinde $S \subseteq E(G)$ olmak üzere,

$$r'(G) = \max\{\omega(G - S) - |S| - m(G - S) : S \subseteq E(G), \omega(G - S) > 1\} \text{ dir.}$$

$\omega(G - S)$, bileşen sayısını ve $m(G - S)$ ise en büyük bileşenin tepe sayısını gösterir.

Teorem 5.1.2. [4] C_n döngüsünün ayrıt kopma derecesi −1'dir.

Teorem 5.1.3. [4] Tüm bipartit çizgenin $K_{a,b}(3 < a \leq b)$,

Ayrıt kırılma derecesi ise, $3 - 2a - b$'dir.

Teorem 5.1.4. [4] $W_{1,n}$ tekerlek çizgesinin ayrıt kopma derecesi −n'dir.

5.2 Ayrıt Kopma Derecesi Kod Yapısı

Ayrıt kopma derecesinde ilk olarak kullanılacak olan kütüphaneler "import" komutu ile tanımlanmıştır.
Bu kütüphaneler kombinasyon, dizi, bağlantı oluşturma, görsel olarak ekranda gösterme ve kopyalama kütüphaneleridir.

```
from itertools import combinations
import array as arr
import networkx as nx
import matplotlib.pyplot as plt
import copy
```

Çizge üzerinde veri elde etmek için "Graph" sınıfı oluşturulmuştur. Bu sınıfta tepeleri ve ayrıtları liste olarak geri döndürme, tepe ekleme, ayrıt ekleme, ayrıt komşuluklarını tanımlama gibi bazı tanımlı fonksiyonlar mevcuttur.

```
class Graph(object):
    def __init__(self, cizgesoz=None):
        if cizgesoz == None:
            cizgesoz = {}
        self.__cizgesoz = cizgesoz
    def Tepeler(self):
        return list(self.__cizgesoz.keys())
    def Ayritlar(self):
        return self.AyritlariOlustur()
    def TepeEkle(self, tepe):
        if tepe not in self.__cizgesoz:
            self.__cizgesoz[tepe] = []
    def AyritEkle(self, ayrit):
        ayrit = set(ayrit)
        (tepe1, tepe2) = tuple(ayrit)
        if tepe1 in self.__cizgesoz:
            self.__cizgesoz[tepe1].append(tepe2)
        else:
            self.__cizgesoz[tepe1] = [tepe2]
    def AyritlariOlustur(self):
        ayritdizi = []
        for tepe in self.__cizgesoz:
            for komsu in self.__cizgesoz[tepe]:
                if {komsu, tepe} not in ayritdizi:
                    ayritdizi.append({tepe, komsu})
```

```
    return ayritdizi
```

Bileşen sayısı $\omega(G - S)$ bulmak için tanımlı "KokAl" fonksiyonu aşağıdaki gibidir.

```
def KokAl(aKoms):
   def KokBul(aTepe,aKok):
      while aTepe != aKok[aTepe][0]:
         aTepe = aKok[aTepe][0]
      return (aTepe,aKok[aTepe][1])
   kokdizi = {}
   for TepeNok in aKoms.keys():
      kokdizi[TepeNok] = (TepeNok,0)
   for i in aKoms:
      for j in aKoms[i]:
         (kokdizi_i,derinliki) = KokBul(i,kokdizi)
         (kokdizi_j,derinlikj) = KokBul(j,kokdizi)
         if kokdizi_i != kokdizi_j:
            Mindeger = kokdizi_i
            Maksdeger = kokdizi_j
            if  derinliki > derinlikj:
               Mindeger = kokdizi_j
               Maksdeger = kokdizi_i
            kokdizi[Maksdeger] =
(Maksdeger,max(kokdizi[Mindeger][1]+1,kokdizi[Maksdeger][1]))
            kokdizi[Mindeger] = (kokdizi[Maksdeger][0],-1)
   yenidizi = {}
   for i in aKoms:
      if kokdizi[i][0] == i:
         yenidizi[i] = []
   for i in aKoms:
      yenidizi[KokBul(i,kokdizi)[0]].append(i)
   return yenidizi
```

Bu fonksiyonda tüm çizge döngü yardımı ile dolaşılarak komşulukları kontrol edilir ve bileşen sayısı bulunur. Döngüler içerisinde kullanılmak üzere "KokBul" fonksiyonu tanımlanmıştır.

Döngü içerisinde "KokBul" fonksiyonu yardımı ile bulunan sonuçlar "yenidizi" dizisi içerisine eklenir ve dizi geri döndürülür. Böylece $\omega(G - S)$ bulunmuş olur.

```
def cikarilmisedge(graph, edge, edges):
   list  = []
   list2 = []
   graf = graph
   for n in edge:
      list= []
      for ns in n:
         list.append(ns)
      list2.append(list)
   for l in list2:
```

```
        dzlist = []
        say =0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say+1]) != "" and str(dzlist[say+1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say+1]))
        if str(dzlist[say]) != ""  and str(dzlist[say]) in graf[str(dzlist[say+1])]:
            graf[str(dzlist[say+1])].remove(str(dzlist[say]))
    yenigraph = graf
    return yenigraph
```

“cikarilmisedge” fonksiyonu, çıkarılacak olan ayrıtı çıkarıp, çıkarılmış halini elde etmemizi sağlamaktadır.

Tüm tepeler dolaşılarak çıkarılacak olan ayrıt bulunur ve “remove” yardımı ile çıkarılır. Son hali elde edilir.

```
def edgesnumber(graph, edge,edges):
    list   = []
    list2 = []
    graf = graph
    s = len(edge)
    sonuc = 0
    for n in edge:
        list = []
        for ns in n:
            list.append(ns)
        list2.append(list)
    for l in list2:
        dzlist = []
        say = 0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say+1]) != "" and str(dzlist[say+1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say+1]))
        if str(dzlist[say]) != ""  and str(dzlist[say]) in graf[str(dzlist[say+1])]:
            graf[str(dzlist[say+1])].remove(str(dzlist[say]))
    yenigraph = graf
    w = len(KokAl(yenigraph))
    m =  KokAl(yenigraph)
    sq = 0
    asd = []
    for i in m:
        sycc = 0
        for j in m[i]:
            sycc = sycc + 1
        asd.append(sycc)
        sq = sq + 1
    maxdeger = 0
    for f in asd:
```

```
        if f > maxdeger:
            maxdeger = f
    sonuc = w - s - maxdeger
    mgs = "m(G-S)= " + str(maxdeger)
    wgs = "w(G-S)= " + str(w)
    ss = "|S|= " + str(s)
    esg = "r'(G) = w(G-S)-|S|-m(G-S) = " + str(sonuc)
    goster = mgs + " " +wgs + " " + ss + " " + esg
    veri = str(sonuc) + "*" + goster + "*" + str(yenigraph)
    if w > 1:
        return veri
    else:
        return 'none'
```

“edgesnumber” fonksiyonu çizge üzerinde çıkarılması gereken ayrıtı çıkarıp, döngüler yardımı ile sonucu bulmamızı sağlamaktadır. Çıkarılan ayrıt sonrasında çizge içerisinde döngü ile tüm tepeler dolaşılarak $m(G - S)$ bulunmaktadır.

Bu fonksiyon ana fonksiyonumuz olan “EdgeRuptureDegree” fonksiyonunda kullanılmak üzere tanımlanmıştır. Gerekli parametreler “*” ile birleştirilerek geri döndürülmüştür.

```
def sonucal(graph, edge, edges):
    list = []
    list2 = []
    graf = graph
    s = len(edge)
    sonuc = 0
    for n in edge:
        list = []
        for ns in n:
            list.append(ns)
        list2.append(list)
    for l in list2:
        dzlist = []
        say = 0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say + 1]) != "" and str(dzlist[say + 1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say + 1]))
        if str(dzlist[say]) != "" and str(dzlist[say]) in graf[str(dzlist[say + 1])]:
            graf[str(dzlist[say + 1])].remove(str(dzlist[say]))
    yenigraph = graf
    w = len(KokAl(yenigraph))
    m = KokAl(yenigraph)
    sq = 0
    asd = []
    for i in m:
        sycc = 0
        for j in m[i]:
```

```
            sycc = sycc + 1
        asd.append(sycc)
        sq = sq + 1
    maxdeger = 0
    for f in asd:
        if f > maxdeger:
            maxdeger = f
    sonuc = w - s - maxdeger
    return str(sonuc)
```

“sonucal” fonksiyonu sırası ile çizgeden ayrıtları çıkartıp Ayrıt Kopma Derecesinin sonucunu döndürmektedir.

```
def EdgeRuptureDegree(othergraph):
    crg = copy.deepcopy(othergraph)
    graph = createGraphClass(crg)
    edges = graph.edges()
    vertices = graph.vertices()
    cg = copy.deepcopy(othergraph)
    graph = createGraph(cg)

    G = nx.Graph()
    for vert in vertices:
        G.add_node(vert)
    for gr in graph:
        for g in graph[gr]:
            G.add_edge(str(gr),g)
    nx.draw_networkx(G)
    plt.show()

    komb = []
    abc  = []
    z = 0
    for i in range(1, len(edges)+1):
        iBoyluKomb = [list(x) for x in combinations(edges, i)]
        komb.extend(iBoyluKomb)
    abc = komb
    sonuclar = []
    z=0
    xyz = []
    for x in komb:
        f = 0
        xyz.append(x)
        z=z+1
    sayac   = ""
    maxbul  = []
    maxlist = []
    sonliste = []
    print("Edge Rupture Degree")
    for x in xyz:
```

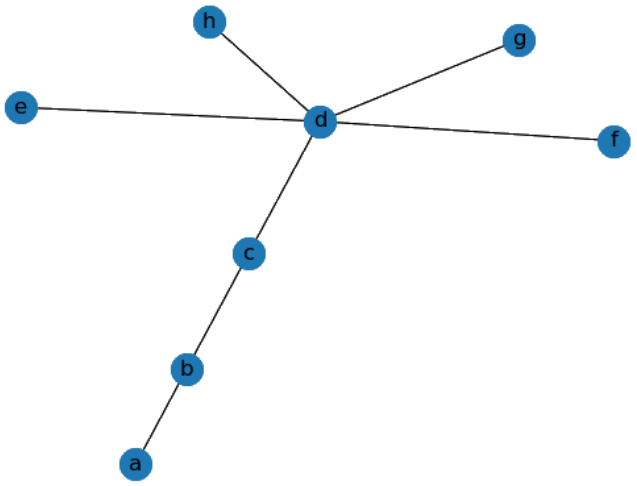
```
            copy_list = copy.deepcopy(othergraph)
            graf = createGraph(copy_list)
            i = 0
            sonliste.append(x)
            for say in x:
                sayac  = sayac + str(say)
                i = i + 1
                if(i != len(x)):
                    sayac  = sayac + ", "
            snc = edgesnumber(graf,x,edges)
            if snc != 'none':
                sonuclar.append(sonucal(graf, x, edges))
                dizi      = snc.split("*")
                esdeger   = dizi[0]
                maxbul.append(dizi[0])
                maxlist.append([sayac])
                dizis = [maxbul,sayac]
                print("Graftan S|"+sayac+"| çıkarılınca sonuç= "+dizi[1])
            sayac = ""
        nums = [int(s) for s in maxbul]
        maksdeger = max(nums)
        maksindex = nums.index(max(nums))
        formul = ""
        i = 1
        for sn in sonuclar:
            if i!= len(sonuclar):
                formul = formul + sn + ","
            else:
                formul = formul + sn
            i = i+1
        print("\nr'(G) = max{"+formul+"} = "+str(maksdeger))
        print("\nr'(G) maximum değeri = "+ str(maksdeger)+", çıkarılan edge listesi S|"+
    str(maxlist[maksindex])+"|")
        cgc = copy.deepcopy(othergraph)
        cgraph2 = createGraphClass(cgc)
        edges2 = cgraph2.edges()
        vertices2 = cgraph2.vertices()
        cgg = copy.deepcopy(othergraph)
        cgraph2 = createGraph(cgg)
        G2 = nx.Graph()
        for vt in vertices2:
            G2.add_node(vt)
        yenilist  = sonliste[maksindex]
        songoster = cikarilmisedge(cgraph2,yenilist,edges2)
        for gr2 in songoster:
            for g2 in songoster[gr2]:
                G2.add_edge(str(gr2),g2)
        nx.draw_networkx(G2)
        plt.show()
```

Bu algoritmada tüm iterasyonları dolaştığımız döngünün içerisinde bulunan "sonucal" ve "edgesnumber" fonksiyonlarının döngülerinin zaman karmaşıklık değeri $O(n^2)$'dir. En dışta bulunan döngünün ise maliyeti $O(n^3)$'tür. Böylece algoritmanın zaman karmaşıklık değeri $O(n^3)$'tür.

5.3 Ayrıt Kopma Derecesi Algoritma Uygulaması

Ayrıt kopma derecesi algoritması özel çizgelerde uygulanmıştır.

5.3.1. Kuyruklu Yıldız Çizge Ayrıt Kopma Derecesi



Komşuluklar

a tepesi için {b},

b tepesi için {a,c },

c tepesi için {b,d},

d tepesi için {c,e,f,g,h},

e tepesi için {d},

f tepesi için {d},

g tepesi için {d},

h tepesi için {d} tepeleri komşu tepelerdir.

Ayrıt kopma derecesi tanımından yola çıkarsak,

$$r'(G) = \max\{\omega(G - S) - |S| - m(G - S) : S \subseteq E(G), \omega(G - S) > 1\}$$

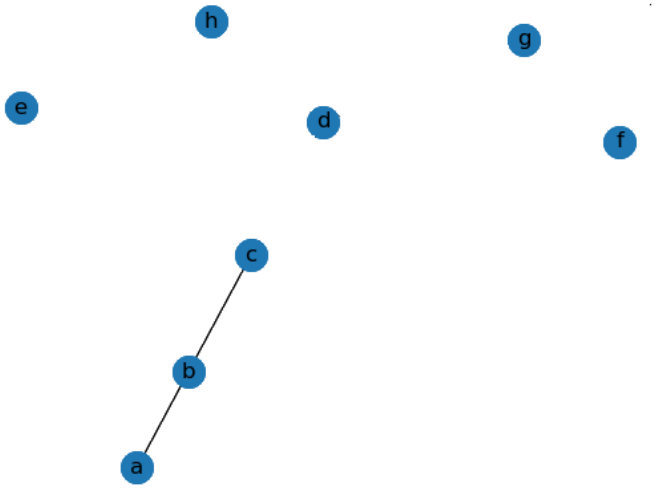
$\omega(G - S)$, bileşen sayısını,

$m(G - S)$, en büyük bileşenin tepe sayısını,

$|S|$ ise çıkarılan tepeleri gösterir.

Tüm iterasyonlar denenerek en uygun sonuç bulunur.

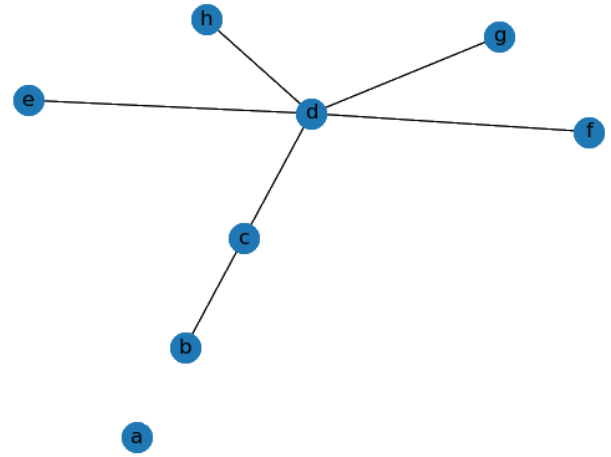
$|S_1| = \{ [c, d], [d, e], [d, f], [d, g], [d, h]\}$ ayrıtı çıkarılması durumu için,



c ile d, d ile e, d ile f, d ile g ve d ile h tepeleri arasındaki ayrıtlar çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda bileşen sayısı, en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısından, çizgeden çıkarılan ayrıt sayısını ve en büyük bileşenin tepe sayısını çıkararak ayrıt kopma derecesini elde ederiz. O halde,

$m(G - S) = 3,$
$\omega(G - S) = 6,$
$|S| = 5,$
$r'(G) = \omega(G - S) - |S| - m(G - S) = -2$'dır.
Bileşen sayısı 6, çıkarılan ayrıt sayısı 5, en büyük bileşenin tepe sayısı 3'dür.
Bulunan sonuç ise 6 - 5 - 3 = -2'dir.
Bu şekilde tek tek çıkarılabilecek tüm iterasyonları deneyip en verimli sonuca ulaşılmaktadır.

$|S_2| = \{a, b\}$ ayrıtı çıkarılması durumu için,



a ile b tepeleri arasındaki ayrıt çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda bileşen sayısı, en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısından, çizgeden çıkarılan ayrıt sayısını ve en büyük bileşenin tepe sayısını çıkararak ayrıt kopma derecesini elde ederiz. O halde,

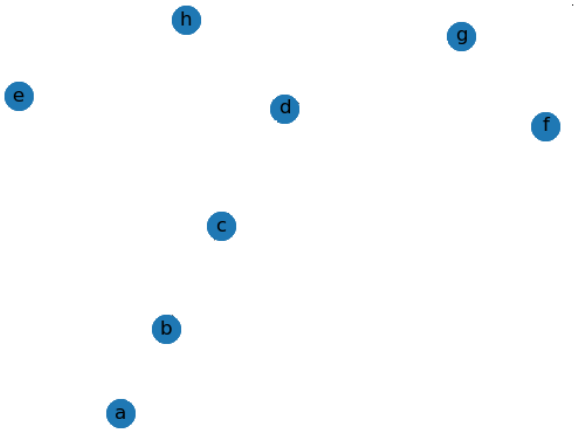
$$m(G - S) = 1, \omega(G - S) = 8, |S| = 7,$$

$$r'(G) = \omega(G - S) - |S| - m(G - S) = 0\text{'dır.}$$

Bileşen sayısı 8, çıkarılan ayrıt sayısı 7, en büyük bileşenin tepe sayısı 1'dir.

Bulunan sonuç ise 8 - 7 - 1 = 0'dır.

$|S_3| = \{[a, b], [c, b], [c, d], [d, e], [d, f], [d, g], [d, h]\}$ ayrıtları çıkarılması durumu için,



a ile b, c ile b, c ile d, tepeleri arasındaki ayrıt çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda bileşen sayısı, en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısından, çizgeden çıkarılan ayrıt sayısını ve en büyük bileşenin tepe sayısını çıkararak ayrıt kopma derecesini elde ederiz. O halde,

$m(G - S) = 1$,

$\omega(G - S) = 8$,

$|S| = 7$,

$r'(G) = \omega(G - S) - |S| - m(G - S) = 0$'dır.

Bileşen sayısı 8, çıkarılan ayrıt sayısı 7, en büyük bileşenin tepe sayısı 1'dir.

Bulunan sonuç ise 8 - 7 - 1 = 0'dır.

Tüm iterasyonların denenmesi ile çıkan sonuç ise,

$$r'(G) = \max\{r_1, r_2, r_3, \dots\} = \max\{-5, -4, -3, \dots, 0\} = 0\text{'dır.}$$

6. AYRIT BÜTÜNLÜĞÜ (EDGE INTEGRITY)

Bu bölümde, ayrıt bütünlüğünün tanımı, hesaplanan bazı teoremler verilmiştir. Daha sonra bazı özel çizgelerin ayrıt bütünlüğü sayısı hesaplanmıştır.

6.1 Tanım ve Teoremler

Tanım 6.1.1 [1] Bağlantılı bir G çizgesinde S kümesi ve ona bitişik tüm ayrıtların çizgeden çıkarıldıktan sonra geriye kalan çizgedeki en büyük boyutlu bileşenin tepe sayısı $m(G - S)$ olmak üzere ayrıt bütünlük değeri,

$$I'(G) = \min\{|S| - m(G - S) : S \subseteq E(G)\}$$

burada G - S içinde $\omega(G - S)$ bileşen sayısını ve $m(G - S)$ en büyük bileşenin tepe sayısını gösterir.

Teorem 6.1.2 [1] Eğer H, G'nin alt çizgesi ise $I'(H) \leq I'(G)$

Teorem 6.1.3 [1] T, p ve p'nin yolu olan P_p'nin ağaç yapısı olsun.

$$\left[\sqrt[2]{p}\right] - 1 = I'(P_p) \leq I'(T)$$

Teorem 6.1.4 [1] $I'(K_p) = p$

Teorem 6.1.5 [1] $I'(K_{1,n}) = n + 1$

Teorem 6.1.6 [1] Herhangi bir G çizgesi için, $I'(G) \geq d(G) + 1$

Teorem 6.1.7 [1] $I'(K_p) + I'(\overline{K}_p) = p + 1$

Teorem 6.1.8 [1] $p \geq 5$ için, $p \geq 5, I'(P_p) + I'(\overline{P}_p) = p + \left[\sqrt[2]{p}\right] - 1$

Teorem 6.1.9 [1] Eğer $m \leq n$ ise $I'(K_{m,n}) + I'(\overline{K}_{m,n}) = m + 2n$

Tablo 1: [2] Bazı Yaygın Özel Çizgelerin Tepe ve Ayrıt Bütünlükleri

Graph G	Vertex-integrity $I(G)$	Edge-integrity $I'(G)$
Complete graph K_p	p	p
Null graph $\bar{K}_p$	1	1
Star $K_{1,n}$	2	$n+1$
Path P_n	$\lceil 2\sqrt{n+1} \rceil - 2$	$\lceil 2\sqrt{n} \rceil - 1$
Cycle C_n	$\lceil 2\sqrt{n} \rceil - 1$	$\lceil 2\sqrt{n} \rceil$
Complete bipartite graph $K_{m,n}$	$1 + \min\{m, n\}$	$m + n$
n-cube Q_n	$(?)$	2^n

6.2 Ayrıt Bütünlüğü Kod Yapısı

Ayrıt bütünlüğü ilk olarak kullanılacak olan kütüphaneler "import" komutu ile tanımlanmıştır. Bu kütüphaneler kombinasyon, dizi, bağlantı oluşturma, görsel olarak ekranda gösterme ve kopyalama kütüphaneleridir.

```
from itertools import combinations
import array as arr
import networkx as nx
import matplotlib.pyplot as plt
import copy
```

Çizge üzerinde veri elde etmek için "Graph" sınıfı oluşturulmuştur. Bu sınıfta tepeleri ve ayrıtları liste olarak geri döndürme, tepe ekleme, ayrıt ekleme, ayrıt komşuluklarını tanımlama gibi bazı fonksiyonlar mevcuttur.

```
class Graph(object):
    def __init__(self, cizgesoz=None):
        if cizgesoz == None:
            cizgesoz = {}
        self.__cizgesoz = cizgesoz
```

```
    def Tepeler(self):
        return list(self.__cizgesoz.keys())
    def Ayritlar(self):
        return self.AyritlariOlustur()
    def TepeEkle(self, tepe):
        if tepe not in self.__cizgesoz:
            self.__cizgesoz[tepe] = []
    def AyritEkle(self, ayrit):
        ayrit = set(ayrit)
        (tepe1, tepe2) = tuple(ayrit)
        if tepe1 in self.__cizgesoz:
            self.__cizgesoz[tepe1].append(tepe2)
        else:
            self.__cizgesoz[tepe1] = [tepe2]
    def AyritlariOlustur(self):
        ayritdizi = []
        for tepe in self.__cizgesoz:
            for komsu in self.__cizgesoz[tepe]:
                if {komsu, tepe} not in ayritdizi:
                    ayritdizi.append({tepe, komsu})
        return ayritdizi
```

Bileşen sayısı $\omega(G - S)$ bulmak için tanımlı "KokAl" fonksiyonu aşağıdaki gibidir.

```
def KokAl(aKoms):
    def KokBul(aTepe,aKok):
        while aTepe != aKok[aTepe][0]:
            aTepe = aKok[aTepe][0]
        return (aTepe,aKok[aTepe][1])
    kokdizi = {}
    for TepeNok in aKoms.keys():
        kokdizi[TepeNok] = (TepeNok,0)
    for i in aKoms:
        for j in aKoms[i]:
            (kokdizi_i,derinliki) = KokBul(i,kokdizi)
            (kokdizi_j,derinlikj) = KokBul(j,kokdizi)
            if kokdizi_i != kokdizi_j:
                Mindeger = kokdizi_i
                Maksdeger = kokdizi_j
                if  derinliki > derinlikj:
                    Mindeger = kokdizi_j
                    Maksdeger = kokdizi_i
                kokdizi[Maksdeger] =
(Maksdeger,max(kokdizi[Mindeger][1]+1,kokdizi[Maksdeger][1]))
                kokdizi[Mindeger] = (kokdizi[Maksdeger][0],-1)
    yenidizi = {}
    for i in aKoms:
        if kokdizi[i][0] == i:
            yenidizi[i] = []
    for i in aKoms:
```

```
        yenidizi[KokBul(i,kokdizi)[0]].append(i)
    return yenidizi
```

Bu fonksiyonda tüm çizge döngü yardımı ile dolaşılarak komşulukları kontrol edilir ve bileşenler bulunur. Döngüler içerisinde kullanılmak üzere "KokBul" fonksiyonu tanımlanmıştır.

Döngü içerisinde "KokBul" fonksiyonu yardımı ile bulunan sonuçlar "yenidizi" dizisi içerisine eklenir ve dizi geri döndürülür. Fonksiyon yardımı ile Bileşenleri bulunan çizge, "len(KokAl(yenigraph))" komutu ile bileşenlerin sayısı bulunur ve böylece $m(G - S)$ bulunmuş olur.

```
def cikarilmisedge(cgraf, edge2,edges):
    list1 = []
    list3 = []
    graf2 = cgraf
    for nk in edge2:
        list1 = []
        for nsy in nk:
            list1.append(nsy)
        list3.append(list1)
    for lx in list3:
        dzlist2 = []
        say2 = 0
        for tx in lx:
            dzlist2.append(tx)
        if str(dzlist2[say2 + 1]) != "" and str(dzlist2[say2 + 1]) in
graf2[str(dzlist2[say2])]:
            graf2[str(dzlist2[say2])].remove(str(dzlist2[say2 + 1]))
        if str(dzlist2[say2]) != "" and str(dzlist2[say2]) in graf2[str(dzlist2[say2 + 1])]:
            graf2[str(dzlist2[say2 + 1])].remove(str(dzlist2[say2]))
    yenigraph2 = graf2
    return yenigraph2
```

"cikarilmisedge" fonksiyonu, çıkarılacak olan ayrıtı çıkarıp, çıkarılmış hali ile geri döndürmektedir.
Bu fonksiyonda çizge, çıkarılacak olan ayrıt ve çizgedeki ayrıtları içeren dizi kullanılmıştır.
Döngü yardımı ile tüm tepeler dolaşılarak çıkarılacak olan ayrıt bulunur ve "remove" yardımı ile çıkarılır. Son hali geri döndürülür.

```
def edgesnumber(graph, edge,edges):
    list  = []
    list2 = []
    graf = graph
    s = len(edge)
    sonuc = 0
    for n in edge:
        list= []
```

```
        for ns in n:
            list.append(ns)
        list2.append(list)
    for l in list2:
        dzlist = []
        say =0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say+1]) != "" and str(dzlist[say+1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say+1]))
        if str(dzlist[say]) != ""  and str(dzlist[say]) in graf[str(dzlist[say+1])]:
            graf[str(dzlist[say+1])].remove(str(dzlist[say]))
    yenigraph = graf
    m =  KokAl(yenigraph)
    sq = 0
    asd = []
    for i in m:
        sycc = 0
        for j in m[i]:
            sycc = sycc + 1
        asd.append(sycc)
        sq = sq + 1
    maxdeger = 0
    for f in asd:
        if f > maxdeger:
            maxdeger = f
    sonuc = maxdeger + s
    mgs= "m(G-S) = " + str(maxdeger)
    ss = "|S| = " + str(s)
    esg = "I'(G) = |S| + m(G-S) = " + str(sonuc)
    goster = mgs + " " + ss + " " + esg
    veri = str(sonuc) + "*" + goster + "*" + str(yenigraph)
    return veri
```

“edgesnumber” fonksiyonu çizge üzerinde çıkarılması gereken ayrıtı çıkarıp, döngüler yardımı ile sonucu bulmamızı sağlamaktadır. Çıkarılan ayrıt sonrasında çizge içerisinde döngü ile tüm tepeler dolaşılarak $m(G - S)$ bulunmaktadır.

Bu fonksiyon ana fonksiyonumuz olan “EdgeIntegrity” fonksiyonunda kullanılmak üzere tanımlanmıştır. Gerekli parametreler “*” ile birleştirilerek geri döndürülmüştür.

```
def sonucal(graph, edge,edges):
    list  = []
    list2 = []
    graf = graph
    s = len(edge)
    sonuc = 0
    for n in edge:
        list= []
        for ns in n:
```

```
            list.append(ns)
        list2.append(list)
    for l in list2:
        dzlist = []
        say =0
        for t in l:
            dzlist.append(t)
        if str(dzlist[say+1]) != "" and str(dzlist[say+1]) in graf[str(dzlist[say])]:
            graf[str(dzlist[say])].remove(str(dzlist[say+1]))
        if str(dzlist[say]) != ""  and str(dzlist[say]) in graf[str(dzlist[say+1])]:
            graf[str(dzlist[say+1])].remove(str(dzlist[say]))
    yenigraph = graf
    m =  KokAl(yenigraph)
    sq = 0
    asd = []
    for i in m:
        sycc = 0
        for j in m[i]:
            sycc = sycc + 1
        asd.append(sycc)
        sq = sq + 1
    maxdeger = 0
    for f in asd:
        if f > maxdeger:
            maxdeger = f
    sonuc = maxdeger + s
    return str(sonuc)
```

“sonucal” fonksiyonu sırası ile çizgeden ayrıtları çıkartıp ayrıt bütünlüğünün sonucunu döndürmektedir.

```
def EdgeIntegrity(othergraph):
    crg = copy.deepcopy(othergraph)
    graph = createGraphClass(crg)
    edges = graph.Ayritlar()
    vertices = graph.Tepeler()
    cg = copy.deepcopy(othergraph)
    graph = createGraph(cg)
```

Çizgenin ilk halini görsel olarak ekranda gösteren kod yapısı aşağıdaki gibidir.

```
 G = nx.Graph()
    for vert in vertices:
        G.add_node(vert)
    for gr in graph:
        for g in graph[gr]:
            G.add_edge(str(gr),g)
    nx.draw_networkx(G)
    plt.show()
```

Tüm kombinasyonları belirleyip dizi içerisine atan kod yapısı aşağıdaki gibidir.

```
komb = []
abc  = []
z = 0
for i in range(1, len(edges)+1):
   iBoyluKomb = [list(x) for x in combinations(edges, i)]
   komb.extend(iBoyluKomb)
abc = komb

sonuclar = []
z=0
xyz = []
for x in komb:
   f = 0
   xyz.append(x)
   z=z+1
sayac   = ""
maxbul  = []
minbul  = []
maxlist = []
sonliste = []
```

Tüm kombinasyonları döngü yardımı ile işleme alan kod yapısı aşağıdaki gibidir.

```
for x in xyz:
   copy_list = copy.deepcopy(othergraph)
   graf = createGraph(copy_list)
   i = 0
   sonliste.append(x)
   for say in x:
      sayac = sayac + str(say)
      i = i + 1
      if(i != len(x)):
         sayac = sayac + ","
   snc = edgesnumber(graf,x,edges)
   sonuclar.append(sonucal(graf, x, edges))
   dizi      = snc.split("*")
   esdeger   = dizi[0]
   minbul.append(dizi[0])
   maxlist.append([sayac])
   dizis = [maxbul,sayac]
   print("Çizgeden S|"+sayac+"| çıkarılınca sonuç= "+dizi[1])
   sayac = ""

nums = [int(s) for s in minbul]
maksdeger = max(nums)
mindeger  = min(nums)
maksindex = nums.index(max(nums))
minindex = nums.index(min(nums))
```

```
    formul = ""
    i = 1
    for sn in sonuclar:
        if i!= len(sonuclar):
            formul = formul + sn + ","
        else:
            formul = formul + sn
        i = i+1

    print("\nI'(G) = min{"+formul+"} = "+str(mindeger))
    print("\nI'(G) minimum değeri = "+ str(mindeger)+", çıkarılan edge listesi S|"+
str(maxlist[minindex])+"|")
```

Bu algoritmada tüm iterasyonları dolaştığımız döngünün içerisinde bulunan "sonucal" ve "edgesnumber" fonksiyonlarının döngülerinin zaman karmaşıklık değeri $O(n^2)$'dir. En dışta bulunan döngünün ise maliyeti $O(n^3)$'tür. Böylece algoritmanın zaman karmaşıklık değeri $O(n^3)$'tür.

Sonuçları ekranda görsel olarak gösteren kod yapısı aşağıdaki gibidir.

```
    cgc = copy.deepcopy(othergraph)
    cgraph2 = createGraphClass(cgc)
    edges2 = cgraph2.Ayritlar()
    vertices2 = cgraph2.Tepeler()
    cgg = copy.deepcopy(othergraph)
    cgraph2 = createGraph(cgg)
    G2 = nx.Graph()
    for vt in vertices2:
        G2.add_node(vt)

   yenilist  = sonliste[minindex]
   sonucgrafi = cikarilmisedge(cgraph2,yenilist,edges)

   for gr2 in sonucgrafi:
      for g2 in sonucgrafi[gr2]:
          G2.add_edge(str(gr2),g2)
   nx.draw_networkx(G2)
   plt.show()
```

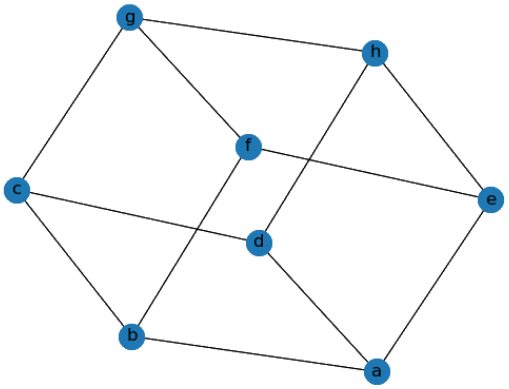
"EdgeIntegrity" fonksiyonu ilk olarak ayrıtları ve tepe bağlantılarını belirleyip çizgenin ilk halini görsel olarak ekrana yazdırmaktadır. Sonrasında tüm kombinasyonları belirleyip bir dizi içerisine atarak, döngü ile tüm iterasyonlara yöntemi uygulamaktadır.

Tüm iterasyonların sonuçları ekrana yazdırılmaktadır. En uygun sonuç belirlenerek tüm sonuçların kümesi ile en uygun sonuca ait çizge hem görsel hem de yazısal olarak ekranda gösterilmektedir.

6.3. Ayrıt Bütünlüğü Algoritma Uygulaması

Ayrıt bütünlüğü algoritması özel çizgelerde uygulanmıştır.

6.3.1. Küp Çizge Ayrıt Bütünlüğü



Komşuluklar

a tepesi için {b,d,e},

b tepesi için {a,c,f},

c tepesi için {b,d,g},

d tepesi için {a,c,h},

e tepesi için {a,f,h},

f tepesi için {b,e,g},

g tepesi için {c,f,h},

h tepesi için {d,e,g} tepeleri komşu tepelerdir.

Ayrıt bütünlüğü tanımından yola çıkarsak,

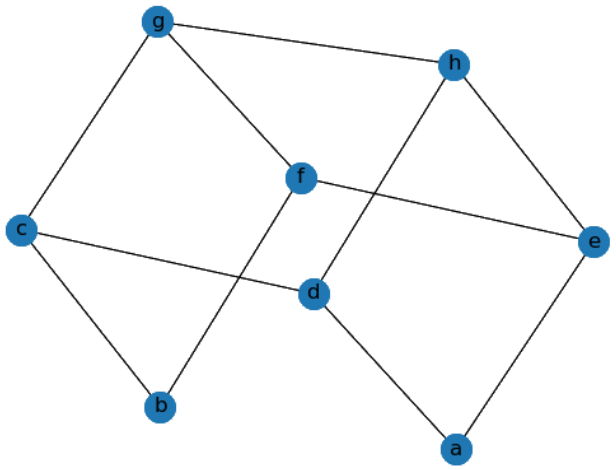
$$I'(G) = \min\{|S| + m(G - S) : S \subseteq E(G)\}$$

$m(G - S)$, en büyük bileşenin tepe sayısını,

$|S|$ ise çıkarılan tepeleri gösterir.

Tüm iterasyonlar denenerek en uygun sonuç bulunur.

$|S_1| = \{ a, b \}$ ayrıtı çıkarılması durumu için,



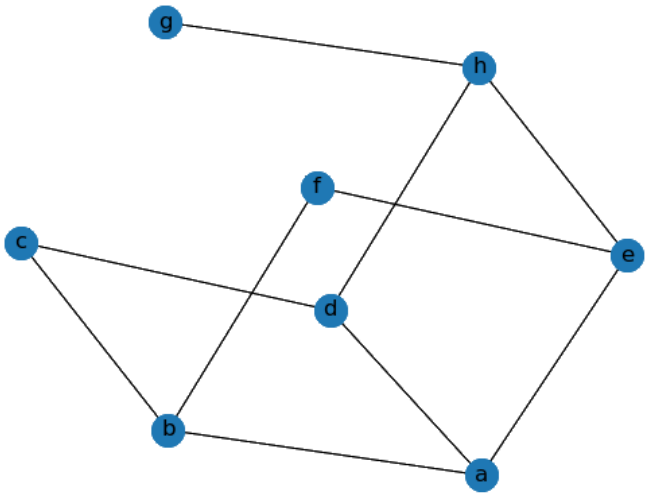
a ile b tepeleri arasındaki ayrıt çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısı ile çıkarılan ayrıt sayısını topladığımızda ayrıt bütünlüğünü elde ederiz. O halde,

$m(G - S) = 8, |S| = 1, I'(G) = |S| + m(G - S) = 9$'dur.

Çıkarılan ayrıt sayısı 1, en büyük bileşenin tepe sayısı 8'dır.

Bulunan sonuç ise 1+8=9'dir.

$|S_2| = \{ [g, c], [g, f] \}$ ayrıtlarının çıkarılması durumu için,



g ile c, g ile f tepeleri arasındaki ayrıt çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısı ile çıkarılan ayrıt sayısını topladığımızda ayrıt bütünlüğünü elde ederiz. O halde,

$m(G - S) = 8,$

$|S| = 2,$

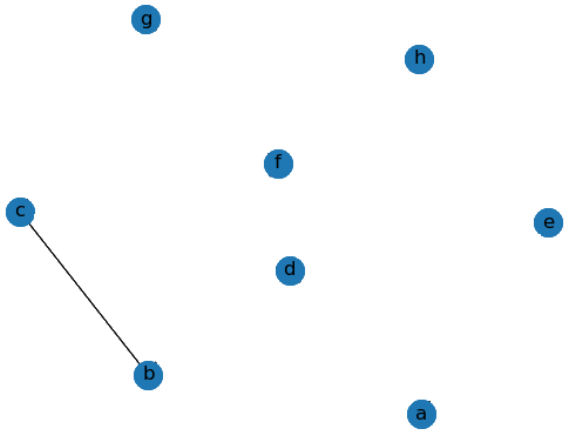
$I'(G) = |S| + m(G - S) = 10$'dur.

Çıkarılan ayrıt sayısı 2, en büyük bileşenin tepe sayısı 8'dır.

Bulunan sonuç ise 2+8=10'dir.

$|S_3| = \{ [a, b], [a, d], [a, e], [b, f], [d, c], [g, c], [d, h], [f, e], [h, e], [g, f], [g, h] \}$

ayrıtlarının çıkarılması durumu için,



a ile b, a ile d, a ile e, b ile f, d ile c, g ile c, d ile h, f ile e, h ile e, g ile f, g ile h tepeleri arasındaki ayrıtlar çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısı ile çıkarılan ayrıt sayısını topladığımızda ayrıt bütünlüğünü elde ederiz. O halde,

$m(G - S) = 2,$

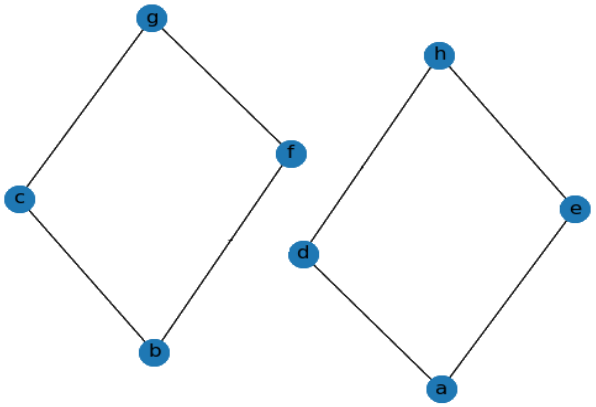
$|S| = 11,$

$I'(G) = |S| + m(G - S) = 13$'dur.

Çıkarılan ayrıt sayısı 11, en büyük bileşenin tepe sayısı 2'dır.

Bulunan sonuç ise 2+11=13'dir.

$|S_4| = \{ [a, b], [c, d], [e, f], [g, h] \}$ ayrıtı çıkarılması durumu için,



a ile b, c ile d, e ile f, g ile h tepeleri arasındaki ayrıtlar çıkarıldığında tepelerin birbirleri ile bağlantıları kesilmiş olur. Kesilen bağlantı sonucunda en büyük bileşenin tepe sayısını ve çıkarılan ayrıt sayısını elde etmiş oluruz. Bileşen sayısı ile çıkarılan ayrıt sayısını topladığımızda ayrıt bütünlüğünü elde ederiz. O halde,

$m(G - S) = 4,$

$|S| = 4,$

$I'(G) = |S| + m(G - S) = 8$'dir.

Çıkarılan ayrıt sayısı 4, en büyük bileşenin tepe sayısı 4'dür.

Bulunan sonuç ise 4+4=8'dir.

Bu şekilde tek tek çıkarılabilecek tüm iterasyonları deneyip en verimli sonuca ulaşılmaktadır.

Tüm iterasyonların denenmesi ile çıkan sonuç ise,

$$I'(G) = \min\{I_1, I_2, I_3, \dots\} = \min\{8,9,10,11,12, \dots\} = 8\text{'dir.}$$

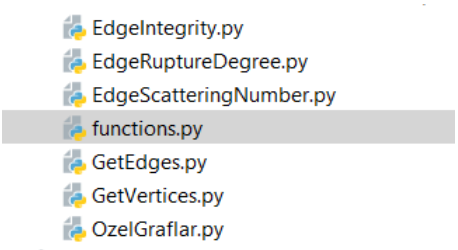
7. SONUÇLAR

Ağ tasarımında en önemli kısımlar güvenilirlik ve verimliliktir. Ağ tasarımında dayanıklılık önemli bir kriterdir.

Bir çizgenin ayrıt saçılma sayısı, ayrıt kopma derecesi ve ayrıt bütünlüğü, o çizgeyi daha iyi anlamak için kullanılan yöntemlerdir. Çizgenin tepelerinin birbirleri ile bağlantılarını ve o bağlantılarının durumlarını anlama açısından önemli bir rol oynar. Komşuluklar arasındaki zedelenebilirlik ölçümlerini tüm olasılıkları ile hesaplar.

Bazı özel çizgeler üzerinde bu yöntemler denenmiştir. Bu yöntemler denenirken çizgelerin tüm tepelerinin komşulukları arasındaki tüm kesme olasılıkları göz önüne alınmıştır.

Yöntemlere ait algoritmalar Python'da uygulanmıştır. Algoritmalar Graph Plotting ile analiz edilmiştir. Her bir yöntem için ayrı algoritma uygulanmıştır. Kullanmış olduğumuz Edge Scattering Number, Edge Integrity, Edge Rupture Degree yöntemleri için ayrı ayrı sayfalar içinde oluşturulan sınıflar ile, kendi tanımlarına uygun 3 ayrı sayfada fonksiyon olarak çağırılacak şekilde kod yazılmıştır. Özel çizgelerin tanımlandığı özel çizge sayfası ve her bir çizge için ayrıtlarını ve tepelerini gösteren 2 ayrı sayfada 2 ayrı fonksiyon içeren kod yazılmıştır. Bu tanımlamaların ardından her yöntemi üzerinde çalıştıracağımız functions adında bir python sayfası oluşturulmuştur. Oluşturulan sayfalar aşağıdaki gibidir.



Şekil 7.1. Tanımlı yöntemlere ait sayfalar

Tüm oluşturulan sayfalar ve içerisindeki fonksiyonları kullanılabilmesi için functions sayfası içerisinde import edilmiştir.

```
from EdgeScatteringNumber import *
from EdgeRuptureDegree import *
from EdgeIntegrity import *
from GetVertices import *
from GetEdges import *
from OzelGraflar import *

```

Şekil 7.2. Ana fonksiyona yöntemlerin dahil edilmesi

Dahil etme işleminden sonra her yöntem fonksiyonu çağrılarak çalıştırılabilmektedir. Çağırılma işleminde yöntem fonksiyonun adı ve gönderilecek olan özel çizge belirtilmesi yeterlidir.

EdgeScatteringNumber(kupcizge) şeklinde belirtirsek küp çizgenin ayrıt saçılma sayısını, EdgeRuptureDegree(completegraph) şeklinde belirtirsek tam çizgenin ayrıt kopma derecesini, EdgeIntegrity(stargraph) şeklinde belirtirsek ise yıldız çizgenin ayrıt bütünlüğünü hesaplatmış oluruz.

Özel çizgelerin ayrıt ve tepeleri için GetVertices(stargraph) şeklinde belirtirsek yıldız çizgenin tepelerini, GetEdges(stargraph) şeklinde belirtmemiz durumunda ise yıldız çizgenin ayrıtlarını gösterir. Sonuçlar aşağıdaki gibidir.

```
stargraph = {
        "a": ["g"],
        "b": ["g"],
        "c": ["g"],
        "d": ["g"],
        "e": ["g"],
        "f": ["g"],
        "g": ["a", "b", "c", "d", "e", "f"]
}
```

Şekil 7.3. Yıldız çizge tanımlaması

```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

Şekil 7.4. Yıldız çizgenin tepeleri

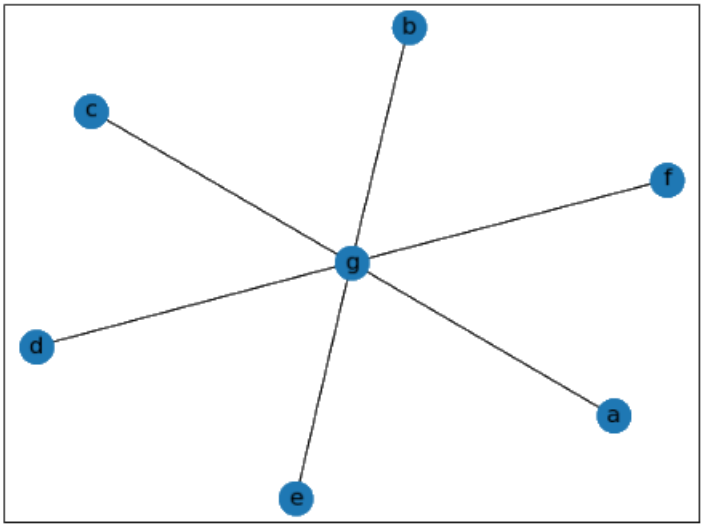
```
[{'g', 'a'}, {'g', 'b'}, {'c', 'g'}, {'g', 'd'}, {'g', 'e'}, {'g', 'f'}]
```

Şekil 7.5. Yıldız çizgenin ayrıtları

Ağlarda güvenlik sorunlarını belirlemek için algoritmaların nasıl uygulanabileceğini, verilen zedelenebilirlik parametresinin nasıl fayda sağlayabileceği belirtilmiştir. Çalışmanın amacı, tanımlanan bazı özel çizgelerde kullanılan yöntemler ile zedelenebilirlik ölçümleri ve her ihtimali düşünerek en uygun sonuca varılmasıdır.

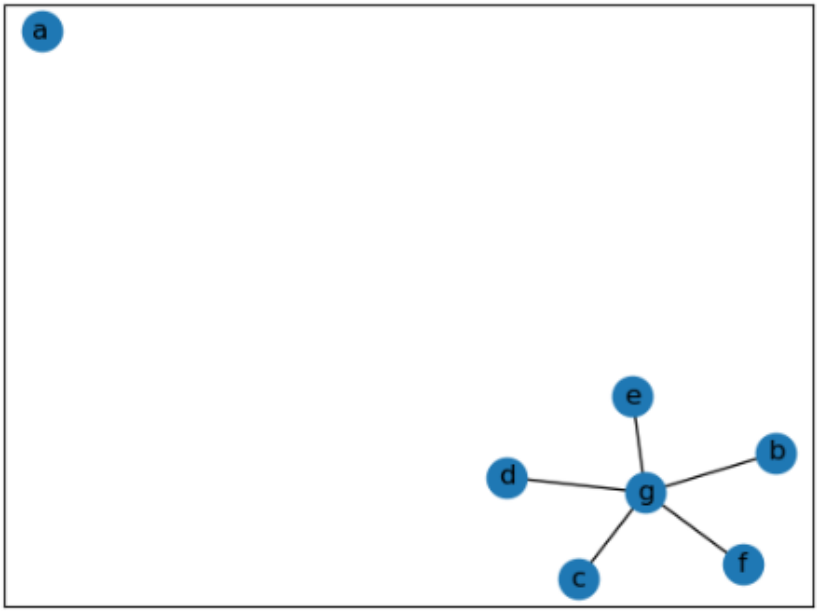
Bu çalışmada, ayrıt saçılma sayısı, ayrıt bütünlüğü ve ayrıt kopma derecesi özel çizgeler üzerinde test edilmiştir.

EdgeScatteringNumber(stargraph) yani yıldız çizgenin ayrıt saçılma sayısında elde ettiğimiz sonuçlar aşağıdaki gibidir.



Şekil 7.6. Yıldız çizgenin gösterimi

Fonksiyon çağırıldığında yıldız çizgenin kendisi ekrana gelmektedir ve ardından tüm komşuluklar ile tanıma uygun olası durumlar hesaplanarak elde ettiğimiz sonuç gözükmektedir.



Şekil 7.7. Yıldız çizgesinin ayrıt saçılma sayısı sonucu çizge gösterimi

```
Graftan S|{'g', 'a'}| çıkarılınca sonuç= w(G-S)= 2 |S|= 1 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'}| çıkarılınca sonuç= w(G-S)= 2 |S|= 1 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'}| çıkarılınca sonuç= w(G-S)= 2 |S|= 1 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 2 |S|= 1 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 2 |S|= 1 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 2 |S|= 1 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 3 |S|= 2 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
```

Şekil 7.8. Yıldız çizgesinin ayrıt saçılma sayısı tüm kombinasyon hesaplamaları

```
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 4 |S|= 3 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'd'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'c'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 5 |S|= 4 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'd'},{'g', 'e'}| çıkarılınca sonuç= w(G-S)= 6 |S|= 5 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'd'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 6 |S|= 5 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 6 |S|= 5 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 6 |S|= 5 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'c'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 6 |S|= 5 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'b'},{'g', 'c'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 6 |S|= 5 es(G) = w(G-S) - |S| = 1
Graftan S|{'g', 'a'},{'g', 'b'},{'g', 'c'},{'g', 'd'},{'g', 'e'},{'g', 'f'}| çıkarılınca sonuç= w(G-S)= 7 |S|= 6 es(G) = w(G-S) - |S| = 1
```

Şekil 7.9. Yıldız çizgesinin ayrıt saçılma sayısı tüm kombinasyon hesaplamaları

Şekil 7.8. ve Şekil 7.9'da belirtildiği üzere, yönteme uygun tüm olası durumlar çıkarılarak, tüm komşuluklar üzerinde en iyi sonuç bulunması sağlanmıştır.

```
es(G) = max{1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1} = 1
```

Şekil 7.10. Yıldız çizgesinin ayrıt saçılma sayısı sonucu

Şekil 7.10.'da uygulanan tüm olası durumlar sonucunda maksimum değer 1 bulunmuştur.

```
es(G) maximum değeri = 1, çıkarılan edge listesi S|["{'g', 'a'}"]|
```

Şekil 7.11. Yıldız çizgesinin ayrıt saçılma sayısı sonucu çıkarılan ayrıtlar

Şekil 7.11.'de ise bu işlemler sonucunda çıkarılan ayrıtlar belirtilmiştir.

Tablo 2. Bazı Özel Çizgelerin Sonuçları

Özel Çizgeler	Ayrıt Saçılma Sayısı	Ayrıt Kopma Derecesi	Ayrıt Bütünlüğü
Yıldız Çizge	1	0	7
Yol Çizge	1	0	3
Dikenli Çizge	1	0	4
Çevre Çizge	0	-1	5

Tablo 2'de bazı özel çizgelerin yöntemler üzerinde sonuçları belirtilmiştir. Tüm iterasyonlar yöntemlere uygun bir şekilde uygulanarak aralarından en verimli, en doğru veri alınarak sonuca ulaşılmıştır. Tüm komşuluk iterasyonları bu yöntemler üzerinde uygulanarak bize en verimli, en doğru sonuçları vermektedir.

Çizgenin bağlantıları arasında herhangi bir kopma ya da bir hasar durumunda bu verilere bakarak kolaylıkla bir çözüm bulabilir ve durum analizi yapabiliriz.

Uygulamış olduğumuz yöntemlerin kod yapılarında, en dıştaki döngü ve içerisindeki fonksiyonun döngüleri ile birlikte iç içe en fazla 3 döngü bulunmaktadır. Döngü içerisinde bulunan döngülerin maliyeti $O(n^2)$, en dışta bulunan döngünün maliyeti ise $O(n^3)$'tür. Böylece yöntemlerimizin zaman karmaşıklık değerleri $O(n^3)$'tür.

KAYNAKLAR

[1] Laskar, R., Stueckle, S., Piazza, B. On the edge-integrity of some graphs and their complements, 1993, 245-253.

[2] Bagga, K.S., Beineke, L.W., Lipman M.J., Pippert, R.E., Edge-integrity: a survey, Discrete Mathematics, 1994, 3-12.

[3] R. Laskar, S. Stueckle and B. Piazza, On the edge-integrity of some graphs and their complements, 1993, 245-253.

[4] Aslan, E., Edge Neighbor Rupture Degree of a Graph, 2013.

[5] Barefoot, C.A., Entringer, R., Swart, H. C., Integrity of trees and powers of cycles, 1987, 103–114.

[6] Golumbic, M.C., Algorithmic Graph Theory and Perfect Graphs, Computer Science and Applied Mathematics Academic Press, New York, 1980.

[7] Lipman, M.J., Goddard, W., Beineke, L.W., Graphs with Maximum Edge Integrity.

[8] Arumugam, S., Velammal, S., Edge Domination in Graphs, 1998, 173-179

[9] Aslan, E., Kürkçü, Ö., Edge Scattering Number of Gear Graphs, 2015, 25-31

[10] Barefoot, C. A., Entringer, R. and Swart, H. C. 1987a, Vulnerability in graphs- a comparative survey. J. Combin. Math. Combin. Comput., 1: 13–22.

[11] Harary, F., Graph Theory, Addison-Wesley Publishing, New York, USA, 1969, 274 s.

[12] Bondy, J. A., Murty, U.S.R., Graph Theory with Applications, Elsevier Science Publishing, New York, USA, 1976, 264 s.

[13] Woodall, D.R., The binding number of a graph and its Anderson number, J. Combinatorial Theory Ser., 1973, 225–255.