**T.C.**
**ERCİYES ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**
**BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**

# YAZILIM TANIMLI AĞ TABANLI BÜYÜK VERİ AĞ MİMARİLERİ ÜZERİNDE YÜK DENGELEME VE KONTROLÖR YERLEŞTİRME PROBLEMİNİN YAPAY ZEKA YAKLAŞIMLARI İLE ÇÖZÜMÜ

**Hazırlayan**
**Banu ULU**

**Danışman**
**Doç. Dr. Bilal BABAYİĞİT**

**Doktora Tezi**

**Temmuz 2021**
**KAYSERİ**

**T.C.**
**ERCİYES ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**
**BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**

# YAZILIM TANIMLI AĞ TABANLI BÜYÜK VERİ AĞ MİMARİLERİ ÜZERİNDE YÜK DENGELEME VE KONTROLÖR YERLEŞTİRME PROBLEMİNİN YAPAY ZEKA YAKLAŞIMLARI İLE ÇÖZÜMÜ

**(Doktora Tezi)**

**Hazırlayan**
**Banu ULU**

**Danışman**
**Doç. Dr. Bilal BABAYİĞİT**

**Bu çalışma, Erciyes Üniversitesi Bilimsel Araştırma Projeleri Birimi tarafından FDK-2018- 8073 kodlu proje ile desteklenmiştir.**

**Temmuz 2021**
**KAYSERİ**

# BİLİMSEL ETİĞE UYGUNLUK

Bu çalışmadaki tüm bilgilerin, akademik ve etik kurallara uygun bir şekilde elde edildiğini beyan ederim. Aynı zamanda bu kural ve davranışların gerektirdiği gibi, bu çalışmanın özünde olmayan tüm materyal ve sonuçları tam olarak aktardığımı ve referans gösterdiğimi belirtirim.

Banu ULU

İmza

**“Yazılım Tanımlı Ağ Tabanlı Büyük Veri Ağ Mimarileri Üzerinde Yük Dengeleme ve Kontrolör Yerleştirme Probleminin Yapay Zekâ Yaklaşımları ile Çözümü”** adlı Doktora tezi, Erciyes Üniversitesi Lisansüstü Tez Önerisi ve Tez Yazma Yönergesi’ ne uygun olarak hazırlanmıştır.

| **Hazırlayan** | **Danışman** |
| --- | --- |
| Banu ULU | Doç. Dr. Bilal BABAYİĞİT |

**Bilgisayar Mühendisliği ABD Başkanı**

Prof. Dr. Veysel ASLANTAŞ

# TEŞEKKÜR

Tez konusunun ve sınırlarının belirlenmesi, altyapısının oluşturulması aşamalarında bilimsel yetkinliği ile vermiş olduğu katkının yanı sıra maddi ve manevi desteği, bilgi, emek ve hoşgörüsünü cömertçe sunması ve çalışmalar süresince teşvik etmesi sebebiyle değerli danışman hocam Doç. Dr. Bilal BABAYİĞİT'e sonsuz teşekkürlerimi sunuyorum. Hayatım boyunca maddi ve manevi her konuda desteğini ve sevgilerini sunan babam İbrahim DAĞLIOĞLU ve annem FATMA DAĞLIOĞLU'na şükranlarımı sunuyorum. Bana çalışmalarım süresince her türlü yardımı ve fedakârlığı sağlayan, yükümü hafifleten sevgili eşim Burak ULU'ya ve zorlu tempomda sorgulamadan her yerde bana eşlik eden dünyanın en güzel nimeti, varlığına şükrettiğim biricik oğlum Ahmet Kerem ULU'ya teşekkür ediyorum.

Doktora tez çalışmamı YÖK 100/2000 Doktora Burs Programı kapsamında yürütülen 100/2000 Doktora Bursu Programı ile destekleyen Yüksek Öğretim Kurulu Başkanlığına ve Yurt İçi Öncelikli Alanlar Doktora Burs Programı kapsamında yürütülen 2211-C Öncelikli Alanlar Doktora Burs Programı ile destekleyen TÜBİTAK Bilim İnsanı Destekleme Daire Başkanlığı Birimine teşekkür ederim.

Banu ULU

Temmuz 2021, KAYSERİ

**YAZILIM TANIMLI AĞ TABANLI BÜYÜK VERİ AĞ MİMARİLERİ ÜZERİNDE YÜK DENGELEME VE KONTROLÖR YERLEŞTİRME PROBLEMİNİN YAPAY ZEKA YAKLAŞIMLARI İLE ÇÖZÜMÜ**

**Banu ULU**

**Erciyes Üniversitesi, Fen Bilimleri Enstitüsü**
**Doktora Tezi, Temmuz 2021**
**Danışman: Doç. Dr. Bilal BABAYİĞİT**

# ÖZET

Yeni nesil ağ modellerinin gelişmesiyle birlikte artan bilgi işlem kaynağı talebini etkin bir şekilde karşılamak için, iyi organize edilmiş anahtarlarla bağlantılı sunuculardan oluşan büyük veri merkezi (BVM) yapılarına ihtiyaç vardır. Ancak geleneksel ağ yapısı, ağ altyapısı ve operasyon bakımından BVM'nin büyük ölçek, geniş uygulama çeşitliliği, yüksek güç yoğunluğu ve yüksek güvenilirlik gibi gereksinimlerini karşılayamamaktadır. Bu gereksinimleri karşılamak için yazılım tanımlı ağ (YTA) umut vaat edici çözümler sunmaktadır. YTA umut vaat edici çözümlere sahip olmasına rağmen BVM'ler ve çoklu kontrolörlü denetleyicilerin yerleştirilmesi problemleri için geliştirilmesi gerekmektedir. Bu tez çalışmasında, YTA tabanlı BVM'lerde yük dengeleme için çeşitli yapay zekâ teknikleri kullanılmış ve başarımları test edilmiştir. Kullanılan algoritmaların performansı, doğruluk oranı ve cevap süresi açısından karşılaştırılmış ve literatürde ilk defa kullanılan derin öğrenme yaklaşımının diğer yapay zekâ tekniklerine göre oldukça başarılı olduğu görülmüştür. Ayrıca bu tez çalışmasında YTA'da çoklu kontrolör yerleştirme problemi (KYP) incelenmiş ve bu problemin çözümüne genetik algoritma, k-means, optimize k-means, yapay arı koloni algoritması (YAK) uygulanmış ve de hibrit bir yapay arı koloni (Hibrit-YAK) algoritması önerilmiştir. Kontrol sayısı, uçtan uca gecikme en iyi konum tespiti gibi parametrelerde dikkate alınarak önerilen Hibrit-YAK algoritmasının başarımı YAK'a göre genel gecikmenin ve denetleyiciler ile ilgili anahtarlar arasındaki gecikmenin en aza indirilmesi ve kontrolörlerin konumunu en optimum şekilde bulunması konusunda daha iyi çözüm vermiştir.

**Anahtar Kelimeler:** Yazılım Tanımlı Ağ, Büyük Veri Merkezi, Yük Dengeleme, Kontrolör Yerleştirme Problemi, Derin Öğrenme, Yapay Zekâ

**SOLVING LOAD BALANCING AND CONTROLLER PLACEMENT PROBLEMS IN SOFTWARE-DEFİNED NETWORK BASED BIG DATA NETWORK ARCHITECTURES USİNG ARTIFICIAL INTELLIGENCE APPROACHES**

**BANU ULU**

**Erciyes University, Graduate School of Natural and Applied Sciences**
**PhD Thesis, July 2021**
**Supervisor: Assoc. Prof. Dr. Bilal BABAYİĞİT**

## ABSTRACT

With the development of new generation networking model, the amount of data transfer has gradually increased. To effectively meet the increasing demand for computing resources, big data center network (DCN) structures consisting of well-organized switches of interconnected servers are needed. However, traditional network structures cannot meet the DCN requirements such as large scale, wide application variety, high power density and high reliability in terms of network infrastructure and operation. Software-defined networking (SDN) offers promising solutions to meet these requirements. Although, SDN has promising solutions, it needs to be developed for DCNs and multi-controller placement problems. In this thesis study, for load balancing of SDN-based DCNs, several artificial intelligence techniques are used, and their performances are tested. The performance of the proposed algorithms is compared in terms of accuracy and response time, and it is seen that deep learning approach which is used for the first time in the literature, is quite successful compared to other artificial intelligence techniques. In addition, in this thesis, the multiple controller placement problem (CPP) in SDN is examined, and the genetic algorithm, k-means, optimized k-means, artificial bee colony (ABC) algorithm are applied to the solution of the CPP problem and also a hybrid artificial bee colony algorithm is proposed. Considering the parameters such as the number of controllers, the end-to-end latency and the best position detection controller, the performance of the Hybrid-ABC algorithm give a better solution than the ABC in minimizing the overall latency, the between the controllers and the relevant switches latency and finding the best position of the controllers.

**Keywords:** Software-Defined Network, Big Data Center, Load Balancing, Controller Placement Problem, Deep Learning, Artificial Intelligence

# İÇİNDEKİLER

## YAZILIM TANIMLI AĞ TABANLI BÜYÜK VERİ AĞ MİMARİLERİ ÜZERİNDE YÜK DENGELEME VE KONTROLÖR YERLEŞTİRME PROBLEMİNİN YAPAY ZEKA YAKLAŞIMLARI İLE ÇÖZÜMÜ

BİLİMSEL ETİĞE UYGUNLUK ..... ii
YÖNERGEYE UYGUNLUK ..... ii
KABUL VE ONAY ..... iv
TEŞEKKÜR ..... v
ÖZET ..... vi
ABSTRACT ..... vii
İÇİNDEKİLER ..... viii
KISALTMALAR ..... x
TABLOLAR LİSTESİ ..... xi
ŞEKİLLER LİSTESİ ..... xii
GİRİŞ ..... 1

**1. BÖLÜM**

**GENEL BİLGİLER ve LİTERATÜR ÇALIŞMASI**

**1.1. Problem Durumu ..... 3**
**1.2. Araştırmanın Amacı ..... 7**
**1.3. Araştırmanın Önemi ..... 7**

**2. BÖLÜM**

**YÖNTEM VE MATERYAL**

**2.1. Yazılım Tanımlı Ağ (YTA) ..... 9**
**2.1.1. Uygulama Düzlemi ..... 10**
**2.1.2. Kontrol Düzlemi ..... 10**
**2.1.2.1. Kontrolörler ..... 12**
**2.1.3. Veri Katmanı ..... 14**
**2.1.4. Veri ve Kontrol Düzlemi Arasındaki İletişim: Güneye Bağlı API'ler ..... 14**
**2.1.5. Kontrol Programları ve Kontrolör Arasındaki İletişim: Kuzeye Bağlı API'ler ..... 15**
**2.1.6. Doğu- Batı Bağlı API'ler ..... 16**

**2.1.7. Simülasyon ve Emülasyon Platformları ..... 17**
**2.2. Büyük Veri ..... 17**
**2.3. Yük Dengeleme ..... 19**
**2.4. Kontrolör Yerleştirme Problemi (KYP) ..... 20**
**2.5. Yapay Zekâ Teknikleri ..... 20**
**2.5.1. Lojistik Regresyon (LR) ..... 20**
**2.5.2. Destek Vektör Makinesi (DVM) ..... 21**
**2.5.3. Yapay Sinir Ağları (YSA) ..... 21**
**2.5.4. Derin Öğrenme (DÖ) ..... 22**
**2.5.5. Genetik Algoritma ..... 23**
**2.5.6. K-means ve Optimize Edilmiş K-Means Algoritması ..... 24**
**2.5.7. Yapay Arı Koloni Algoritması ve Hibrit Yapay Arı Koloni Algoritması ... 24**
**2.6. Yöntem ..... 27**
**2.6.1. YTA tabanlı BVM'lerde Yük Dengeleme ..... 27**
**2.6.2. YTA'da Kontrolör Yerleştirme Problemine Çözüm Üretilmesi ..... 31**
**2.6.2.1. Genetik Algoritma Yaklaşımı ..... 31**
**2.6.2.2. K-means ve Optimize Edilmiş K-means Algoritması Yaklaşımı ..... 31**
**2.6.2.3. YAK ve Hibrit-YAK Algoritması Yaklaşımı ..... 33**

**3. BÖLÜM**
**BULGULAR**

**3.1. YTA Tabanlı BVM'lerde Yük Dengeleme ..... 34**
**3.2. YTA'da Kontrolör Yerleştirme Problemine Çözüm Üretilmesi ..... 43**
**3.2.1. Genetik Algoritma Yaklaşımı ..... 43**
**3.2.2. K-means ve Optimize Edilmiş K-means Algoritması Yaklaşımı ..... 47**
**3.2.3. YAK ve Hibrit-YAK Algoritması Yaklaşımı ..... 54**

**4. BÖLÜM**
**SONUÇ ve GELECEK ÇALIŞMALAR**

**4.1. Sonuç ..... 59**
**4.2. Gelecek Çalışmalar ..... 60**
**KAYNAKÇA ..... 61**
**ÖZGEÇMİŞ ..... 69**

## KISALTMALAR

| | | |
|---|---|---|
| ABC | : | Artificial Bee Colony |
| BVM | : | Büyük Veri Merkezi |
| BGP | : | Sınır Geçiş Protokolü |
| CPP | : | Controller Placement Problem |
| DA | : | Dijkstra Algoritması |
| DCN | : | Data Center Network |
| DÖ | : | Derin Öğrenme |
| DVM | : | Destek Vektör Makinesi |
| GA | : | Genetik Algoritma |
| Hibrit-YAK | : | Hibrit- Yapay Arı Koloni |
| KTH | : | Kareler Toplam Hatası |
| KKO | : | Karınca Kolonisi Optimizasyonu |
| KYP | : | Kontrolör Yerleştirme Problemi |
| LR | : | Lojistik Regresyon |
| SDN | : | Software Defined Networking |
| TCP | : | Transmission Control Protocol |
| TLS | : | Transport Layer Security |
| YAK | : | Yapay Arı Koloni |
| YDY | : | Yük Dengeleme Yönlendiricisi |
| YSA | : | Yapay Sinir Ağları |
| YTA | : | Yazılım Tanımlı Ağ |

# TABLOLAR LİSTESİ

Tablo 2.1. Sıklıkla Kullanılan Kontrolörler ve Özellikleri ..... 13
Tablo 3.1. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi (ms) ..... 36
Tablo 3.2. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryoda LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi (ms) ..... 37
Tablo 3.3. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Ortalama Yanıt Süresi (ms) ..... 38
Tablo 3.4. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Ortalama Yanıt Süresi (ms) ..... 38
Tablo 3.5. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Doğruluk Oranları ..... 39
Tablo 3.6. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Doğruluk Oranları ..... 39
Tablo 3.7. KYP Problemine YAK ve Hibrit-YAK Algoritması Yaklaşımında Algoritmaların Performansının Karşılaştırılması ..... 57

# ŞEKİLLER LİSTESİ

Şekil 2.1. YTA mimarisi ..... 10
Şekil 2.2. Klasik bir k-ary fat tree topolojisi ..... 18
Şekil 2.3. Bir Yapay Sinir Ağı Diyagramı ..... 21
Şekil 2.4. Genetik Algoritma Akış Diyagramı ..... 23
Şekil 2.5. Yapay Arı Koloni Algoritması Akış Diyagramı ..... 26
Şekil 2.6. YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Oluşturulan Sistemin Şeması ..... 27
Şekil 2.7. YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Ağ Uygulamalarında Kullanmak Üzere Geliştirilen Yapay Zekâ Teknikleri İş Akışı ..... 28
Şekil 2.8. YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Oluşturulan Sistemin Akış Diyagramı ..... 29
Şekil 2.9. YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Oluşturulan Sistemin Blok Diyagramı ..... 30
Şekil 3.1. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Kullanılan YSA Modeli ..... 35
Şekil 3.2. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Kullanılan DÖ Modeli ..... 35
Şekil 3.3. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi Grafiği ..... 37
Şekil 3.4. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için LR, DVM (C=1 ve C=100), YSA ve DÖ Algoritmalarının Yanıt Süresi Grafiği ..... 38
Şekil 3.5. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için DÖ ROC Eğrisi ..... 39
Şekil 3.6. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için YSA ROC Eğrisi ..... 40
Şekil 3.7. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için DVM C=1 ROC Eğrisi ..... 40
Şekil 3.8. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için DVM C=100 ROC Eğrisi ..... 41

Şekil 3.9. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için DÖ ROC Eğrisi ..... 41
Şekil 3.10. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için YSA ROC Eğrisi ..... 42
Şekil 3.11. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için DVM C=1 ROC Eğrisi ..... 42
Şekil 3.12. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için DVM C=100 ROC Eğrisi ..... 43
Şekil 3.13. KYP Problemine Genetik Algoritma Yaklaşımının ULAKNET Veri Tabanı için Yakınsama Grafiği ..... 44
Şekil 3.14. KYP Problemine Genetik Algoritma Yaklaşımının Colt Veri Tabanı için Yakınsama Grafiği ..... 44
Şekil 3.15. KYP Problemine Genetik Algoritma Yaklaşımında ULAKNET Veri Tabanı için Denetleyici Yerleşim Yerlerinin Sayısal Değerleri ..... 45
Şekil 3.16. KYP Problemine Genetik Algoritma Yaklaşımında ULAKNET Veri Tabanı için Denetleyici Yerleşim Yerlerinin Grafiksel Gösterimi ..... 465
Şekil 3.17. KYP Problemine Genetik Algoritma Yaklaşımında Colt Veri Tabanı için Denetleyici Yerleşim Yerlerinin Sayısal Değerleri ..... 46
Şekil 3.18. KYP Problemine Genetik Algoritma Yaklaşımında Colt Veri Tabanı için Denetleyici Yerleşim Yerlerinin Grafiksel Gösterimi ..... 46
Şekil 3.19. KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=2) ..... 47
Şekil 3.20. KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=3) ..... 48
Şekil 3.21. KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=4) ..... 48
Şekil 3.22. KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=5) ..... 49
Şekil 3.23. KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=6) ..... 49
Şekil 3.24. KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=7) ..... 50

Şekil 3.25. KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=1 ms) ..... 50
Şekil 3.26. KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=2 ms) ..... 51
Şekil 3.27. KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=3 ms) ..... 51
Şekil 3.28. KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=4 ms) ..... 52
Şekil 3.29. KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=5 ms) ..... 52
Şekil 3.30. KYP Problemine K-means Yaklaşımında Kontrolör Sayısı ile Maksimum Gecikme Arasındaki İlişki ..... 53
Şekil 3.31. KYP Problemine Optimize Edilmiş K-means Yaklaşımında Kontrolör Sayısı ile Maksimum Gecikme Arasındaki İlişki ..... 54
Şekil 3.32. KYP Problemine YAK Algoritması Yaklaşımında Denetleyici Yerleşim Yerleri ..... 55
Şekil 3.33. KYP Problemine Hibrit-YAK Algoritması Yaklaşımında Denetleyici Yerleşim Yerleri ..... 55
Şekil 3.34. KYP Problemine YAK Algoritması Yaklaşımında Ağırlıklandırılmış Denetleyici Yerleşim Yerleri ..... 56
Şekil 3.35. KYP Problemine Hibrit-YAK Algoritması Yaklaşımında Ağırlıklandırılmış Denetleyici Yerleşim Yerleri ..... 56

# GİRİŞ

İnsan gücünün artması kuşkusuz 21. Yüzyılın en önemli teknolojik başarıları arasında yer almaktadır. Çünkü, verileri daha yüksek oranlarda işleyebilmek yeni alanların keşfedilmesine sebep olmuştur. Özellikle büyük veri merkezi (BVM) yapıları ve yazılım tanımlı ağ (YTA) oluşturma bu başarılara doğrudan katkıda bulunan yöntem ve teknolojiler arasındadır. BVM'ler devasa işlem gücüne sahiptir ve geleneksel yöntemlerle asla erişilemeyen bilgileri çıkarabilmektedir. YTA'lar ise BVM'lerin yönetilmelerine yardımcı olmaktadır. Özellikle ağ anahtarlarını kontrol etmek ve yeni ağ protokollerini dağıtmak için YTA'lar kullanılmaktadır. BVM'ler ise YTA kontrolörlerinin toplanan ağ bilgilerini daha iyi analiz etmesine ve kaynakların farklı ağ akışlarına tahsisi konusunda verimli kararlar almasına da yardımcı olabilmektedir.

BVM'lerdeki ana zorluk sunucular arasındaki yükü dengelemektir. Bu zorluk için olası bir çözüm olarak ağ taleplerini karşılamak amacıyla yapay zekâ teknikleri kullanılmıştır. YTA destekli büyük veri akışına sahip ağlar tek bir kontrolöre, ağ kesintilerine ve veri kaybına neden olabileceğinden BVM'lerde çoklu kontrolörlü YTA ağı oluşturmak ve yönetmek BVM'ler için faydalı olmaktadır. Ancak çoklu kontrolörlü YTA yaklaşımında kullanılması gereken kontrolör sayısının belirlenmesi ve bu kontrolörlerin yerleşiminin uçtan uca gecikme, genel gecikme ve kontrolörler arasındaki gecikmeyi optimum bir şekilde belirlemek oldukça problemlidir.

Bu tez çalışmasında BVM ağlarının daha iyi yönetilebilmesi için, YTA tabanlı BVM'lerde derin öğrenme (DÖ), yapay sinir ağları (YSA), lojistik regresyon (LR) ve destek vektör makineleri (DVM) teknikleri ile yük dengeleme çalışmasının yapıldığı ve kontrolör yerleştirme problemin de genetik algoritma (GA), k-means, optimize edilmiş k-means, yapay arı koloni algoritması (YAK) ve hibrit yapay arı koloni (Hibrit-YAK) algoritmaları ile algoritmaların kontrolör sayısını, uçtan uca gecikme, genel gecikme ve kontrolör- anahtar gecikmesini değerlendirdiği bir çalışma gerçekleştirilmiştir. BVM'de

yükü dengelemek için DÖ, LR, DVM ve YSA algoritmaları ilk kez kullanılmıştır. Kullanılan algoritmalar cevap süresi ve doğruluk oranları bakımından karşılaştırılmıştır. Yük dengeleme için derin öğrenme algoritmasının daha kararlı ve başarılı sonuçlar ürettiği görülmüştür. Kontrolör yerleştirme problemi için, GA, k-means ve optimize edilmiş k-means ve YAK ve Hibrit-YAK algoritması yaklaşımları önerilmiştir. Önerilen algoritmalar, kontrolör yerlerinin belirlenmesi için uçtan uca gecikmeyi ve kontrolörler arası gecikmeyi azaltmak ve ayrıca kontrolör sayısının belirlenmesi gibi farklı unsurlar için değerlendirilmiştir. Bu çalışmalarla, kontrolör yerleştirme problemi için ilk kez optimize dilmiş k-means algoritması, YAK algoritması ve Hibrit-YAK algoritması kullanılmıştır.

Bu tez çalışması dört bölümden oluşmaktadır. Çalışma şu şekilde organize edilmiştir:

1. Bölümde, YTA ve BVM kavramlarının yanı sıra mevcut en son gelişmeler ve mimariler hakkında bilgiler verilmiştir. Ayrıca problemin durumu ile bu araştırmanın amacı ve önemi de bu bölümde ayrıntılı olarak açıklanmıştır.

2. Bölümde, çalışmada kullanılan YTA, BVM, yük dengeleme, kontrolör yerleştirme problemi gibi terimler ve yapay zekâ teknikleri açıklanmıştır. Ayrıca, YTA tabanlı BVM'lerde yük dengeleme çalışmasının yöntemi ve kontrolör yerleştirme probleminde kullanılan farklı algoritmaların çalışma yöntemleri sunulmuştur.

3. Bölümde, YTA tabanlı BVM'lerde yük dengeleme çalışmasında elde edilen ve kontrolör yerleştirme probleminde kullanılan farklı algoritmalardan elde edilen bulgular ve bu bulguların değerlendirildiği tartışma bölümü verilmiştir.

4. Bölümde tez çalışması ile ilgili sonuçlar ve öneriler verilmiştir.

# 1. BÖLÜM

# GENEL BİLGİLER VE LİTERATÜR ÇALIŞMASI

## 1.1. Problem Durumu

Yeni iletişim teknolojilerinin gelişmesiyle birlikte, veri aktarım miktarı giderek artmıştır. Bu artan bilgi işlem kaynağı talebini etkin bir şekilde karşılamak için, iyi organize edilmiş anahtarlarla bağlantılı sunuculardan oluşan BVM yapıları ihtiyacı doğmuştur [1]. Ancak geleneksel ağ yapısı, ağ altyapısı ve operasyon bakımından BVM'nin büyük ölçek, geniş uygulama çeşitliliği, yüksek güç yoğunluğu ve yüksek güvenilirlik gibi gereksinimlerini karşılayamamaktadır. Bu gereksinimleri karşılamak için YTA umut vaat edici çözümler sunabilir çünkü yeni bir ağ mimarisi olarak ortaya çıkan YTA, kontrol düzlemini veri düzleminden ayırma ve ağ uygulama geliştirme için programlanabilirlik sağlama dahil olmak üzere iki ayırt edici özelliğe sahiptir. YTA, dinamik ağ optimizasyonu için küresel bir ağ görünümü ve optimizasyon algoritması uygulamasını kolaylaştırmak için mantıksal olarak merkezileştirilmiş kontrol sağlamaktadır. Ayrıca YTA, yenilikçi ağ tasarımlarına uyum sağlamak için daha verimli yapılandırma, daha iyi performans, daha yüksek esneklik sağlayacak şekilde konumlandırılmıştır ve ağ anahtarlarını kontrol ederek ve yeni ağ protokollerini dağıtarak BVM'lerin yönetilmesinde kullanılmaktadır [2]. YTA'nın gelişmiş özelliklerinin olmasına rağmen, gelişme ihtiyacı duyan ve problem oluşturan özellikleri de vardır. Araştırmacılar, özellikle güvenlik, yük dengeleme, servis kalitesi, kontrolörlerin yerleştirilmesi, ölçeklenebilirlik, esneklik ve tek hata noktası gibi problemlere çözüm bulmak için çalışmalar yapmaktadırlar. Literatürde, YTA tabanlı BVM üzerine çeşitli çalışmalar vardır [2-10]. Jain vd. [3]'nin yaptığı çalışmada, Google'ın tüm dünyadaki veri merkezlerini birbirine bağlayan ve özel bir tasarım olan B4'ün uygulaması ve değerlendirmesi sunulmuştur. B4 ile OpenFlow kullanan bir YTA mimarisi oluşturulmuştur. B4, birçok bağlantıyı uzun süreler boyunca %100'e yakın

kullanımda çalıştırarak, düşük maliyetli bant genişliği sağlamıştır, ancak yük dengeleme, servis kalitesi gibi önemi parametrelerin değerlendirilmesi konusunda eksiklikleri mevcuttur. Tavakoli vd. [4]'nin yaptığı çalışmada, gelişmiş kontrol düzlemlerini basit veri düzlemleriyle birleştiren YTA, yönetim sistemi olarak NOX önerilmiştir. Bu sistemin amacı, çok çeşitli ihtiyaçları karşılayacak kadar esnek ağ kontrolü sağlamaktır. Ancak, belirli ayarlara uygun ağ yönetimi yetenekleri, yeniden kullanılabilir yazılım modüllerinde somutlaştırılarak uygulanmadığı gibi YTA'nın ve BVM'nin özelliklerini iyileştirecek yeni teknikler de önerilmemiştir.

Yük dengeleme, iş yükünü birden çok kaynağa bölerek bir kaynağın aşırı yüklenmesini önleyen bir tekniktir. Ağdaki ölçeklenebilirliği en üst düzeye çıkarır. Geleneksel ağ yük dengeleme uygulaması, yazılım ve donanım uygulamasını içerir. Ancak, geleneksel ağlarda, yük dengeleme stratejisini küresel olarak uygulamak kolay olmadığı için tüm ağ durumunu elde etmek zordur. Ek olarak, geleneksel yük dengeleme yönteminin ağ durumundaki değişikliklere ve ayarlamalara uyum sağlaması da zordur ve bu cihazlar belirli donanımlar için tasarlandığı için maliyetlidir, esnek değildir ve satıcıya göre farklılık gösterir. YTA'da yük dengeleyiciler, ağ yükünü yönetmek için YTA denetleyicisinde program kodları olarak kolayca uygulanabilir ancak geliştirilmeye ihtiyacı vardır. Literatürde YTA'daki, trafik mühendisliği, yönlendirme, yük dengeleme gibi farklı problemler için yapay zekâ algoritmaları uygulanmıştır [6,11-44]. Yük dengeleme için yapılan çalışmalara bakılacak olursa, Wang vd. [11], karınca kolonisi optimizasyonu (KKO) tekniğine dayalı bir bağlantı yük dengeleme algoritması önermiştir. Önerilen algoritmada, yükü düğümden düğüme dengelemek için KKO'nun araştırma kuralı ve bağlantı yük dengeleme yöntemi kullanılmıştır. Yük dengeleme faktörleri için bağlantı yükü, gecikme ve paket kaybı oranı kullanılmıştır. Stefano vd. [12], YTA'da trafik mühendisliği uygulamalarında kullanılması için geliştirilmiş KKO ile yük dengeleme algoritması sunmuştur. Çalışmasının performansı, Dijkstra algoritmasının (DA) performansıyla karşılaştırılmış ve DA'ya göre %55 daha verimli olduğu gösterilmiştir. Bu çalışmanın başarımı, otonom dinamik yönlendirme ve ağ performansında yapılan iyileştirmelerde ve ağ bant genişliğinden daha iyi yararlanma konusunda karşılaştırılan algoritma performanslarından daha iyidir. Zhu vd. [13], tarafından yapılan çalışmada, trafik yükünü dengelemek ve servis kalitesini sağlamak için çok amaçlı bir parçacık sürüsü optimizasyon algoritması kullanılmıştır. Bu

algoritma ile güç tüketiminin azaltılması için en uygun yollar tespit edilmiştir. Böylece enerji verimliliği üzerinde yük dengeleme çalışması gerçekleştirilmiştir. Al-tam ve Correia'nın [14] çalışmasında, anahtar geçiş sorununu çözmek için sezgisel bir yaklaşım sunulmuştur ve farklı algoritma performansları ile karşılaştırıldığında yük dengeleme sonuçlarını %14'e kadar iyileştirdiği görülmüştür. Xue vd. [15], YTA performansının iyileştirilmesi için genetik algoritmayı, karınca koloni algoritması ile bütünleştiren yeni bir dinamik yük dengeleme şeması önermişlerdir. Önerilen şema ile, en uygun yol arama hızı, gidiş-dönüş süresi ve paket kayıp oranı açısından Round Robin ve KKO algoritmasına göre oldukça iyi sonuçlar elde edilmiştir. Prakash ve Deepalakshmi [16], mevcut kaynakların etkin bir şekilde kullanması amacıyla, geri yayılımlı yapay sinir ağı modelini YTA'da dinamik aracı tabanlı yük dengeleme çalışması için önermiştir. Bu çalışmada, yük dengeleme yaklaşımının genel ağ verimliliğini geliştirdiği ve veri geçişi üzerinde iyi çalıştığı gösterilmiştir. Todorov vd. [17] yaptığı çalışmada, makine öğrenimine dayalı segment yönlendirme algoritmasını ve yük dengeleme mekanizmalarını birleştirmiş ve ağ üzerindeki toplam bant genişliğini azaltmıştır. Bu çalışma ile YTA ağlarındaki yönlendirme mekanizmasının iyileştirmesi amaçlanmıştır. Fu vd. [18] tarafından yapılan çalışmada, YTA tabanlı veri merkezi ağları için derin Q-öğrenmeye dayalı yeni bir yönlendirme stratejisi, optimum yönlendirme yollarını otonom olarak oluşturmak için önerilmiştir. Önerilen yönlendirme şeması akışların ortalama gecikmesini ve ortalama paket kayıp oranını azaltmış, ortalama verimi artırmıştır. Lei vd. [19] tarafından yapılan çalışmada YTA kullanılarak, makine öğrenimine dayalı tıkanıklık denetimi ve yük dengeleme problemleri çoklu görev senaryolarında incelenmiştir. Deneysel sonuçlar yöntemin verimli olduğunu göstermiştir.

Kontrolör Yerleştirme Problemi (KYP), büyük ağ mimarileri için YTA'daki kontrolörlerin nasıl dağıtılması gerektiğine bağlı olarak ortaya çıkmış bir problemdir. KYP'de, denetleyicilerin sayısı, kontrolörlerin konumu ve anahtar- kontrolör arasındaki bağlantılar olmak üzere 3 temel problem vardır. Dağıtılan çoklu denetleyicilerin, çeşitli ağ uygulamalarını desteklemek için tipik olarak hızlı bir şekilde yeni akışlar oluşturması ve küresel ağ görünümlerini senkronize etmesi gerekir. Genel gecikme, denetleyicilerin sayısı ve kontrolörlerin konumu YTA'da oldukça önemlidir. Literatürde bu konularda yapılmış farklı çalışmalar mevcuttur [45-64]. Ahmadi ve Khorramizadeh [45], yaptığı

çalışmada büyük ölçekli çok amaçlı denetleyici yerleştirme problemini çözmek için evrimsel algoritmalardan yararlanmıştır. Kontrolörlerin kapasiteleri ve anahtar yükleri kısıtlamalar olarak probleme eklenmiştir. Önerilen algoritma, literatürdeki farklı algoritmalardan daha iyi performans göstermiştir. Huang vd. [46], tarafından yapılan çalışmada, iletişim gecikmesi, kontrol düzlemi kullanımı ve kontrolör iş yükü dağılımı aynı anda NP-hard problemi olarak ele alınmış, KYP için GA ve gradyant iniş algoritması ele alınarak çözüm üretilmiştir. Çalışmada GA, uygun KYP çözümlerini aramak için kullanılırken, her çözümün kalitesi gradyant iniş algoritması aracılığıyla değerlendirilmiştir. İki temsili ağ senaryosundaki (küçük ölçekli ve büyük ölçekli) simülasyon sonuçları, algoritmanın kontrol düzlemi kullanımı ile ağ yanıt süresi arasındaki dengeyi etkin bir şekilde sağlayabildiğini göstermiştir. Sanner vd. [47] yaptıkları çalışmada, KYP için, dinamik yerleşim senaryolarının yönetilmesini, ağların güvenilirliğini artıracak şekilde kontrolörlerin dağıtılmasını ve kontrolörler arasındaki yükün dengelenmesini amaçlamıştır. Bu çalışmanın, küçük ağlarda, iyi performans sergilediği görülmüştür. Liao ve Leung [48] çalışmasında, geniş alan YTA, KYP için anahtardan denetleyiciye gecikmeyi, denetleyiciden denetleyiciye gecikmeyi ve denetleyicideki yük dengesizliğini en aza indirmeyi amaçlayan, parçacık sürüsü optimizasyonu tabanlı mutasyon fonksiyonuna sahip çok amaçlı GA önerilmiştir. Önerilen algoritmanın verilen küresel en iyi konumlara doğru daha büyük bir çeşitlilikle çok daha kısa yakınsama süresinde ulaştığı görülmüştür. Sahoo vd. [49] çalışmasında, anahtardan belirlenmiş bir denetleyiciye geçiş gecikmesini optimize eden ve denetleyicilerin optimal konumlarını kısa sürede bulmayı amaçlayan parçacık sürüsü optimizasyonu ve Firefly tekniği önerilmiştir. Ve Firefly'ın optimale yakın hesaplama süresi ve maliyeti açısından daha iyi bir sonuç sağladığı gösterilmiştir. Wang vd. [50] çalışmasında, gecikmenin azaltılması için KYP problemine, optimize edilmiş k-means algoritması önerilmiştir. Önerilen algoritmanın, standart k-means algoritmasına kıyasla, merkez ve düğümleri arasındaki maksimum gecikmeyi önemli ölçüde azalttığı görülmüştür.

KYP problemi için yapılan bu [45-50] çalışmalarda farklı yapay zekâ teknikleri kullanılmış, konumların kısa sürede bulunması, yük dengelemesi, ağ gecikmesi gibi farklı konular ele alınmıştır. Ancak özellikle genel gecikmenin ve denetleyiciler ile ilgili anahtarlar arasındaki gecikmenin en aza indirilmesi ve kontrolörlerin konumunu en

optimum şekilde ağırlıklandırılmış olarak bulunması KYP için oldukça önemlidir ve bu çalışmalar bu konularda yetersiz kalmıştır.

### 1.2. Araştırmanın Amacı

Bu tez çalışmasında YTA tabanlı BVM modeli oluşturmak ve bu model üzerinde yük dengelemesinin gerçekleştirilmesi amaçlanmıştır. Diğer amacı ise, YTA kontrolör yerleştirme probleminin çözümüne çeşitli optimizasyon algoritmaları uygulayıp, başarımını incelemektir. Bu amaçlar doğrultusunda,

- BVM'deki yüksek veri yoğunluğundan kaynaklanabilecek karmaşıklıkların azaltılması için YTA tabanlı yük dengeleme tekniğinin derin öğrenme ve çeşitli yapay zekâ algoritmaları ile başarımının test edilmesi
- YTA'da kontrolörlerin yerleştirilmesine yönelik oluşan probleme;
  - Genetik algoritma yaklaşımı
  - K-means ve optimize edilmiş k-means yaklaşımı
  - Yapay Arı Koloni (YAK) ve Hibrit-YAK algoritması yaklaşımı önerilerek başarımlarının test edilmesi hedeflenmiştir.

### 1.3. Araştırmanın Önemi

Yapılan çalışmalar incelendiğinde, BVM için YTA'da yük dengeleme çalışmalarının önemli geliştirmelere ihtiyacı olduğu görülmektedir. Yük dengeleme, yol bilgilerinin anlık olarak değerlendirilmesini ve dinamik olarak işlenmesini gerektiren önemli bir tekniktir. Yapılan çalışmalarda, anlık verilerle değil toplanan verilerle statik yük dengelemesi yapıldığı görülmüştür.

Ayrıca YTA'larda, KYP problemi için de konumların kısa sürede bulunması, yük dengelemesi, ağ gecikmesi gibi farklı konular ele alınmıştır. Ancak özellikle genel gecikmenin ve denetleyiciler ile ilgili anahtarlar arasındaki gecikmenin en aza indirilmesi ve kontrolörlerin konumunu en optimum şekilde ağırlıklandırılmış olarak

bulunması KYP için oldukça önemlidir ve bu çalışmalar bu konularda yetersiz kalmıştır.

Bu tez çalışmasında ilk olarak, BVM tabanlı YTA'da dinamik yük dengeleme için DÖ tekniği önerilmiştir Babayigit ve Ulu [65]. DÖ ağını eğitmek için bağlantılar arasındaki değişken yük değerleri kullanılmıştır. DÖ tekniğinin yük dengeleme için tepki süresi, YSA, DVM ve LR gibi farklı yapay zekâ teknikleriyle karşılaştırılmıştır. Deneysel sonuçlar, YSA ve DÖ'nün yanıt süresi sonuçlarının DVM ve LR algoritmalarından daha düşük olduğunu ortaya koymuştur. Ayrıca, DÖ'nün doğruluğunun, YSA doğruluğundan daha yüksek olduğu bilgisine ulaşılmıştır. Sonuç olarak, DÖ'nün, YTA tabanlı BVM'lerde dinamik olarak yük dengelemesi için çok verimli olduğu görülmüştür.

Daha sonra YTA'da, KYP'ye çözüm üretmek amacıyla GA, k-means, optimize edilmiş k-means, YAK ve Hibrit-YAK yaklaşımları önerilmiştir. Önce GA'nın her bir çalışma zamanında en iyi çözümü bulma özelliğinden yararlanılarak uçtan uca gecikmeyi en aza indirdiği bir sistem tasarlanmış, ULAKNET ve Colt veri setlerine uygulanmış ve elde edilen optimum çözümler grafla oluşturulan harita üzerinde gösterilmiştir. Ardından, k-means ve optimize edilmiş k-means algoritmaları, kontrolör sayısının belirlenmesi ve uçtan uca gecikmenin en aza indirilmesi için kullanılmış ve elde edilen sonuçların oldukça verimli olduğu görülmüştür. En son çalışma olarak da YAK ve Hibrit-YAK algoritmaları KYP problemi için kullanılmış, algoritmaların optimal çözüme yakınsamayı önemli ölçüde iyileştirdiği, çözüm kalitesini arttırdığı ve çalışma süresini azalttığı sonucuna varılmıştır.

Bu tez çalışmasıyla elde edilen verilerin gelecek YTA tabanlı BVM çalışmalarında ve YTA'da KYP problemin çözümü çalışmalarında araştırmacılara önemli yol göstereceği düşünülmektedir. Ayrıca YTA tabanlı BVM'de yük dengeleme problemi için ve KYP'de kontrolörlerin sayısının ve yerlerinin optimum bulunması probleminde farklı bakış açılarının ilk kez uygulanmasından dolayı bu problemlerin çözümüne katkı sağlayacağı ve böylece gelecekteki çalışmalarda önemli ilerlemeler sağlanacağı öngörülmektedir.

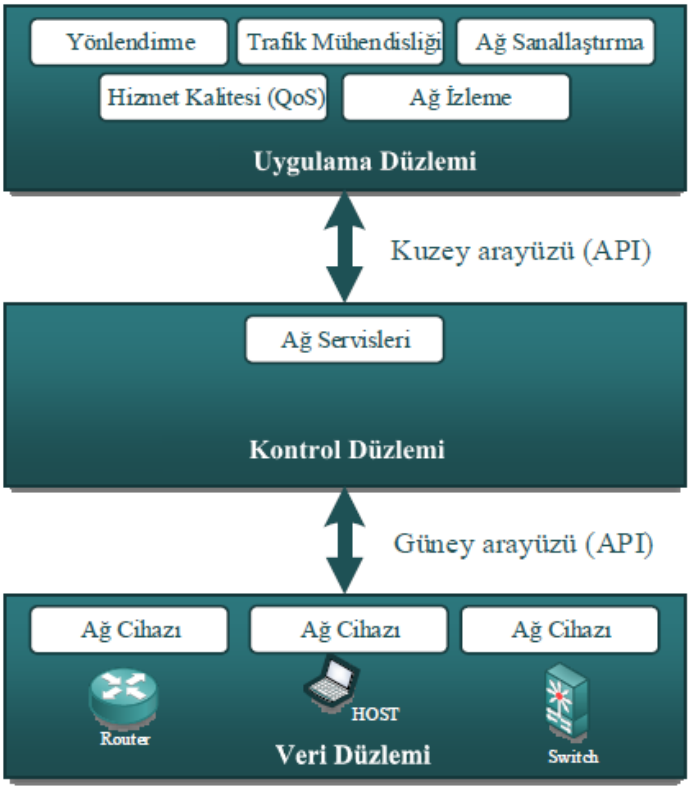
# 2. BÖLÜM

# YÖNTEM VE MATERYAL

Bu bölümde ilk olarak, öncelikle tez çalışmasında kullanılan YTA, büyük veri, KYP ile yapay zekâ tekniklerinden LR, DVM, YSA, DÖ, GA, k-means, optimize edilmiş k-means, YAK ve Hibrit-YAK kavramları genel hatlarıyla açıklanmıştır. Ayrıca, BVM için YTA'larda yük dengeleme çalışması ve YTA'da kontrolör yerleştirme problemine üretilen çözümler anlatılmıştır.

## 2.1. Yazılım Tanımlı Ağ (YTA)

YTA terimi ilk olarak 2009'da OpenFlow adlı bir standart geliştirmede yapılan çalışmaları tanımlayan bir makale [66]'de ortaya çıkmıştır. OpenFlow protokolü, ağ düzenini ve trafik akışını gerçek zamanlı olarak değiştirmek için ağ mühendislerine harici bilgisayarlardan, anahtarlardaki ve yönlendiricilerdeki akış tablolarına erişim sağlamaktadır. Böylece OpenFlow, YTA iletişim ağlarında [67-69] bir evrime neden olmuştur. Bu gelişme, bir kontrolörün, bir ağı açık arabirimler aracılığıyla yönettiği ve çalıştırdığı ağ denetimine daha merkezi bir yaklaşım getirmiştir. YTA, kontrol düzleminin veri düzleminden ayrıldığı ve ağ kontrolünün doğrudan programlanabilir olduğu bir ağ mimarisine sahiptir. Bu mimari konsept ilk olarak Nicira Networks tarafından UCB, Stanford, CMU, Princeton'daki [70] daha önceki gelişmelere dayanılarak önerilmiştir. YTA, açık ara yüz yazılımı aracılığıyla ağın davranışını dinamik olarak kontrol edebilmekte ve yönetebilmektedir. Bu mimari doğrudan programlanabilir, çevik, merkezi olarak yönetilebilen ve açık standartlara dayalı satıcıdan bağımsız bir yapı oluşturmaktadır. YTA mimarisi temel olarak anahtarlar, akış girişleri ve denetleyiciler olmak üzere üç önemli bileşeni ve uygulama düzlemi, kontrol düzlemi ve veri düzlemi olmak üzere üç ana bölümü içermektedir. Şekil 2.1'de YTA mimarisi verilmiştir.




*Şekil 2.1.* YTA mimarisi

### 2.1.1. Uygulama Düzlemi

YTA'nın en üst düzlemi uygulama düzlemidir. Uygulama düzlemi, üst düzey bir dilde yazılmış çok çeşitli uygulamalardan oluşur. Bu uygulamalar, veri düzleminin kontrolünü sağlayan kontrolör(ler) ile iletişim kurar. Uygulamalar, denetleyiciden girdi olarak topolojik bir görünüm ve ağ istatistikleri alır. Uygulamalar, bu girdileri kullanarak yönlendirme, yük dengeleme, güvenlik, ağ sanallaştırma, mobilite yönetimi ve trafik mühendisliği gibi çok çeşitli kullanımlar için programlanabilir.

### 2.1.2. Kontrol Düzlemi

YTA'nın kontrol düzlemi, ağ işletim sistemi adı verilen bir kontrolöre sahiptir. Paketi ağın topolojik görünümüne göre işleyen kontrolör, ağdaki tüm anahtarlara bağlanabilir. Kontrolörler ağ yöneticisine, ağ protokollerinin dağıtılmasını, uygulamaya özel iletim kurallarını ve güvenlik duvarlarının oluşturulmasını farklı API'ler aracılığıyla sağlar.

Bu API'leri kullanan bir kontrolör, ağ yöneticisinin ağı manuel olarak yapılandırmak yerine ağ uygulamasını ihtiyacına göre programlamasını sağlar. Kontrolör, mantıksal olarak merkezileştirilmiş bir aygıttır ve aynı denetleyici uygulamasının bir örneğini çalıştıran farklı sistemler arasında fiziksel olarak dağıtılabilir. Dolayısıyla YTA, merkezi, dağıtılmış ve hibrit model (hem merkezi hem de dağıtılmış) denetleyici modellerini destekler. Her üç modelin de dikkate alınması gereken farklı altyapı unsurları ve gereksinimleri vardır.

Merkezileştirilmiş YTA mimarisinde, tek bir merkezi kontrolör tüm ağı denetler ve yönetir. Ağ bilgisi tek bir karar noktasında merkezileştirilir. Tüm ağları programlamak için tek bir merkezi kontrolör kullanıldığından, yönlendirme yolu boyunca her bir anahtar üzerindeki yüklere ilişkin genel bir bilgiye sahip olunması gerekir. Ayrıca, hangi yönlendiricinin hangi bağlantıda veya hangi akışta darboğaz oluşturduğunun takip edilmesi gerekmektedir. Kontrolör, ağ cihazlarından ağ istatistiklerini toplamak için anahtarlarla iletişim kurar ve bu verileri kontrol düzlemine gönderir. Merkezi kontrol düzlemi, tek bir yönetim noktası ile ağ üzerinde daha iyi bir kontrol avantajına sahip olmasına rağmen, bazı sınırlamaları vardır. Bu sınırlamalardan ilki, denetleyicinin OpenFlow anahtarlarını geleneksel anahtarlardan daha sık güncellemesi gerektiğidir. Tüm bağlantı noktaları doğrusal olarak taranır ve topolojinin keşfedilmesi, yanıt süresini önemli ölçüde artırır. Artan yanıt süresi, büyük ölçekli ağların aşırı yüklenmesine neden olabilir. Merkezileştirilmiş modeldeki ikinci sınırlama, sistemdeki her yeni akışın ilk paketinin kontrol için ilk önce kontrolöre iletilmesidir. Kontrolör, düğümden düğüme iletim akışı için gelecekteki yolu belirtir. Bu nedenle, yeni bir akış programlanacaksa, büyük ağlar için ölçeklenebilirlik sorunu olan kontrolörün yol üzerindeki tüm anahtarlarla iletişim kurması gerekir. Merkezileştirilmiş modeldeki üçüncü sınırlama olan merkezi kontrolör, ağı saldırılara ve izinsiz girişlere karşı oldukça savunmasız kılan tek bir arıza noktasını temsil eder. Kontrol eylemlerini tam olarak koordine etmek ve dağıtılmış işlevler arasında ağ durumlarının tutarlılığını korumak oldukça zordur.

Dağıtılmış YTA modeli, tek hata noktasının ortadan kaldırılmasına ve yükü dağıtılmış kontrolörler arasında paylaşarak ölçeğin büyütülmesine odaklanır. Dağıtılmış YTA katmanı, veri merkezlerinde yerel ağ işlemlerini gerçekleştirmeye daha duyarlı olacak şekilde tasarlanmıştır. Bu YTA modeli, çok çeşitli ağ teknolojilerine sahip geniş alanlı

YTA'lar için ağ gereksinimlerine kolayca uyum sağlayabilir. Ayrıca, dağıtılmış bir denetleyici küresel olayları işlerken daha hızlı ve verimli tepki verebilir. Tüm bunlara rağmen, ağ ölçeklenebilirliğini ve sağlamlığını geliştirmek için dağıtılmış YTA mimarisinin karşılaştığı birkaç temel zorluk vardır. İlk zorluk, dağıtılmış YTA mimarisinin tüm kontrolörler arasında tutarlı bir global görünüm gerektirmesidir. Kontrol düzlemleri ile yönlendirme düzlemleri arasındaki iletişim, halihazırda kullanılan statik konfigürasyonun yerine programlanmalıdır. Bu yapılandırma, yüklerin kontrolörler arasında eşit olarak dağıtılmasını önleyebilir. İkinci zorluk, ağın doğrusal olarak büyümesini sağlayan dağıtılmış kontrol ağı yapısındaki optimal kontrolör sayısını belirlemektir. Üçüncü zorluk, ağın küresel bir görünümünü elde etmek için yerel ve dağıtılmış olayları senkronize etmektir.

Dağıtılmış ve merkezi kontrolör modellerinin her birinde sınırlamaların üstesinden gelmek için hibrit YTA mimarileri önerilir. Hibrit YTA modeli, dağıtılmış modelin ölçeklenebilirliğinden ve esnekliğinden ve merkezi modelin basit kontrol özelliğinden yararlanır. Bu model, dağıtılmış protokollerde durumun ne olduğu, anahtarlar için hangi durumun yerel olması gerektiği ve denetleyiciler arasındaki iletişimi koordine etmek için hangi durumun merkezileştirilmesi gerektiği gibi çeşitli bileşenler gerektirir. Ayrıca model, genel sistemi değiştirmek zorunda kalmadan mevcut altyapının yükseltilmesine izin verir ve güvenlik sorunlarının çözülmesi, ağ optimizasyonunun etkinleştirilmesi ve kontrol düzleminin aşırı yüklenmesi durumunda durum senkronizasyonunun sağlanması için yönetim ilkeleri sağlayabilir. Ancak ağ soyutlama modüllerinin ne kadarının merkezileştirilebileceğinin belirlenmesi bu açıdan kritik bir problemdir. Ayrıca, mantıksal olarak merkezi kontrol görevlerini desteklerken aynı zamanda fiziksel olarak dağıtılmış protokoller sağlamak gibi zorluklar da vardır.

#### 2.1.2.1.Kontrolörler

Ağ zekasını merkezileştiren YTA kontrolörü, gelişmiş hizmetler ve uygulamalar için gereken ağ politikalarını yapılandırır ve bunları ayrı ağ cihazlarına uygular [67, 71]. Ağ anahtarlarında gerçekleştirilen akış girişlerini ekleme, kaldırma ve değiştirme gibi belirtilen talimatlar, güneye bağlı bir iletişim arabirimi kullanılarak anahtarlarla gerçekleştirilir. Anahtarlar, standart bir TLS veya TCP bağlantısı kullanarak önceden tanımlanmış bir bağlantı noktası numarası aracılığıyla denetleyiciyle iletişim kurar.

Kontrolör ve anahtarlar arasındaki iletişimi sağlayan kanal trafik kanalından bağımsızdır. Uygulamalar ve hizmetler için oluşturulan komutlar, kuzeye giden arabirim aracılığıyla kontrolöre iletilir ve kontrolör, bu komutları anahtarlarda düşük seviyeli komutlara dönüştürür. Ağ topolojisine, durumlara veya gereksinimlere bağlı olarak, anahtarlar bir veya daha fazla kontrolörle iletişim kurabilir. Genel olarak, kontrolörler arası iletişim, TCP kanalları üzerinden BGP veya Oturum Başlatma Protokolü SIP gibi harici protokoller tarafından gerçekleştirilir. Birden fazla kontrolör kullanmak, sistemin güvenilirliğini artırmak için önemli olabilir. Örneğin, bir kontrolör veya kontrol kanalı ile ilgili bir sorun varsa, anahtar başka bir kontrolörden akış yönlendirme talimatları alabilir. Tablo 2.1'de sık kullanılan kontrolörler, kontrolörlerin yazıldığı diller, destekledikleri OpenFlow sürümleri gibi özellikler verilmiştir.

Tablo 2.1. Sıklıkla Kullanılan Kontrolörler ve Özellikleri

| Kontrolör | Dili | OpenFlow Versiyonu | Rest - API | Çoklu Uygulama Desteği |
|---|---|---|---|---|
| OpenDayLight | Java/Python | 1.3 | Evet | Evet |
| ONOS | Java | 1.3 | - | Evet |
| OPNFV | Java/Python | 1.3 | Evet | Evet |
| DIFANE | C | 1.0 | - | Evet |
| HyperFlow | C++ | 1.0 | - | - |
| NOX | C++ | 1.0 | - | Hayır |
| SNAC | C++ | 1.0 | - | Hayır |
| POX | Python | 1.0 | - | Hayır |
| Pyretic | Python | 1.0 | - | Hayır |
| Ryu | Python | 1.5 | - | Hayır |
| Ryuretic | Python | 1.5 | - | Hayır |
| Beacon | Java | 1.0 | - | Evet |
| Floodlight | Java | 1.4 | Evet | Evet |
| Kandoo | Go | 1.0 | - | Evet |

### 2.1.3. Veri Katmanı

YTA altyapısını destekleyen özel ağ cihazları veri düzlemini oluşturur. Ağ cihazları ve YTA kontrolörü, güneye bağlı API'yi kullanarak güvenli bir kanal aracılığıyla iletişim kurar. YTA'daki ağ cihazları kendi başlarına veri iletemediklerinden, geleneksel ağ cihazlarına kıyasla aptaldırlar. YTA kontrolörü, ağ üzerinden haberleşme sırasında gerekli aksiyonların alınması için bu cihazlara komutlar gönderir. Kontrolör, bir dizi eylem ve farklı eşleşme alanlarından oluşan akış girişleri ekler. Gelen bir paket, TCP/IP protokolünün farklı katmanları için başlık değerlerine göre bu akış girişleriyle eşleştirilir. Paketin başlık değeri bir akış girişiyle eşleşirse, ağ cihazı paketi yapılacak eyleme yönlendirir. Paket, anahtarda bulunan akış girişleriyle eşleşmezse, paket güvenli bir kanal aracılığıyla kontrolöre iletilir. Akış giriş(ler)i, kontrolör tarafından anahtara eklenir veya silinir. Ardından, kontrol mesajları anahtar ve kontrolör tarafından paylaşılır. Bu akışlar, kontrolörün tüm topoloji hakkında bilgi edinmesini ve ayrıca anahtar tarafından işlenen paket sayısı, anahtarların bağlantı durumu ve anahtarların meşgul portlarının sayısı gibi bilgileri kaydetmesini sağlar.

YTA altyapısını tam olarak anlamak için YTA'daki düzlemler arasındaki protokolleri anlamak da oldukça önemlidir. YTA'da veri düzlemi ile kontrol düzlemi arasındaki iletişim güneye bağlı, kontrol programları ve kontrol düzlemi arasındaki iletişim kuzeye bağlı ve YTA ağ yapısı içerisinde kontrolör-kontrolör, sunucu-sunucu veya kontrolör-sunucu arasındaki iletişim doğu-batı protokolleri üzerinden yürütülmektedir.

### 2.1.4. Veri ve Kontrol Düzlemi Arasındaki İletişim: Güneye Bağlı API'ler

YTA altyapısını tam olarak anlamak için YTA veri düzlemi ile kontrol düzlemi arasındaki protokolleri incelemek önemlidir. YTA denetleyicisinin gerçek zamanlı gereksinimine göre ağ denetimi ve dinamik olarak ağ değişiklikleri sağlayan bu protokollere güneye bağlı API adı verilir. Bu protokollerden bazıları listelenmiştir.

- OpenFlow: Güneye bağlı API'de en çok kullanılan protokollerden biri olan OpenFlow, 2009 yılında Open Network Foundation tarafından oluşturulmuştur. Veri düzlemi cihazlarını programlamak için oluşturulan OpenFlow protokolü, OpenFlow özellikli anahtarlardan, OpenFlow denetleyicisinden ve anahtarlar ile denetleyici arasında güvenli bir kanal olan OpenFlow kanalından oluşur. Bu

protokol, kontrol düzlemindeki OpenFlow denetleyicisinin veri düzlemindeki OpenFlow uyumlu anahtarlarla iletişim kurmasını sağlar. OpenFlow özellikli bir anahtar, kuralları, istatistikleri ve eylemleri olan birden çok akış girişine ve her akış girişiyle ilişkili bir eyleme sahip bir akış tablosundan oluşur.

- OpFlex: CISCO tarafından oluşturulmuştur. OpenFlow'dan farklı olarak OpFlex, bazı akıllı özelliklerin ağ cihazlarına aktarıldığı bir ağ modelini destekler. OpFlex protokolü, denetleyicinin ağ ilkelerini ve uygulama gereksinimlerini anahtarlara göndermesini ve anahtarları buna göre yapılandırmasını sağlar. Bu nedenle, öncelikle politikaların uygulanmasına ve tanımlanmasına odaklanır.

- Network Konfigürasyon Protokolü: Ağ cihazlarını yönetme tekniklerini tanımlar, ağ cihazından gerekli bilgileri alır ve ağ cihazının konfigürasyonunu düzenler.

- Sınır Geçiş Protokolü (BGP): Geleneksel ağlarda kullanılan protokol, çeşitli otonom sistemlerin ağ geçidi yönlendiricileri arasında iletişimini sağlar. BGP, güneye bağlı bir API olarak da kullanılabilir. OpenDaylight denetleyicisi bir BGP uygulaması sağlar.

### 2.1.5. Kontrol Programları ve Kontrolör Arasındaki İletişim: Kuzeye Bağlı API'ler

Kuzeye bağlı iletişim protokolleri, YTA uygulamalarının mimarisi ve API'leri farklı satıcılar arasında farklılık göstermiştir. Bu nedenle Open Network Foundation, kuzeye bağlı arayüz standardizasyonunu sağlamak için Haziran 2013'te özel bir çalışma grubu oluşturmuştur. Kuzeye bağlı API'ler, farklı araştırma alanlarındaki geliştiricilerin farklı alanlarda ağ uygulamaları geliştirmesine olanak tanır. Ağ operatörlerine ağ denetimlerini hızlı bir şekilde gerçekleştirme ve ağı özelleştirme yeteneği sağlar. Aşağıdaki alt bölümler, YTA denetleyicilerinde kullanılan kuzeye bağlı API'ler hakkında bilgi sağlar.

- RESTful-API: Kuzeye bağlı API'lerde en sık kullanılan RESTful, World Wide Web Konsorsiyumu için geliştirilmiş tüm istemci-sunucu iletişimini kapsayan yazılım mimarisi düzenini benimsemiştir. Bu mimari, istemciler ve sunucular

arasında ara bileşenlerin eklenmesine izin vererek ölçeklenebilirlik, genellik ve esneklik sağlar. Müşteriler, ölçeklenebilirliği sağlamak için kendi durumlarını izler. Her kaynakta kullanılan GET (okuma), POST (ekle), PUT (yazma) ve DELETE (kaldır) komut sistemi aynıdır. Kontrolör, firewall kuralları, konfigürasyon, port belirleme gibi birçok uygulama RESTful-API ile tanımlanabilir. RESTful-API'nin en büyük dezavantajlarından biri, bir sayfanın ne zaman değiştiğini bildirmemesi ve sık sık yenileme gerektirmesidir. Bu nedenle geliştiriciler, güncel verilere erişmek için döngü çağrılarını periyodik olarak tekrarlarlar.

- OSGi: OSGi, Java bileşenlerini kullanan yeniden kullanılabilir paketlerden oluşan bir dizi özelliktir. Bu paketler hizmetlerini, OSGi hizmetleri kayıt defterinde yayınlar. Paketler bir yaşam döngüsü API'si aracılığıyla yüklenebilir, başlatılabilir, kaldırılabilir, durdurulabilir ve güncellenebilir. YTA denetleyici platformunun önemli bir örneği, Java tabanlı OSGi çerçevesi kullanılarak oluşturulan OpenDaylight projesidir. OSGi, Java tabanlı ağ (modül) görevlerini başlatmaya, yüklemeye, boşaltmaya ve durdurmaya izin verir. Beacon, Floodlight ve ONOS, OSGi'yi destekler.

### 2.1.6. Doğu- Batı Bağlı API'ler

YTA ağ yapısı içerisinde kontrolör-kontrolör, sunucu-sunucu veya kontrolör-sunucu arasındaki iletişim doğu-batı hattı üzerinden sağlanabilir. Aslında bu sınır, kontrol düzlemleriyle iletişim kurmak için çeşitli ağ alanlarından geçen bir kanal olarak kabul edilir. Bu kanal aracılığıyla, kontrolörler ağ durumu bilgilerini iletebilir veya yönlendirme kararlarını değiştirebilir. Özellikle büyük ağ yapılarında bu sınır çok önemlidir. Büyük ağlar birden çok küçük ağa bölündüğünde, paketleri ağ üzerinden başarılı bir şekilde yönlendirmek için genel ağ görünümüne ihtiyaç duyarlar. Doğu-batı sınırına sahip olmak, heterojen ağ işletim sistemlerinin bir ağ görünümüne sahip olmasına veya paket yollarını koordine etmek için ağ bağlantısına hâkim olmasına izin verir. Böylece her ağ alt ağlara bölünür ve topoloji, erişilebilirlik, ağ protokolleri, ağ durumu, varlıklar vb. bilgiler her alt ağda ağın kontrolöründe tutulur. Ek olarak, doğu-batı sınırı, ağ operatörünün katılımını sınırlayarak ağ kararlarını otomatikleştirir.

### 2.1.7. Simülasyon ve Emülasyon Platformları

YTA'yı iyileştirmek ve fizibilite çalışmaları yapmak için simülasyon ve emülasyon test yatakları oluşturulur. Bu test yataklarından bazıları aşağıda listelenmiştir:

- Mininet [72]: OpenFlow kullanılarak oluşturulan bir YTA ağının tek bir makine kümesi üzerinde benzetimini sağlayan platformdur ve YTA araştırmalarında en çok kullanılan emülasyon ortamıdır. Mininet, test amaçlı oluşturulan platform aracılığıyla yeni hizmetlerin geliştirilmesine olanak tanır.

- NS-3 [73]: Ağ geliştirme ortamında uzun süredir kullanılan ağ simülatörüdür ve OpenFlow anahtarları için desteklemektedir.

## 2.2. Büyük Veri

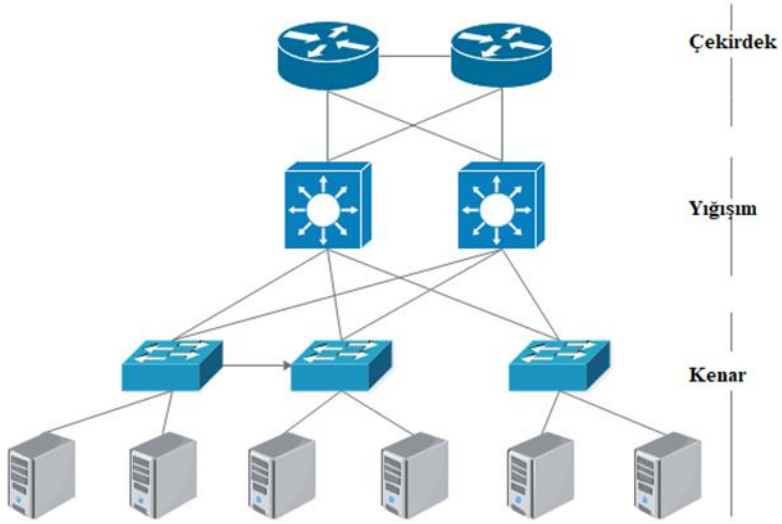
Büyük veri, geleneksel veri işleme yazılımları tarafından ele alınamayacak kadar büyük veya karmaşık olan veri kümelerini analiz etmenin, sistematik olarak bilgi çıkarmanın veya başka bir şekilde bunlarla ilgilenmenin yollarını ele alan bir alandır [74]. Büyük verileri tanımlayan beş temel prensip vardır. Bunlar; hacim, hız, çeşitlilik, değer ve doğruluktur. Bu kavramlardan hacim; büyük verilerle, yüksek hacimli düşük yoğunluklu, yapılandırılmamış verilerin işlenmesini, hız; verilerin alındığı ve üzerinde işlem yapıldığı hızı, çeşitlilik; mevcut veri türlerini, değer; verilerin içsel değerini, doğruluk; verilerin doğruluğunu ifade etmektedir.

Son teknolojik atılımlar, veri depolama ve hesaplama maliyetini oldukça azaltmış ve her zamankinden daha fazla veri depolamayı, daha kolay ve daha ucuz hale getirmiştir. Artan büyük veri hacmi artık daha ucuz ve daha erişilebilir olduğundan, verilerin analiz edilmesi, işlenmesi ve depolanması da önemli bir problem olarak karşımıza çıkmıştır. Bu problemlerin çözülmesi için çeşitli firmalar çeşitli yazılım ve donanım çözümleri önermektedir. Özellikle verilerin depolanması için CISCO, IBM, DELL çeşitli firmalar farklı çözüm önerileri sunmuş ve BVM'ler oluşturulmuştur. BVM'ler için çeşitli topolojiler önerilmiştir.

Bu farklı topolojilerin her biri, veri merkezlerinin performansında iyileştirmeler sağlamayı amaçlayan farklı kaynak gereksinimleri ile karakterize edilir. Standart ağaç tabanlı, özyinelemeli ve esnek ağ topolojileri olmak üzere farklı topoloji çeşitleri vardır.

Sunucular ve anahtarlar köşe, bağlantılar kenar olarak kabul edilirse, her veri merkezi ağı topolojisi bir graf olarak gösterilebilir. Ayrıca sabit ve esnek ağ topolojisi olmak üzere iki farklı mimariden bahsetmek mümkündür. Ağ oluşturulduktan sonra ağ topolojisi değiştirilemiyorsa, bu topoloji sabittir; standart ağaç tabanlı ve özyinelemeli topolojiler böyle bir mimariye sahiptir, ancak ağ yapısı değiştirilebiliyorsa bu esnek bir topolojidir.

Standart ağaç tabanlı mimariler ve bunların çeşitleri, BVM'lerin tasarlanmasında yaygın olarak kullanılmaktadır. Bu topolojilerden en fazla kullanılan topolojik model *k*-ary fat tree'dir. Şekil 2.2'de görülebileceği gibi fat tree, anahtarları, ağaç benzeri bir yapıya dönüştüren klasik bir BVM topolojisidir.




*Şekil 2.2.* Klasik bir *k*-ary fat tree topolojisi

Bir fat tree, tam olarak ikili ağaç yapısına dayalı bir ağdır. Uç katmanındaki her *k* bağlantı noktası anahtarı, $\frac{k}{2}$ sunucuya bağlıdır. Kalan $\frac{k}{2}$ bağlantı noktası, yığışım düzeyinde $\frac{k}{2}$ anahtara bağlanır; $\frac{k}{2}$ yığışım seviyesi anahtarı, $\frac{k}{2}$ kenar seviyesi anahtarı ve kenar anahtarlarına bağlı sunucular, pod adı verilen bir fat tree'nin temel hücresini oluşturur. Çekirdek düzeyde, *k* bölmenin her birine bağlanan $\left(\frac{k}{2}\right)^2$ *k*-port anahtarı vardır. Temel 3 seviyenin tamamı aynı tür anahtarları kullanır. Toplam ve çekirdek

seviyelerinde yüksek performanslı anahtarlar gerekli değildir. *k*-port anahtarlarına sahip bir fat tree'de maksimum sunucu sayısı $\frac{k^3}{4}$ 'tür.

### 2.3. Yük Dengeleme

Genel olarak, yük dengelemenin iki kullanımı vardır. Birincisi, yük dengeleme, iş yükünü birden çok kaynağa bölerek bir kaynağın aşırı yüklenmesini önleyen bir tekniktir. Ağdaki ölçeklenebilirliği en üst düzeye çıkarır. İkinci olarak, yük dengeleme tekniğinde, genel performansı önemli ölçüde artıran paralel işleme için birden çok düğüm cihazına tek bir ağır yük işleminin atanmasıdır. Ayrıca, herhangi bir tek kaynaktan yanıt süresi, kaynak tüketimi ve aşırı yükleme, bir yük dengeleme tekniği ile en aza indirilebilir. Geleneksel ağ yük dengeleme uygulaması, yazılım ve donanım uygulamasını birlikte içerir. Ancak, geleneksel ağlarda, yük dengeleme stratejisini küresel olarak uygulamak kolay olmadığı için tüm ağ durumunu elde etmek zordur. Ek olarak, geleneksel yük dengeleme yönteminin ağ durumundaki değişikliklere ve ayarlamalara uyum sağlaması da zordur çünkü bu cihazlar belirli donanımlar için tasarlanmış, maliyetli, esnek olmayan ve satıcıya göre farklılık gösteren yapıdadırlar. YTA'da yük dengeleme, ağ yükünü yönetmek için YTA kontrolörlerinde program kodları kullanılarak kolayca uygulanabilir.

İki tür yük dengeleme tekniği vardır: IP ağında yük dengeleme ve YTA ağında yük dengeleme. Geleneksel IP tabanlı yük dengelemede, bir yük dengeleme yönlendiricisi (YDY) yükü dengeler. Ağa yeni bir paket gelirse, öncelikle YDY'den geçer. YDY, mevcut trafik akışına göre yeni paketin hedef sunucusu olarak bir sunucu seçer ve bu sunucunun IP adresi pakete eklenir. Yönlendirme protokolleri aracılığıyla paket, hedef sunucuya hesaplanan bir yoldan iletilir. Ancak, YDY'de gerçek zamanlı, düşük yüklü sunucu seçimi kullanılır. Bu gerçek zamanlı kontrol, her yeni paket için gerçekleştirilir ve sunucu yönlendirmesini belirler. YTA'daki yük dengeleme tekniğinin farklı kombinasyonları vardır: statik, dinamik veya hem statik hem de dinamik. Statik yük dengeleme tekniğinde, yönlendirme kuralı doğrudan yük dengeleyicide programlanır, ancak ağ hakkında böyle bir bilgi bulunmadığından ağ performansı olumsuz etkilenir. Dinamik yöntemlerde yük dengeleme, statik yöntemlerden daha verimlidir çünkü yük, yük dengeleyicide programlanmış bazı modellere göre etkileşimli olarak dağıtılır.

## 2.4. Kontrolör Yerleştirme Problemi (KYP)

KYP, büyük YTA ağlarındaki kontrolörleri optimum kontrolör sayısı ve konumuna göre dağıtmak ve kontrolör-kontrolör ya da kontrolör-anahtar arasındaki gecikmeyi minimum seviyeye indirmek için araştırılması ve geliştirilmesi gereken önemli bir konudur. Ağdaki KYP araştırmasını tanımlamak için KYP formülasyonu oluşturulmuştur; YTA ağı genellikle $G = (V, E, S)$ olarak modellenir; burada $G$, bir ağın topolojisini temsil eden graf, $V$, anahtarlar kümesini, $E$, anahtarlar veya kontrolörler arasındaki fiziksel bağlantılar kümesini ve $S$, kontrolörler kümesini göstermektedir. Spesifik olarak, $n = |V|$ düğüm sayısını ve $k = |S|$ kontrolör sayısını ifade etmektedir. Denetleyici yerleştirme problemi üzerine yapılan çalışmalarda genellikle iki soru üzerinde durulur. Bu sorular; $k$'nın değeri ve $S \rightarrow V$ eşleme ilişkisini çözmek için önceden tanımlanmış bir amaç fonksiyonu ile optimize edilir.

## 2.5. Yapay Zekâ Teknikleri

Yapay zekâ, insan beyninin, düşünce yapısının, öğrenme ve karar verme gibi yeteneklerinin makineler, özellikle de bilgisayar sistemleri ile simülasyonudur. Bu simülasyonla makineler de akıllı özellikler sergileyebilmektedir. Bir makinenin yapay zekaya sahip olduğunu söyleyebilmek için birkaç temel özelliğe sahip olması gerekmektedir. Bu özellikler arasında; nedensellik, öğrenme, önem algılayabilme, seçenekler üretebilme, doğal bir dil ile iletişim kurabilme ve belirsiz durumlarda yargıya varabilme sayılabilir. Aşağıda farklı yapay zekâ algoritmaları ve özellikleri verilmiştir.

### 2.5.1. Lojistik Regresyon (LR)

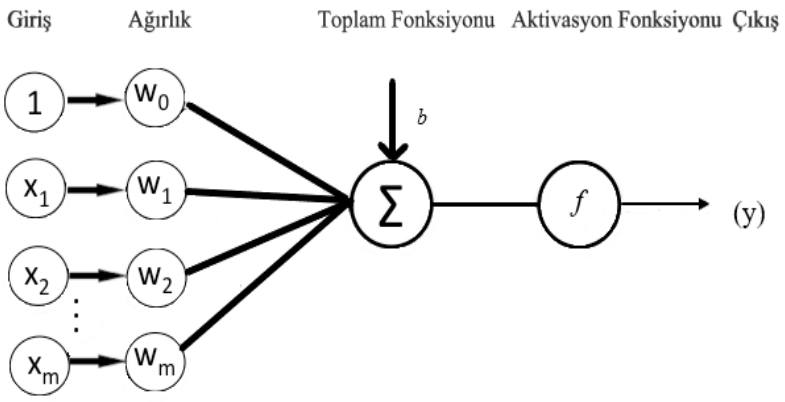
LR, bir veya birden fazla bağımsız değişkeni bulunan ve bir sonucu belirlemek için kullanılan istatistik yöntemidir. Var olan bir veri kümesinin analizinde iki olası sonuçtan birini çıktı olarak verir. Doğrusal sınıflandırma problemlerinde kullanılır. LR, ilgi karakteristiklerinin varlığının olasılığını logit dönüşümünü tahmin etmek için bir formülün katsayılarını üretir ve LR, karekök hataların toplamını ve örnek değerlerin gözlemini olasılıksal olarak en yükseğe çıkaran parametreleri seçer.

### 2.5.2. Destek Vektör Makinesi (DVM)

Makine öğreniminde, DVM, sınıflandırma ve regresyon analizi için verileri analiz eden ilişkili öğrenme algoritmaları ile denetimli öğrenme modelidir. Her biri iki kategoriden birine ait olarak işaretlenmiş bir dizi eğitim örneği verildiğinde, bir DVM eğitim algoritması, bir kategoriye veya diğerine yeni örnekler atayan bir model oluşturur ve onu olasılıksal olmayan bir ikili doğrusal sınıflandırıcı yapar. DVM'yi olasılıksal bir sınıflandırma ayarında kullanmak için ölçeklendirme mevcuttur. DVM, iki kategori arasındaki boşluğun genişliğini en üst düzeye çıkarmak için eğitim örneklerini uzaydaki noktalara eşler. Yeni örnekler daha sonra aynı alana eşlenir ve boşluğun hangi tarafına düştüklerine bağlı olarak bir kategoriye ait oldukları tahmin edilir. LR yöntemini temel alır ancak LR'den farklı olarak, optimal hiperdüzlem iki grup arasındaki en büyük ayrımı temsil eden noktaları ikiye böler. Basit bir işlev, DVM şeklini tanımlayamaz. Bu nedenle, DVM'ye ilk olarak hiperdüzlem ve türev çizgi katsayılarını tanımlayan veri noktalarını (destek vektörleri) bulma görevi verilir ve ardından yeni girdi değerleri LR'ye benzer şekilde hiperdüzlemin her iki tarafına düşerek kategorize edilir.

### 2.5.3. Yapay Sinir Ağları (YSA)

YSA, biyolojik sinir sistemlerinden esinlenmiştir. Biyolojik sinir sistemindeki bir nöron, YSA'daki bir düğüme karşılık gelir. Şekil 2.3'te YSA diyagramı verilmiştir.




*Şekil 2.3.* Bir Yapay Sinir Ağı Diyagramı

Şekil 2.3'te görüldüğü gibi girdiler bir düğüme girilen bilgilerdir. Ağırlıklar, düğümde alınan bilginin önemini ve düğüm üzerindeki etkisini gösterir. Ağırlığın daha büyük veya daha küçük bir sayı olması YSA için önemlidir. Toplama işlevi, bir düğüme net girdiyi hesaplar. En sık kullanılan toplama işlevi ağırlıklı toplamdır. Ağırlıklı toplam fonksiyonunda her bir değer ağırlığı ile çarpılır ve toplanır. Aktivasyon fonksiyonu, düğüme gelen net girdi değerini işler ve düğümün bu girdi için üreteceği çıktıyı belirler. Aktivasyon işlevi genellikle doğrusal olmayan bir işlev olarak seçilir. ReLu, sigmoid, tanjant adım vb. gibi çeşitli aktivasyon fonksiyonları vardır. Giriş 0 veya 0'dan küçükse, ReLu işlevi 0 döndürür. Giriş verileri 0'dan büyükse, ReLu işlevi giriş verilerini döndürür. Sigmoid fonksiyonunda, giriş değerlerinin her biri için 0 ile 1 arasında bir değer üretir. Son olarak, çıktı, aktivasyon fonksiyonu tarafından belirlenen değerdir. YSA algoritmaları, çok katmanlı dönüşümler yoluyla temsili öğrenmeye sahiptir ve hedefe göre tahminler yapmak için doğrusal olmayan işlem birimlerinin çoklu katmanlarını (giriş, gizli, çıkış) kullanarak ham verilerden hiyerarşik olarak bilgi çıkarır.

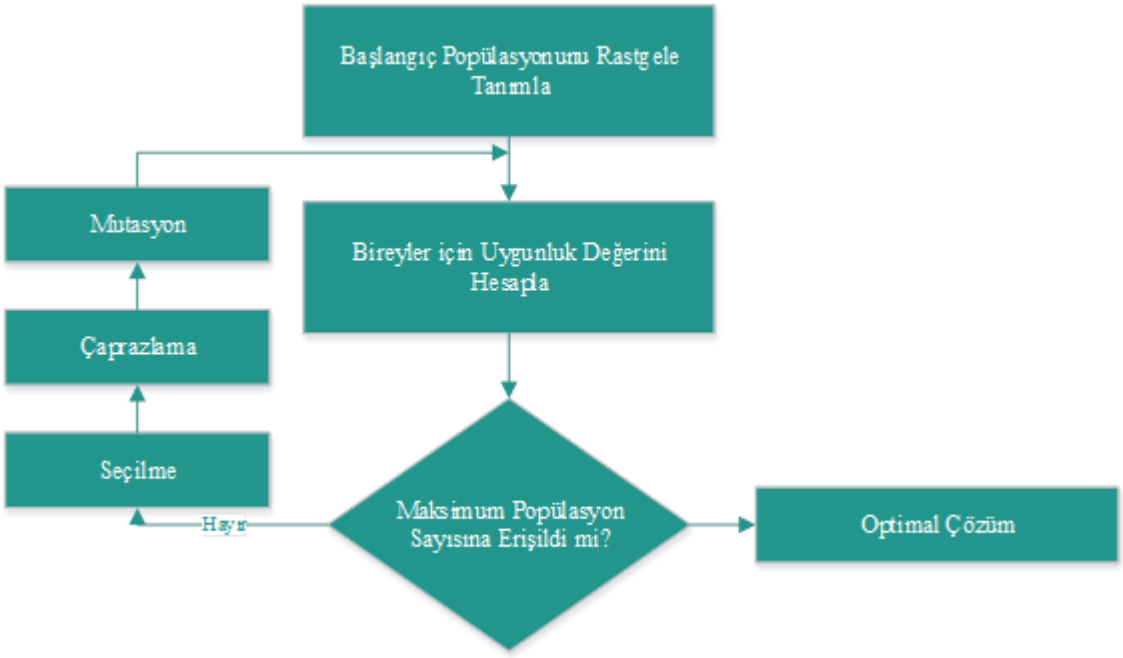
### 2.5.4. Derin Öğrenme (DÖ)

DÖ, girdi verilerinden özellikleri çıkarmak için kademeli katmanları kullanan ve sonunda karar oluşturan bir modeldir. DÖ'nün temel konsepti, artan soyutlama seviyeleri aracılığıyla veri temsillerini öğrenmektir. Hemen hemen tüm seviyelerde, daha yüksek düzeyde daha soyut temsiller, daha düşük seviyelerde daha az soyut temsiller açısından açıklama yoluyla öğrenilir. Bu tür hiyerarşik öğrenme süreci, bir sistemin karmaşık temsilleri doğrudan ham verilerden anlamasına ve öğrenmesine izin verdiği için çok etkilidir, bu da DÖ'yü birçok disiplinde faydalı kılar. DÖ uygulamasında dört önemli konu vardır. (i) DÖ ağının giriş katmanı olarak alınacak uygun sayısal formatlarda ortamın durumunun nasıl ifade edileceği (ii) doğrulama sonuçlarının nasıl temsil edileceği/yorumlanacağı, yani DÖ ağının çıktı katmanının fiziksel anlamı; (iii) ödül değerinin nasıl hesaplanacağı/güncelleneceği ve her bir sinir katmanında yinelemeli ağırlık güncellemesine rehberlik edebilecek uygun ödül işlevinin ne olacağı (iv) kaç tane gizli katman olacağı, her katmanın tasarımı ve katmanlar arasındaki bağlantıların nasıl olacağıdır. DÖ, bilgisayarla görme, mobil ve kablosuz ağlar ve doğal dil işleme gibi alanlardaki sorunları başarıyla çözmektedir [75-77].

### 2.5.5. Genetik Algoritma

GA, doğal seleksiyondan ilham alan bir optimizasyon algoritmasıdır. En uygun olanın hayatta kalması kavramını kullanan popülasyon tabanlı bir arama algoritmasıdır. Yeni popülasyonlar, popülasyonda bulunan bireyler üzerinde genetik operatörlerin yinelemeli kullanımıyla üretilir. Kromozom gösterimi, seçim, çaprazlama, mutasyon ve uygunluk fonksiyonu hesaplaması GA'nın temel unsurlarıdır.

Genellikle rastgele olarak seçilen bireylerle işleme başlanır. Başlangıç popülasyonu oluşturulduktan sonra her bir bireyin uygunluk fonksiyonu değerinin belirlenmesi gerekir. Bu uygunluk fonksiyonunun değeri problemin çeşidine göre ve çözüme olan yakınlık ile doğru orantılıdır. Uygunluk fonksiyon değeri ne kadar yüksek olursa bireyin yaşayıp üreme şansıda o kadar yüksektir. Uygunluk fonksiyon değerleri yüksek olan bireyler çaprazlama yapmak üzere seçilir. Çaprazlama işlemi, çeşitli çaprazlama kurallarının iki bireye uygulanarak yeni bir birey elde edilmesi şeklindedir. Mutasyon aşaması gen havuzunu zenginleştirmek için kullanılır. Bu adımlar gerçekleştirildikten sonra değerlendirmeye hazır yeni bir popülasyon oluşturulmuş olur. Bu adımlar sürekli yenilenerek zamanla en iyi çözüme gitmek amaçlanmaktadır. İşlemler belirlenen durdurma kriterine gelinceye kadar devam eder. Şekil 2.4'te GA akış diyagramı verilmiştir.




*Şekil 2.4.* Genetik Algoritma Akış Diyagramı

### 2.5.6. K-means ve Optimize Edilmiş K-Means Algoritması

Standart k-means algoritması [78] dört ana adımı içerir:

Adım 1. Rastgele örnekleme kullanarak *k* kümeyi başlatın ve her kümeye bir merkez atayın.

Adım 2. Öklid mesafesine göre düğümleri kümelerden birine atayın.

Adım 3. Her küme için tekrar ağırlık merkezini hesaplayın.

Adım 4. Her kümede değişiklik kalmayana kadar 2. ve 3. adımları tekrarlayın.

Standart k-means algoritması, ağ bölümleme topolojilerine doğrudan uygulanamaz. Çünkü k-means algoritması, öklid mesafelerini kullanarak iki düğüm arasındaki mesafeyi hesaplayarak başlangıç merkezlerini rastgele seçer. Bu sebeple, denetleyicinin yerleştirildiği ağırlık merkezi yeri garanti edilemez. Bu problemin ortadan kalkması için, k-means algoritması optimize edilmelidir.

Yeni bir yöntem olarak bu tez çalışmasında önerilen optimize edilmiş k-means algoritmasının ilk adımında, ağın merkezi düğümü rastgele seçilir. Böylece algoritma, ağın gerçek merkezini daha iyi öğrenecektir. İkinci adımda, merkezden tüm düğümlere giden en kısa yol seçilir. Başlangıçta sonsuz uzaklığa sahip olan her düğüm, kendisine en yakın olan kümeye atanır. Dijkstra algoritması en kısa yolu seçmek için kullanılır. Üçüncü adımda, tüm noktalardan en kısa mesafenin toplamını en aza indirgemek için ağırlık merkezi güncellenir. Ağ, belirlenen minimum gecikme süresi parametresine göre seçilen *k* adet alt ağa bölünene kadar işlem devam eder. Böylece, maksimum gecikme, k-means gibi standart kümeleme algoritmalarına kıyasla önemli ölçüde azaltılır.

### 2.5.7. Yapay Arı Koloni Algoritması ve Hibrit Yapay Arı Koloni Algoritması

YAK algoritması, bir arı kolonisinin yiyecek arama davranışını simüle eden bir optimizasyon algoritmasıdır [79]. YAK algoritmasının simüle ettiği bir bal arısı kolonisinde sürü işçi, gözcü ve kâşif arılar olmak üzere üç çeşit arıdan oluşur. Koloninin yarısı işçi arılardan, diğer yarısı da gözcü arılardan oluşur. İşçi arılar, daha önce keşfedilen nektar kaynaklarından yararlanmakla ve kovandaki bekleyen arılara (gözcü arılar) kullandıkları besin kaynağı alanlarının kalitesi hakkında bilgi vermekle sorumludur. Gözcü arılar, işçi arıların paylaştığı bilgilere göre kovanda bekler ve yararlanacakları besin kaynağına karar verirler. Kâşif arılar, ya içsel bir motivasyona bağlı olarak ya da olası dış parametrelere dayalı olarak yeni bir besin kaynağı bulmak

için çevreyi rastgele araştırırlar. Toplayıcı arılarda ortaya çıkan bu akıllı davranış şu şekilde özetlenebilir:

1. Yiyecek arama sürecinin ilk aşamasında, arılar bir besin kaynağı bulmak için rastgele çevreyi keşfetmeye başlarlar.

2. Bir besin kaynağı bulduktan sonra arı, çalışan bir toplayıcı olur ve keşfedilen kaynaktan yararlanmaya başlar. İşçi arı nektarla kovana döner ve nektarı boşaltır. Nektarı boşalttıktan sonra, doğrudan keşfettiği kaynak alanına geri dönebilir veya dans alanında bir dans gerçekleştirerek kaynak alanı hakkında bilgi paylaşabilir. Kaynağı tükenirse kâşif olur ve rastgele yeni bir kaynak aramaya başlar.

3. Kovanda bekleyen gözcü arılar, kaynak dansını izlerler ve kaynağın kalitesiyle orantılı olarak bir dansın sıklığına göre bir kaynak alanı seçerler. Karaboğa tarafından [79] önerilen YAK algoritmasında, bir besin kaynağının konumu, optimizasyon problemine olası bir çözümü temsil eder ve bir besin kaynağının nektar miktarına karşılık gelir. Her besin kaynağı sadece bir işçi arı tarafından kullanılır. Diğer bir deyişle işçi arı sayısı, kovan çevresinde bulunan besin kaynaklarının sayısına (popülasyondaki çözüm sayısı) eşittir. Besin kaynağını terk eden işçi arı, kâşif olur. Temel YAK'ın adımları Şekil 2.5'te verilmiştir.

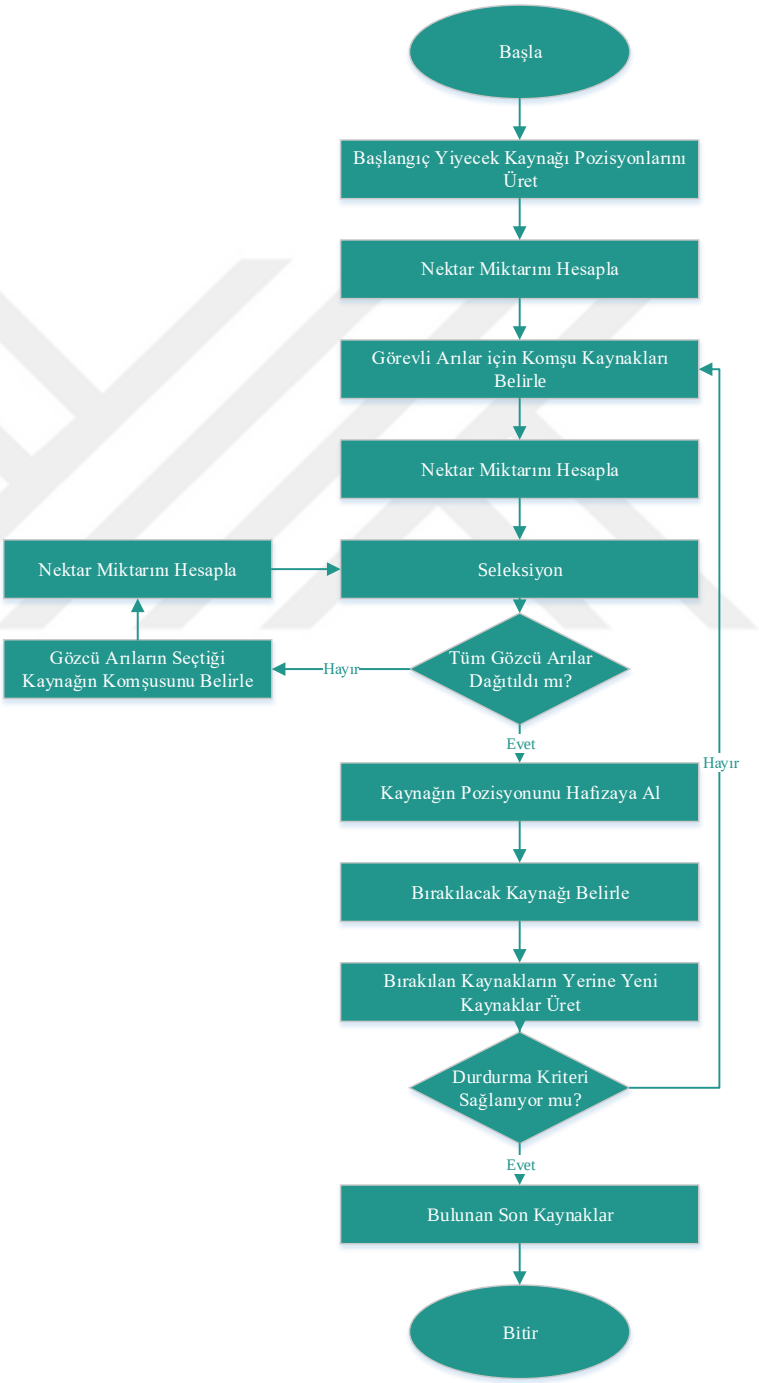
YAK algoritmasında, en iyi çözüm gbest değeri rastgele üretilerek, komşuluk ve mutasyon değerini belirleme aşamasında kullanılmaktadır. YAK algoritmasından farklı olarak, Hibrit-YAK algoritmasında gbest'in başlangıç değerinin belirlenmesinde basitliği ve düşük zaman tüketimi nedeniyle k-means algoritması kullanılmıştır. Böylece algoritmadaki gbest değeri anlamlı bir değerle başlatılmış ve algoritmanın yakınsaması hızlandırılmıştır. Hibrit-YAK algoritmasının adımları şu şekildedir;

Adım 1: Başlangıç yiyecek kaynaklarının rastgele belirlenmesi
Adım 2: k-means algoritması ile üretilen g-best değerinin belirlenmesi
Adım 3: Yap
    Adım 4: Görevli arıların yiyecek kaynaklarına gönderilmesi
    Adım 5: Seleksiyonda kullanacak yiyecek kaynaklarının olasılık değerlerinin hesaplanması

Adım 6: Gözcü arıların olasılık değerlerine göre seçtikleri yiyecek kaynaklarına gönderilmeleri

Adım 7: Kâşif arıların yeni yiyecek kaynaklarını bulmaya çıkması

Adım 8: Sonlandırma Koşulu Sağlanana Kadar Devam Et

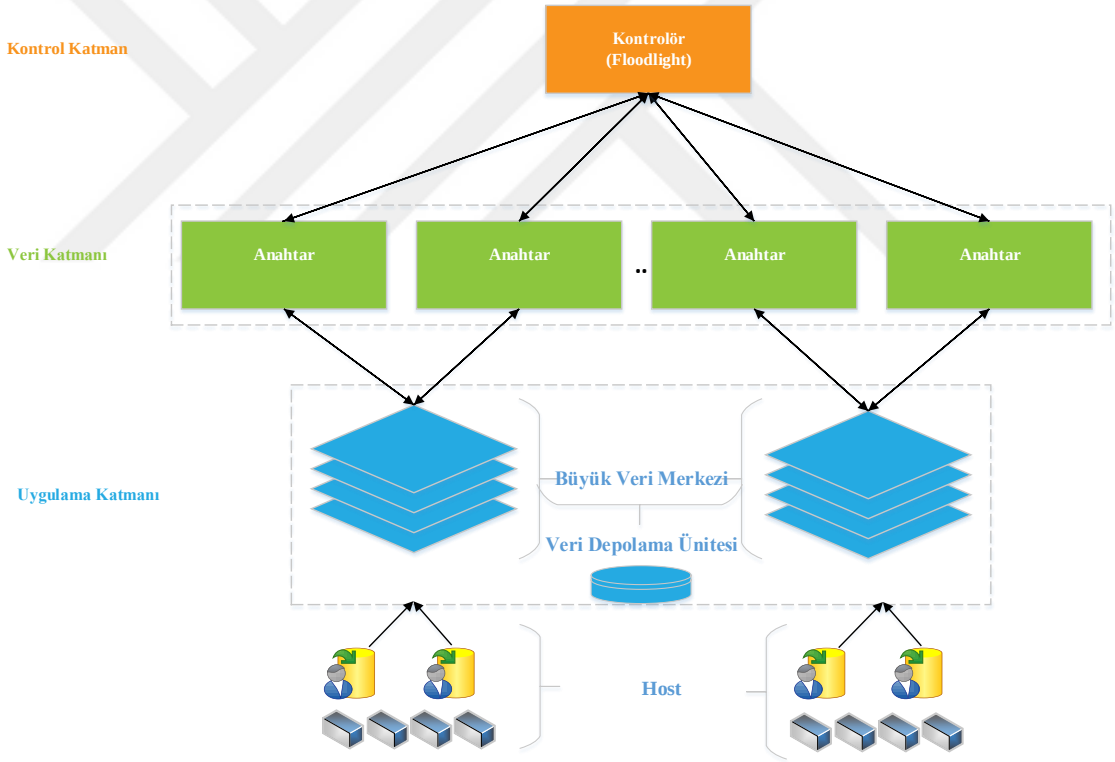



*Şekil 2.5.* Yapay Arı Koloni Algoritması Akış Diyagramı

## 2.6. Yöntem

### 2.6.1. YTA tabanlı BVM'lerde Yük Dengeleme

Bu tez çalışmasında, YTA tabanlı BVM'lerin yük dengelemesine DÖ tekniği literatürde ilk kez bir uygulanmıştır. DÖ ağını eğitmek için bağlantılar arasındaki değişken yük değerleri kullanılmıştır. DÖ algoritmasının performansı, yük dengeleme için yanıt süresi açısından YSA, DVM ve LR gibi algoritmaların performansıyla karşılaştırılır. Mininet emülatör ortamıyla oluşturulan sistem şeması Şekil 2.6' da verilmiştir. Kontrol katmanından tek bir kontrolör kullanılmış ve kontolör olarak Floodlight seçilmiştir. Veri katmanında anahtarlar kullanılmıştır. Uygulama katmanında son kullanıcı olan hostlarla iletişimin sağlandığı büyük veri merkezleri kullanılmıştır.

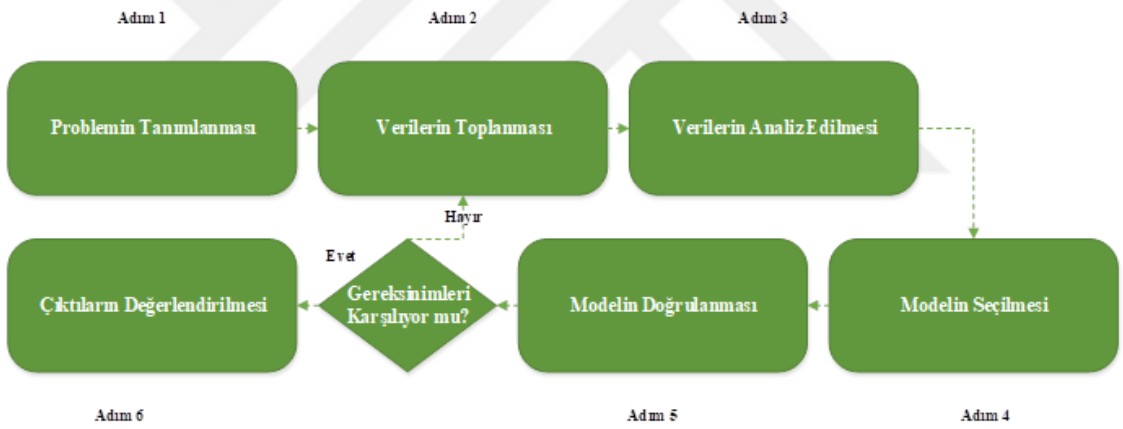



*Şekil 2.6.* YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Oluşturulan Sistemin Şeması

Sistem tasarımı oluşturulduktan sonra, yük dengeleme için iki ana bilgisayar seçilmiştir. Belirlenen iki yol arasında yük dengelemeyi gerçekleştirmek için Floodlight'ta iletim

hızı (tx) ve alma hızı (rx) etkinleştirilmiştir. Iperf ile trafik üretimi gerçekleştirildikten sonra belirlenen yollar arasındaki maliyetler toplanmıştır. Rota bilgilerini almak için RESTful-API kullanılmıştır. Ayrıca IP'ler, anahtarlar, MAC adresleri ve bağlantı noktası eşleme bilgileri RESTful-API'den alınmıştır. Bu bilgiler kullanılarak belirlenen iki yol arasındaki maliyet (belirlenen yol bilgisi ve yol maliyet verisi) milisaniye cinsinden elde edilir ve bir metin (.txt) dosyasında tutulur. Floodlight'ta maliyet toplanır ve RESTful-API uygulamalarını gerçekleştirmek için belirli ayarlar güncellenir. Yolların maliyeti, iletim hızı (bir paket iletimi sırasında iletilen değer) ile alma hızının (bir paket iletimi sırasında alınan değer) toplamıdır.

Toplanan veriler LR, DVM, YSA ve DÖ algoritmalarının eğitimi için kullanılır. LR, DVM, YSA ve DÖ teknikleriyle yük dengeleme uygulamasının iş akış diyagramı Şekil 2.7'de görülmektedir.




*Şekil 2.7.* YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Ağ Uygulamalarında Kullanmak Üzere Geliştirilen Yapay Zekâ Teknikleri İş Akışı

Şekil 2.7'de görüleceği gibi, iş akışı altı adımdan oluşmaktadır. Bu adımlar; problemin tanımlanması, verilerin toplanması, verilerin analiz edilmesi, modelin seçilmesi, modelin doğrulanması ve çıktıların değerlendirilmesidir.

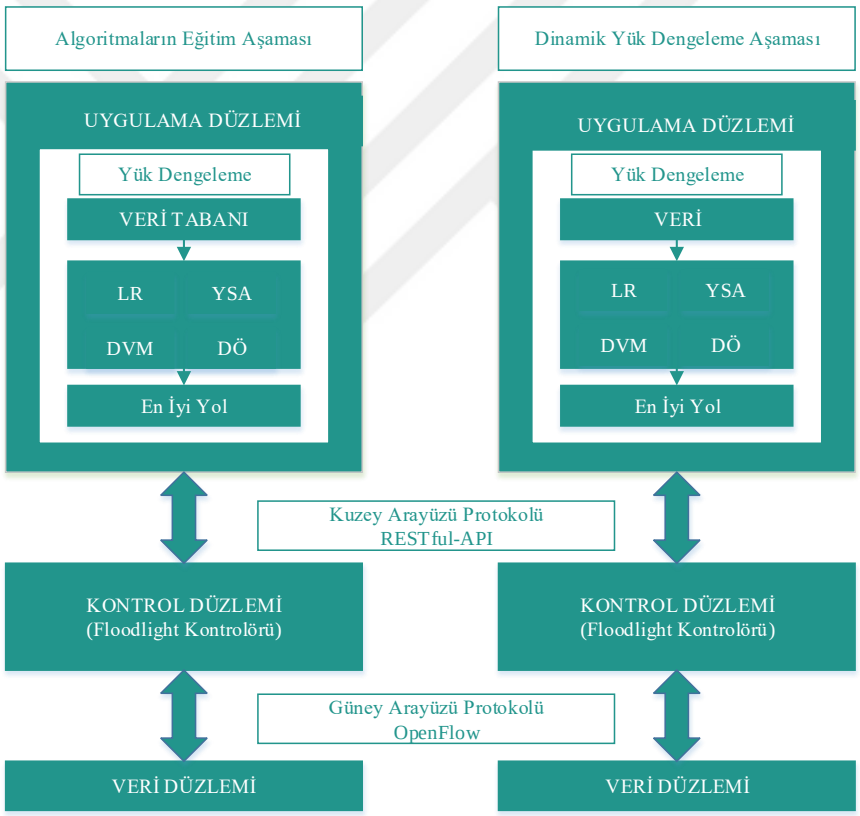
Geliştirilen program verileri her dakika güncellemeye devam ederek sistemi dinamik hale getirir. Algoritmalardan dinamik olarak en iyi yol bilgisi elde edildikten sonra bu

en iyi yol bilgisi yük dengeleme için kullanılır. Simülasyonlarda kullanılan yük dengeleme mekanizmasının blok diyagramı Şekil 2.8'de gösterilmiştir.

YTA tabanlı BVM'lerde yük dengelemesi için, iki farklı senaryo incelenmiştir. İlk senaryoda, bağlantılar MiniEdit'te oluşturulur ve yük dengeleme için iki ana bilgisayar (Host 1 (h1) ve Host 4 (h4)) seçilmiştir. İki ana bilgisayar arasındaki rotalara ilişkin maliyet bilgileri, h1'den h4'e kadar olan iki yolun RESTful-API'si tarafından toplanmış, düşük maliyetli bir rota seçilmiştir. Bu yollar h1 - switch7 (s7) - switch4 (s4) - switch8 (s8) - h4, h1 - s7 - switch3 (s3) - s8 - h4'tür. Anahtarların MAC adresleri ve yol maliyeti verilerek RESTful-API üzerinden yolların maliyet değerleri elde edilmiştir. Örnek bir veri satırı ""02 :: 15 :: 01": 288, "02 :: 0a :: 01": 1530" şeklindedir. Elde edilen yol bilgileri ve maliyet değerlerine göre sınıflandırma değerleri veri tabanına eklenmiştir. Veri tabanındaki örnek bir veri satırı "288, 1530, 0"'dır.


*Şekil 2.8.* YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Oluşturulan Sistemin Akış Diyagramı

İkinci senaryoda, birinci senaryodan farklı olan iki yol üzerinde yük dengeleme gerçekleştirilmiştir. Mininet bağlantıları oluşturulur ve aynı anda iki farklı rota (Host 1 (h1)- Host 3 (h3)) ve (Host 6 (h6)- Host 7 (h7)) seçilmiştir. Bu ana bilgisayarlar arasındaki yol maliyeti bilgileri RESTful-API tarafından toplanmıştır. h1'den h3'e ve h6'dan h7'ye iki farklı rota vardır ve düşük maliyetli rotalar seçilmiştir. Elde edilen yol bilgisi ve maliyet değerlerine göre sınıflandırma değerleri veri tabanına eklenmiştir. Veri tabanındaki örnek veri satırı "1254, 694, 1 ve 81659, 80769, 1"" şeklindedir. Bu şekilde 800 eğitim ve 192 test verisi oluşturulmuştur. Şekil 2.9'da oluşturulan sistemin blok diyagramı verilmiştir.




*Şekil 2.9.* YTA tabanlı BVM'lerde Yük Dengelemesi Yapılması için Oluşturulan Sistemin Blok Diyagramı

### 2.6.2. YTA'da Kontrolör Yerleştirme Problemine Çözüm Üretilmesi

#### 2.6.2.1.Genetik Algoritma Yaklaşımı

GA yaklaşımında, YTA'larda kontrolör yerleştirme problemi için DA temelli bir genetik algoritma yaklaşımı sunulmuştur. Çalışmada, uçtan uca gecikmenin minimum seviyeye indirilmesi amaçlanmıştır.

Öncelikle Topology Zoo [80] veri tabanı bünyesinde bulunan ULAKNET ve Colt veri setlerindeki enlem ve boylam değerleri kullanılarak Türkiye ve Avrupa için sabit birer node haritaları çizdirilmiştir. Çizilen haritalarda, her bir şehir bir numara ile temsil edilmiştir. Daha sonra Dijkstra algoritması kullanılarak haritalardaki her bir şehrin diğer tüm şehirlere olan en kısa mesafelerinin olduğu birer matris oluşturulmuştur. KYP'de GA'nın uygulanabilmesi için birey sayısı, ebeveyn yüzdesi, mutasyon oranı, rastgele seçim oranı ve jenerasyon sayısı parametreleri belirlenmiştir. Bu parametrelerden yola çıkarak çalışma başında rastgele üç noktanın seçildiği 100 adet 1*3'lük matris oluşturulmuştur. Bu matrislerden birisi denetleyicilerin en iyi yerleşim yerlerini içerecektir. Her bir bireyin uygunluk fonksiyonu her jenerasyonun sonunda hesaplanmıştır. Uygunluk fonksiyonunun değeri bir şehrin diğer şehirlere olan uzaklıkları ile ters orantılıdır. Her bir jenerasyonda bir çözüm kümesi için ortalama bir uygunluk fonksiyonu belirlenmiştir. Buradaki çözüm yerel optimum noktası ve bulunan ortalama değer yerel optimum değeridir. Tüm jenerasyonlar tamamlandıktan sonra yerel optimum noktası ve yerel optimum değerleri arasından en optimum değer seçilerek global optimum olarak atanmıştır. Seçilen bazı bireyler mutasyona uğratıldıktan sonra çaprazlama yöntemi ile yeni bireyler elde edilmiştir. Her bir jenerasyon için optimum değerler tutulmuş ve tutulan değere göre denetleyici yerleşim problemi için bir yakınsama grafiği elde edilmiştir.

#### 2.6.2.2.K-means ve Optimize Edilmiş K-means Algoritması Yaklaşımı

Bu yaklaşımda, çoklu kontrolör yerleştirme problemi optimize edilmiş k-means ve standart k-means algoritmaları ile çözümlenmiştir. Çalışmanın amacı, KYP'de uçtan uca gecikmeyi azaltmaktır. Önerilen test ortamı çoklu kontrolör için yüksek kullanılabilirlikli bir kontrol düzlemi sağlar ve optimize edilmiş k-means algoritması, test ortamındaki standart k-means ile karşılaştırılır.

Çalışmada ilk olarak, Topology Zoo veri tabanından kontrolör ve yerleştirilecek anahtar noktaları için ULAKNET veri seti elde edilmiştir. ULAKNET veri seti, şehirlerin enlem ve boylam bilgilerini içerir. Enlem ve boylam bilgisi olan şehirlerin, şehirler arası mesafesi ölçülmüştür. Elde edilen mesafe değerleri bir dizide tutulmuştur. Ayrıca kontrolörlerin kontrolörlere ve kontrolörlerin anahtarlara olan mesafesini bulmak için iki ayrı fonksiyon oluşturulmuştur. Mesafe değerlerinin tutulduğu dizi kullanılarak, Dijkstra algoritması ile en kısa mesafelerin hesaplanması sağlanmıştır. Optimize edilmiş k-means algoritması ve standart k-means algoritması bu dizi değerleri ile çalıştırılır. *k* değeri, optimize edilmiş k-means'de gecikmeyi sınırlayıcı faktör olarak belirlenirken, standart k-means algoritmasında kontrolör sayısını sınırlayıcı faktör olarak belirlenmiştir. Optimize edilmiş k-means algoritmasında ilk olarak, ağın merkezi düğümü rastgele seçilmiştir. Merkez rastgele seçildiğinden, ikinci adımda ağın gerçek merkezi daha iyi anlaşılacaktır. Spesifik olarak, algoritmada her bir düğümün diğer düğümlere olan uçtan uca toplam gecikmesi hesaplanmış ve merkez olarak minimum toplam gecikmeye sahip olan seçilmiştir. Üçüncü adımda, algoritmada ağın ikinci merkezi bulunacaktır. İkinci merkez, merkeze karşı en büyük uçtan uca gecikmeye sahip olan düğüm olarak seçilmiştir. Dördüncü adımda algoritmada, ilk merkez ve ikinci merkez, iki başlangıç merkezi olarak ele alınmış ve ilişkili düğümler bu merkezlere tahsis edilmiştir. Her bir düğüm için bu iki merkeze olan gecikme hesaplanmış ve iki gecikme elde edilmiştir. Bu iki gecikme karşılaştırılmış ve düğüm, kendisine yakın olan merkeze atanmıştır. Düğüm bir merkeze atandığında, bu kümenin merkezi, ikinci adımda açıklanan minimum toplam uçtan uca gecikmeye bağlı olarak yeniden hesaplanmıştır. Süreç, ağ "*k*" alt ağına bölünene kadar devam ettirilmiştir. Tüm "*k*" merkezlerini bir kerede rastgele seçen ve ardından tekrarlamalar sırasında hepsini optimize eden düzenli kümeleme algoritmalarından farklı olarak, önerilen algoritma ilk önce tüm ağı iki alt ağa, ardından üç alt ağa böler. Her bölüm sırasında, merkezler ve ilişkili düğümler arasındaki maksimum uçtan uca gecikmeyi kısaltabilir ve dolayısıyla maksimum gecikme k-center ve k-means gibi standart kümeleme algoritmalarına kıyasla önemli ölçüde azaltılır. Çalışma sonucunda elde edilen veriler grafik yöntemi ile haritalanmıştır.

#### 2.6.2.3.YAK ve Hibrit-YAK Algoritması Yaklaşımı

Bu yaklaşımda, çoklu kontrolör yerleştirme problemi YAK ve Hibrit-YAK algoritmaları ile çözümlenmiştir. Çalışmanın amacı, KYP'de uçtan uca gecikmenin azaltılması amacıyla optimum kontrolör yerlerini bulmaktır. Önerilen test ortamı çoklu kontrolör için yüksek kullanılabilirlikli bir kontrol düzlemi sağlar ve YAK algoritması, test ortamındaki Hibrit-YAK algoritması ile karşılaştırılmıştır.

İlk olarak veri setleri oluşturulmuştur. Veri setlerinin oluşturulmasında, Topology zoo veri tabanındaki ULAKNET veri setinde, şehirler arasındaki bağlantılardan yararlanılmıştır. Veri setinin oluşturulmasında, iki şehrin arasındaki node (şehir) sayısı DA algoritması kullanılarak en kısa olacak şekilde belirlenmiştir. Tüm şehirlerin, diğer şehirlerle arasındaki node değeri 81*81'lik bir matriste tutulmuştur. Bu matris ilk veri setini oluşturmuştur. İkinci veri seti ise, ilk veri setinde elde edilen matris değerlerinin, her bir şehrin nüfus yoğunluk oranıyla çarpılması sonucunda oluşturulmuştur. Bu iki veri seti algoritmalara girdi olarak verilmiştir. YAK ve Hibrit-YAK algoritmalarının popülasyon sayısı, limit değeri ve maksimum iterasyon sayısı belirlenmiş ve algoritmalar çalıştırılmıştır.

.

# 3. BÖLÜM

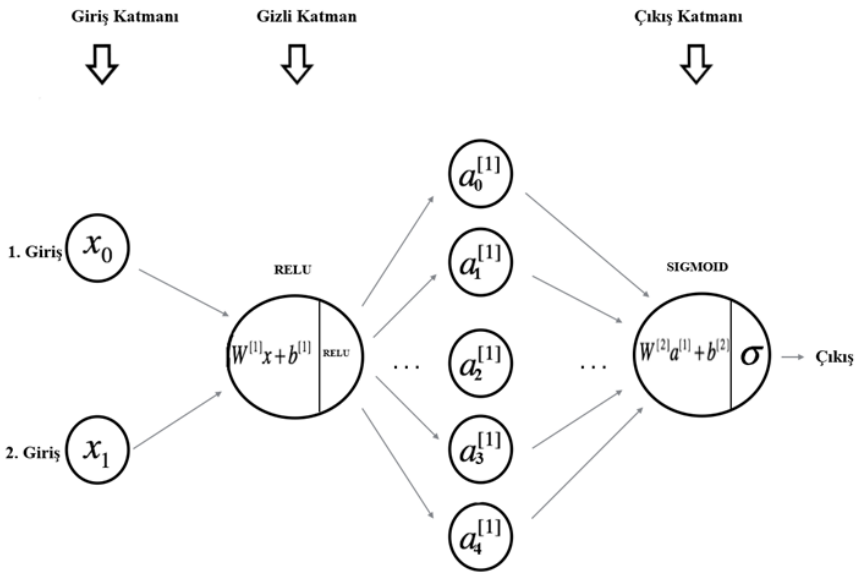
# BULGULAR

Bu bölümde, gerçekleştirilen BVM için YTA'larda yük dengeleme çalışmasının ve YTA'da kontrolör yerleştirme problemine önerilen, genetik algoritma yaklaşımı, k-means ve optimize edilmiş k-means yaklaşımı ve yapay arı koloni algoritması yaklaşımının sonuçları verilmiştir. Elde edilen sonuçların değerlendirilmesi tartışma bölümünde anlatılmıştır.

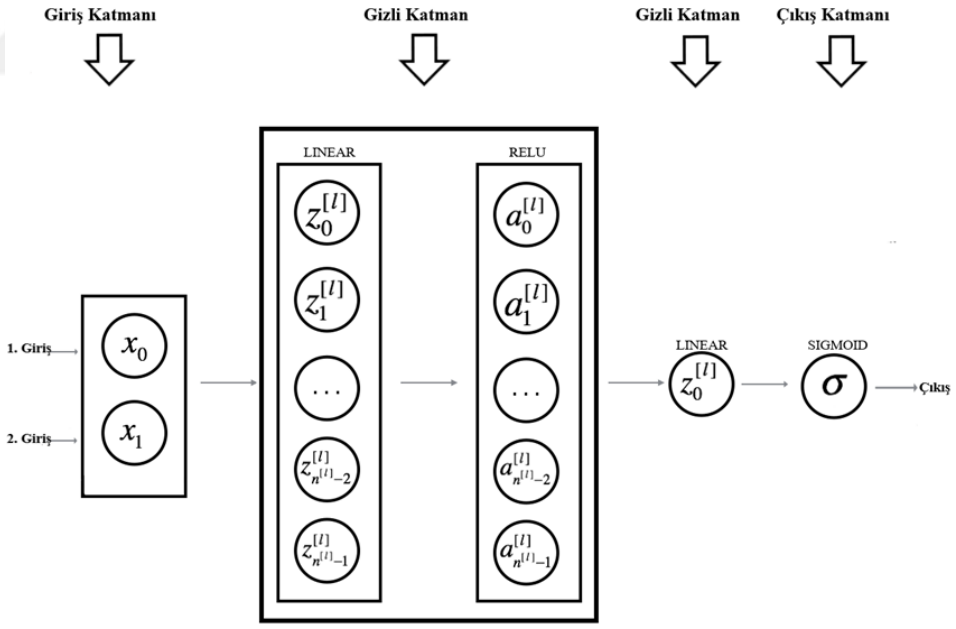
## 3.1. YTA Tabanlı BVM'lerde Yük Dengeleme

YTA tabanlı BVM'lerde yük dengeleme için yapılan tüm simülasyon deneyleri, Intel i7 işlemci ve 8 GB RAM'li bir bilgisayarda Ubuntu 16.04 işletim sistemi üzerinde gerçekleştirilmiştir. BVM yapısı olarak *k*-ary fat tree topolojisi seçilmiştir ve topolojinin *k* parametre değeri 4 olarak ayarlanmıştır. LR, DVM, YSA ve DÖ algoritmalarının kodları Python dilinde yazılmış ve derleyici olarak Jupyter Notebook kullanılmıştır. YTA kontrolörü olarak Floodlight ve simülasyon ortamı olarak mininet aracı kullanılmıştır. DVM algoritmasında *C* parametresi 1 ve 100 olarak belirlenmiştir.

Şekil 3.1'de görüldüğü gibi, iki katmanlı bir YSA modeli kullanılmıştır. Bu modelde birinci katman için beş düğüm, ikinci katman için ise bir düğüm seçilmiştir. Öğrenme oranı 0.00025 olarak alınmıştır. Birinci ve ikinci katmanlar için sırasıyla ReLu ve sigmoid aktivasyon fonksiyonları kullanılmıştır. Şekil 3.1'de *x0* ve *x1* değerleri, birinci ve ikinci yollar için girdi değişkenleridir. *A, w* ve *b* parametreleri sırasıyla aktivasyon, ağırlık ve sapma değerlerini belirtmektedir.




*Şekil 3.1.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Kullanılan YSA Modeli




*Şekil 3.2.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Kullanılan DÖ Modeli

Şekil 3.2'de YTA tabanlı BVM'de yük dengeleme problemi için dört katmanlı bir DÖ modeli önerilmiştir. Önerilen modelin birinci, ikinci, üçüncü ve dördüncü düğümlerinde

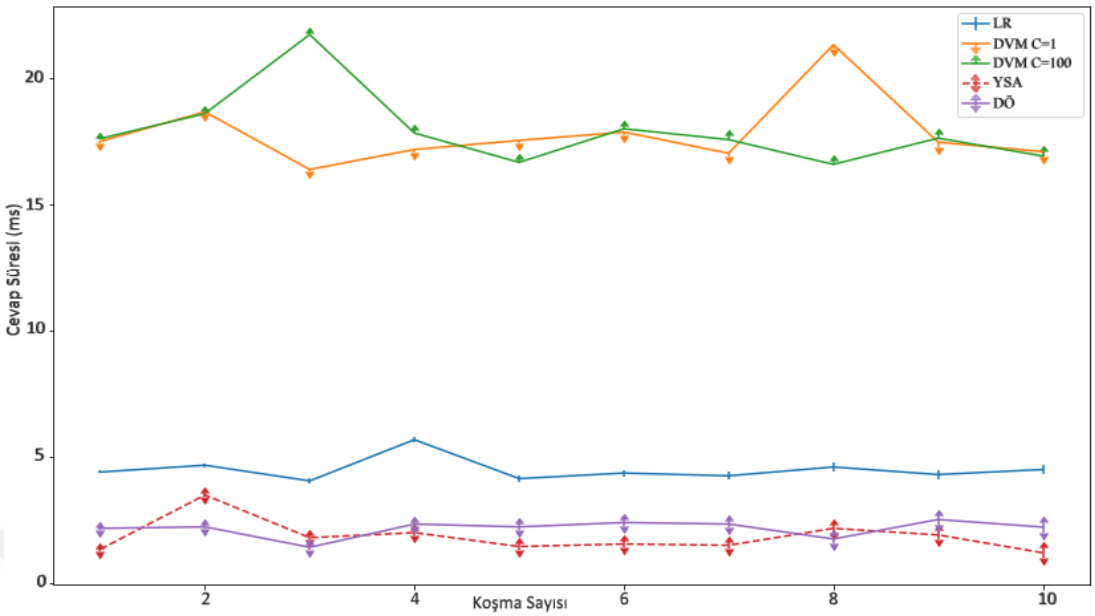
sırasıyla 20, 7, 5, 1 düğüm sayıları olarak alınmıştır. ReLu, dördüncü katman hariç tüm katmanlarda aktivasyon fonksiyonu olarak kullanılmıştır. Dördüncü katmanda sigmoid aktivasyon fonksiyonu kullanılmıştır. *x0* ve *x1*, YSA modelinde olduğu gibi girdi değişkenleridir. Üst simge *[l]*, l. katmanla ilişkili bir niceliktir. Alt simge *n*, bir vektörün n'inci girişini belirtir. *Z* aktivasyon öncesi parametre olarak da adlandırılan aktivasyon fonksiyonunun girişidir.

YTA tabanlı BVM yük dengeleme problemi için, önerilen DÖ algoritmasının yanında LR, DVM, YSA algoritmaları çalıştırılmış ve elde edilen test verilerine göre her bir algoritmanın yanıt süresi iki farklı senaryo için hesaplanmıştır.

Birinci ve ikinci senaryo için LR, DVM (*C*=1 ve *C*=100 için), YSA ve DÖ algoritmaları çalıştırılmıştır. Birinci senaryo için milisaniye cinsinden elde edilen yanıt süresi Tablo 3.1 ve Şekil 3.3'te, ikinci senaryo için elde edilen yanıt süresi ise sırasıyla Tablo 3.2 ve Şekil 3.4'te verilmiştir. 100 kez ardışık algoritma yürütmesinden sonra elde edilen milisaniye cinsinden ortalama yanıt süreleri sırasıyla Tablo 3.3 ve Tablo 3.4'te listelenmiştir.

Tablo 3.1. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi (ms)

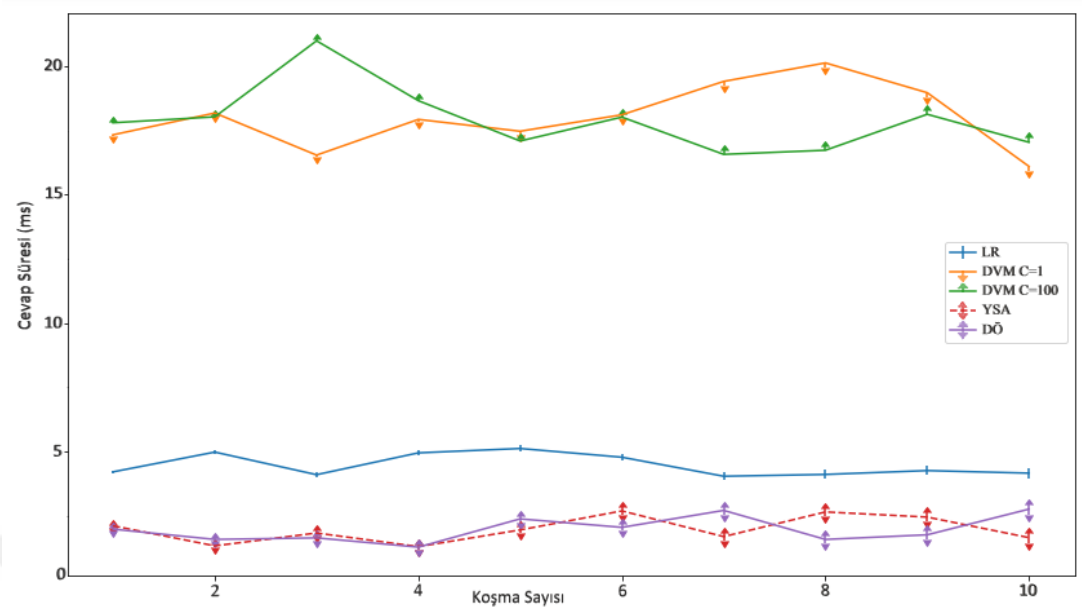
| **Koşma Sayısı** | **LR** | **DVM C = 1** | **DVM C = 100** | **YSA** | **DÖ** |
|---|---|---|---|---|---|
| 1 | 4.40 | 17.49 | 17.60 | 1.34 | 2.17 |
| 2 | 4.67 | 18.66 | 18.59 | 3.49 | 2.23 |
| 3 | 4.05 | 16.38 | 21.72 | 1.80 | 1.42 |
| 4 | 5.68 | 17.17 | 17.82 | 2.00 | 2.34 |
| 5 | 4.14 | 17.53 | 16.66 | 1.45 | 2.23 |
| 6 | 4.36 | 17.86 | 17.99 | 1.55 | 2.40 |
| 7 | 4.25 | 17.02 | 17.56 | 1.50 | 2.34 |
| 8 | 4.6 | 21.32 | 16.58 | 2.17 | 1.75 |
| 9 | 4.3 | 17.46 | 17.62 | 1.91 | 2.52 |
| 10 | 4.5 | 17.08 | 16.91 | 1.20 | 2.22 |




*Şekil 3.3.1.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi Grafiği

Tablo 3.2. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi (ms)

| **Koşma Sayısı** | **LR** | **DVM C = 1** | **DVM C = 100** | **YSA** | **DÖ** |
|---|---|---|---|---|---|
| 1 | 4.20 | 17.32 | 17.79 | 2.10 | 1.97 |
| 2 | 4.97 | 18.17 | 18.02 | 1.32 | 1.57 |
| 3 | 4.09 | 16.53 | 20.98 | 1.82 | 1.63 |
| 4 | 4.94 | 17.92 | 18.65 | 1.29 | 1.27 |
| 5 | 5.11 | 17.46 | 17.08 | 1.94 | 2.37 |
| 6 | 4.77 | 18.10 | 18.01 | 2.68 | 2.04 |
| 7 | 4.03 | 19.40 | 16.56 | 1.68 | 2.70 |
| 8 | 4.10 | 20.12 | 16.72 | 2.64 | 1.57 |
| 9 | 4.25 | 18.96 | 18.12 | 2.44 | 1.75 |
| 10 | 4.15 | 16.09 | 17.03 | 1.64 | 2.75 |




*Şekil 2.6.23.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryo için LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Yanıt Süresi Grafiği

Tablo 3.3. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Ortalama Yanıt Süresi (ms)

| **LR** | **DVM C = 1** | **DVM C = 100** | **YSA** | **DÖ** |
|---|---|---|---|---|
| 5.20 | 17.98 | 18.14 | 1.85 | 2.18 |

Tablo 3.4. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda LR, DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Ortalama Yanıt Süresi (ms)

| **LR** | **DVM C = 1** | **DVM C = 100** | **YSA** | **DÖ** |
|---|---|---|---|---|
| 5.22 | 17.66 | 18.26 | 1.69 | 1.97 |

Sınıflandırıcı sistemin doğruluğunun ölçülmesi ve değerlendirilmesi amacıyla veri setine 10 kat çapraz doğrulama uygulanmıştır. İki senaryoya göre çapraz doğrulama ile elde edilen algoritmaların sınıflandırma performansları sırasıyla Tablo 3.5 ve Tablo 3.6'da verilmiştir.

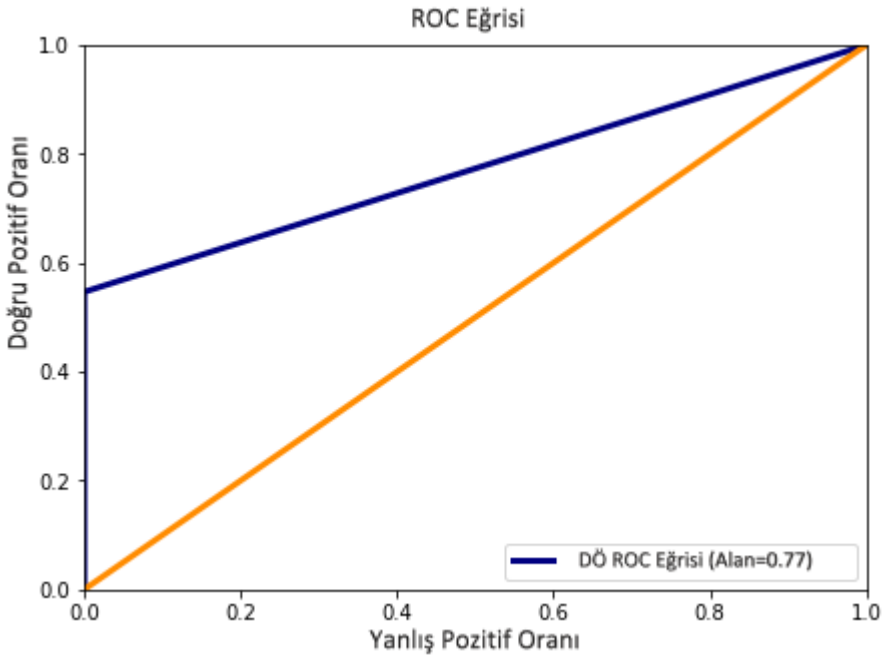
Tablo 3.5. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Doğruluk Oranları

| **DVM C = 1** | **DVM C = 100** | **YSA** | **DÖ** |
|---|---|---|---|
| 0.96 | 0.98 | 0.61 | 0.82 |

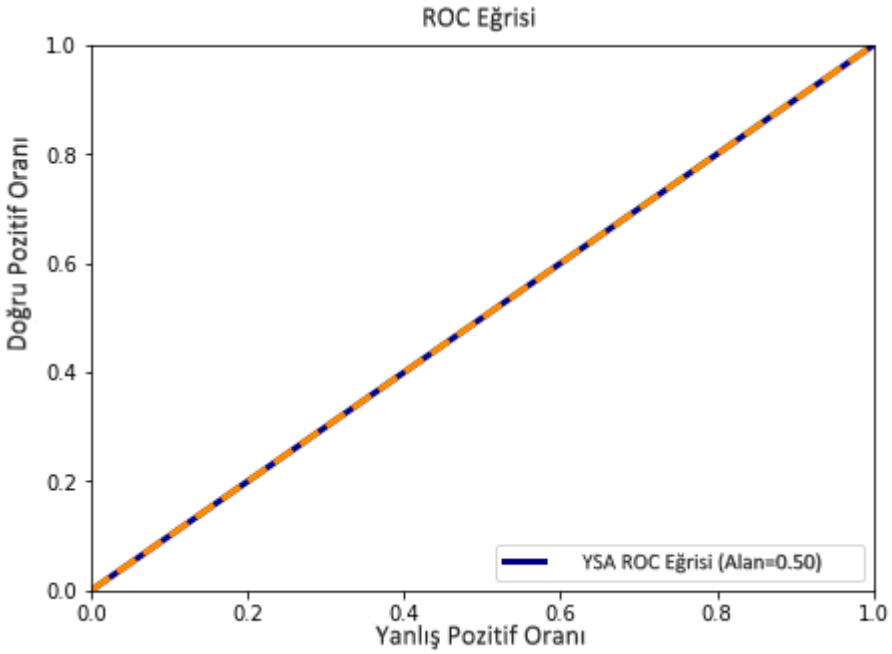
Tablo 3.6. YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryo için DVM (*C*=1 ve *C*=100), YSA ve DÖ Algoritmalarının Doğruluk Oranları

| **DVM C = 1** | **DVM C = 100** | **YSA** | **DÖ** |
|---|---|---|---|
| 0.94 | 0.98 | 0.67 | 0.92 |

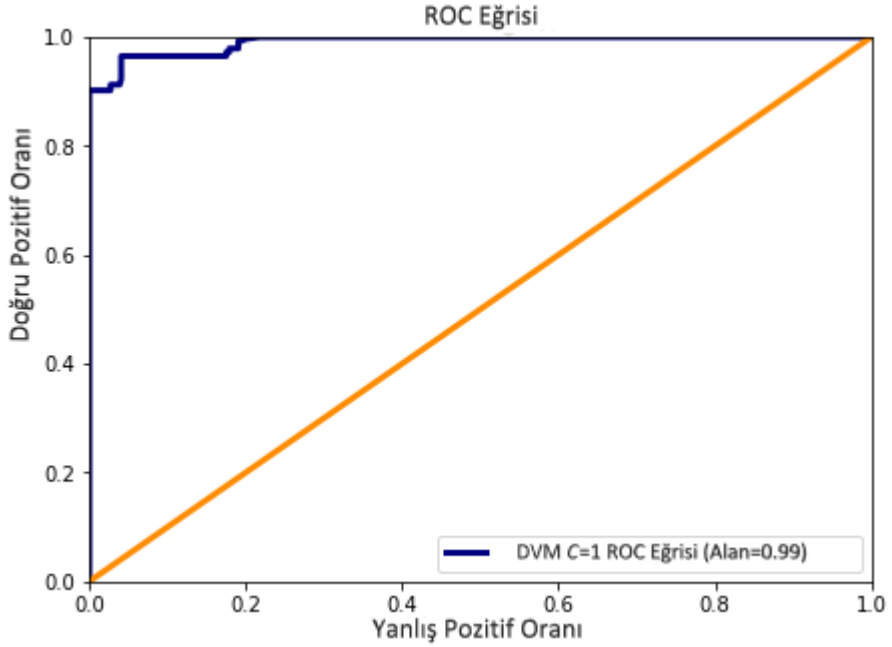
Performans ölçümü yapay sinir ağlarında önemli bir konudur. Bu nedenle, bir sınıflandırma probleminin performansının değerlendirilmesi için ROC eğrisi kullanılır. ROC eğrisi, farklı sınıflar için bir olasılık eğrisidir ve modelin ne kadar iyi tahmin yaptığını açıklar. Her iki senaryonun ROC eğrileri, DÖ, YSA, DVM *C*=1 ve *C*=100 için sırasıyla Şekil 3.5- 3.12'de verilmiştir.



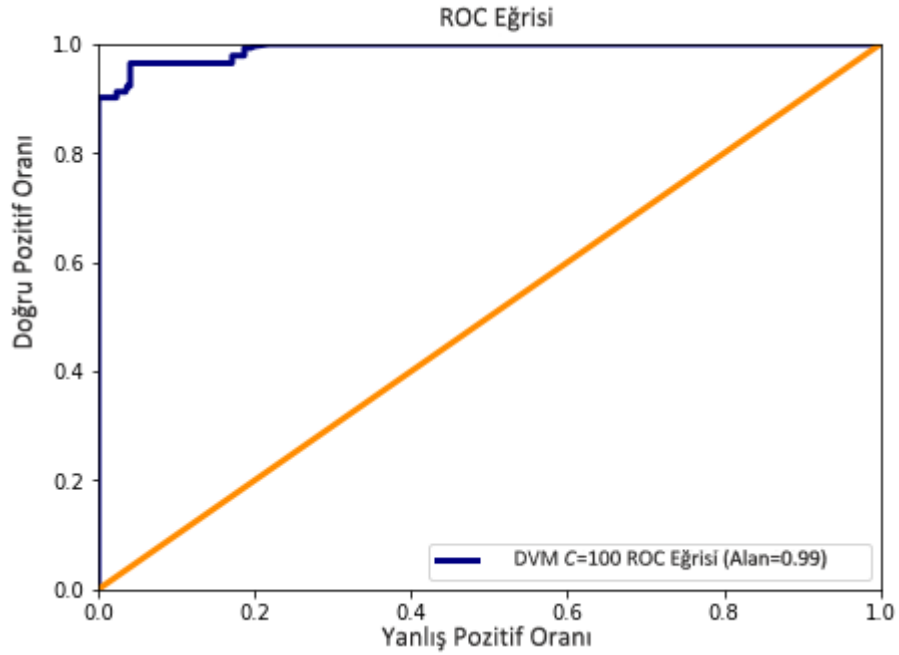

*Şekil 3.4.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda DÖ ROC Eğrisi



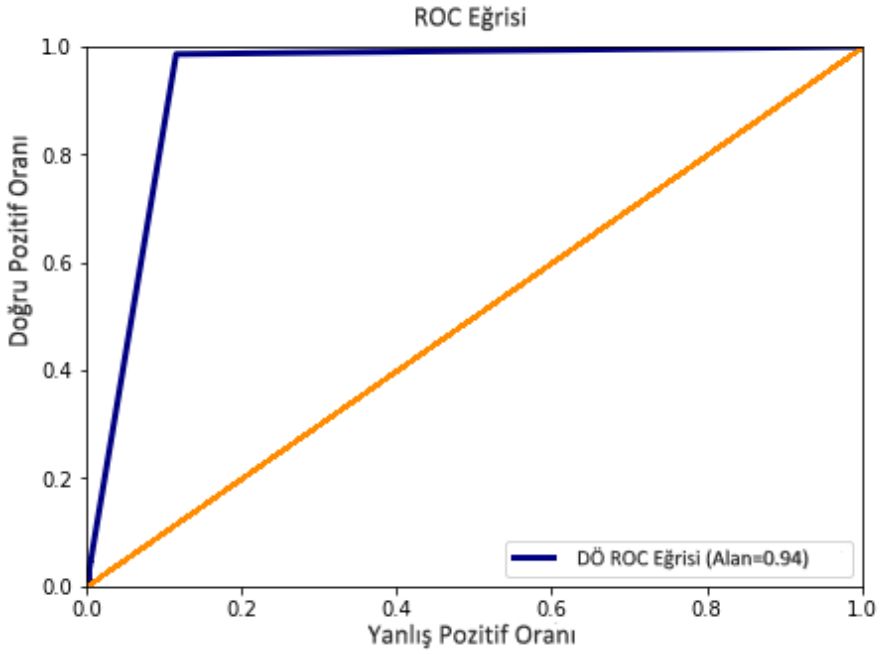

*Şekil 3.5.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda YSA ROC Eğrisi



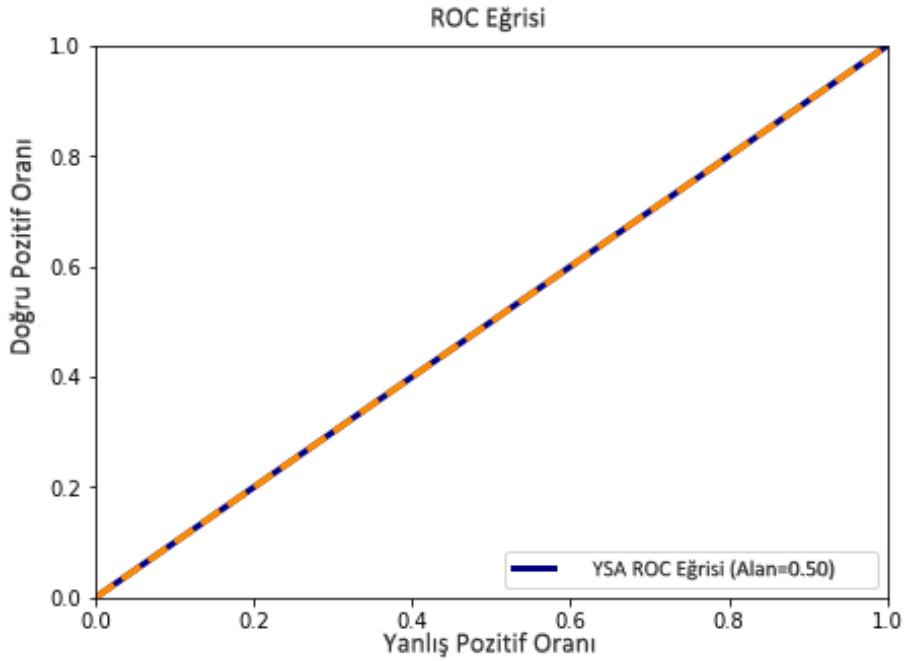

*Şekil 3.6.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda DVM *C*=1 ROC Eğrisi



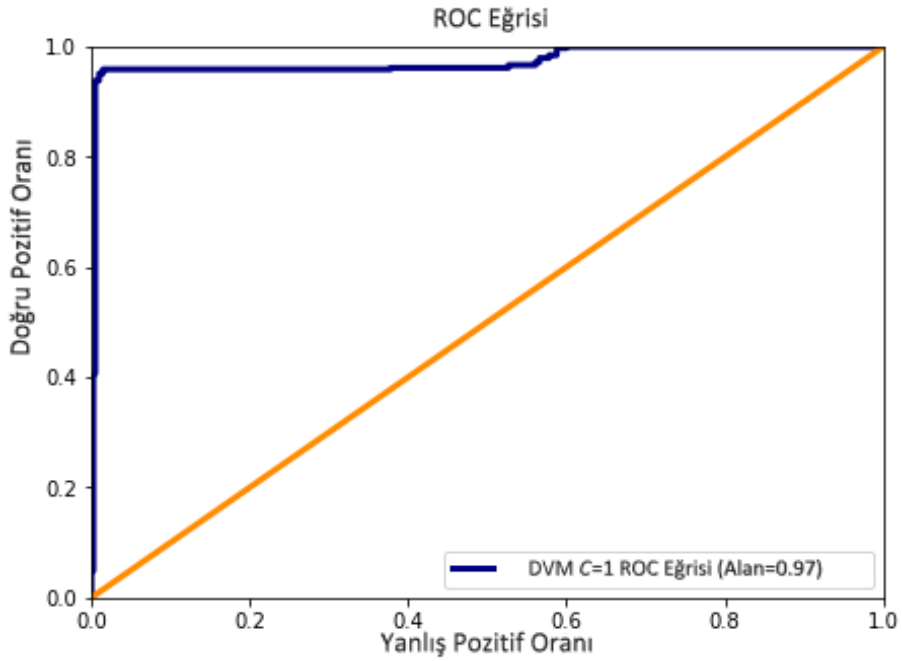

*Şekil 3.7.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İlk Senaryoda DVM *C*=100 ROC Eğrisi



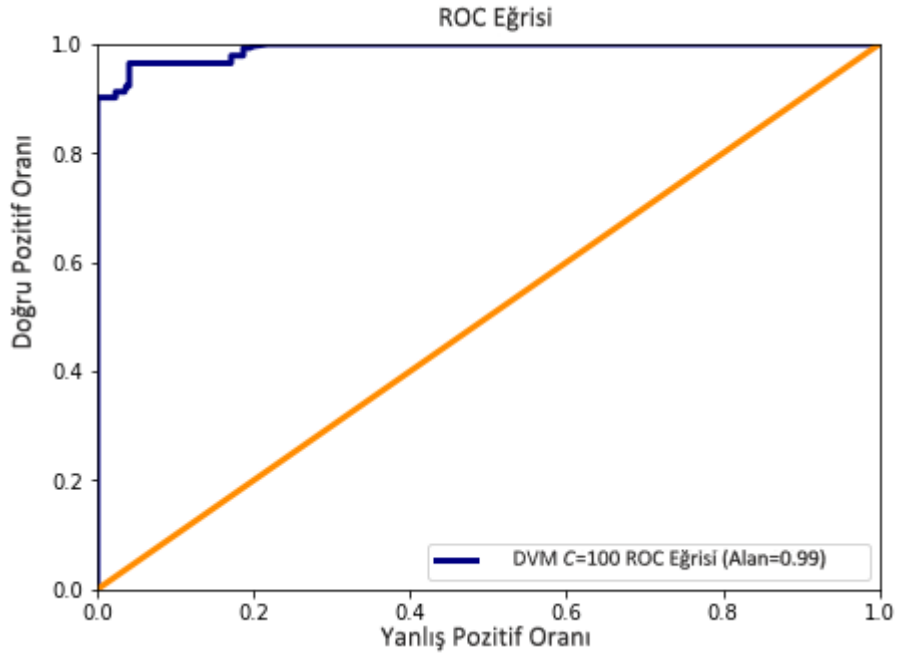

*Şekil 3.8.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryoda DÖ ROC Eğrisi




*Şekil 3.9.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryoda YSA ROC Eğrisi




*Şekil 3.10.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryoda DVM *C*=1 ROC Eğrisi



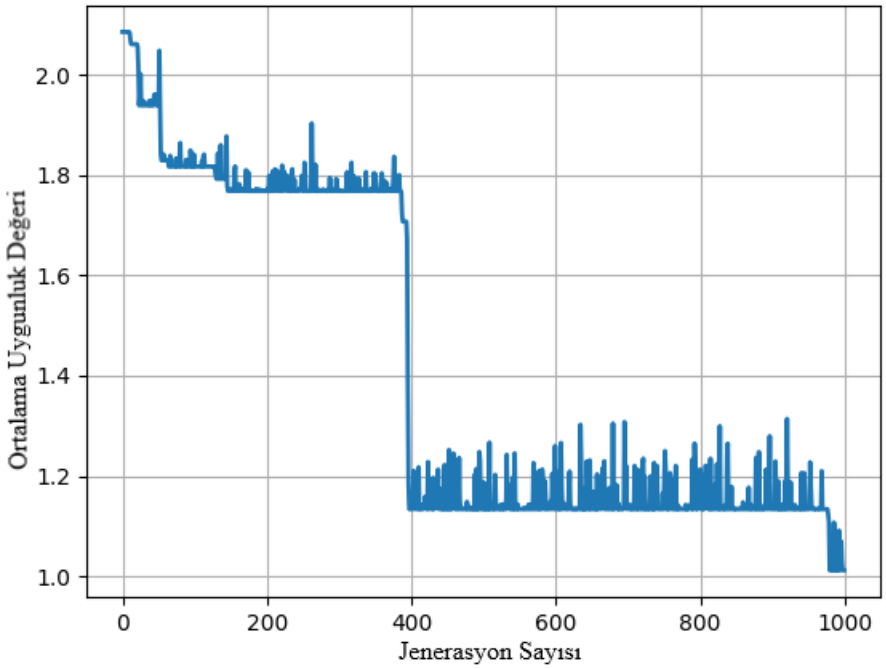

*Şekil 3.11.* YTA Tabanlı BVM'de Yük Dengeleme Problemi için Önerilen İkinci Senaryoda DVM *C*=100 ROC Eğrisi

## 3.2. YTA'da Kontrolör Yerleştirme Problemine Çözüm Üretilmesi

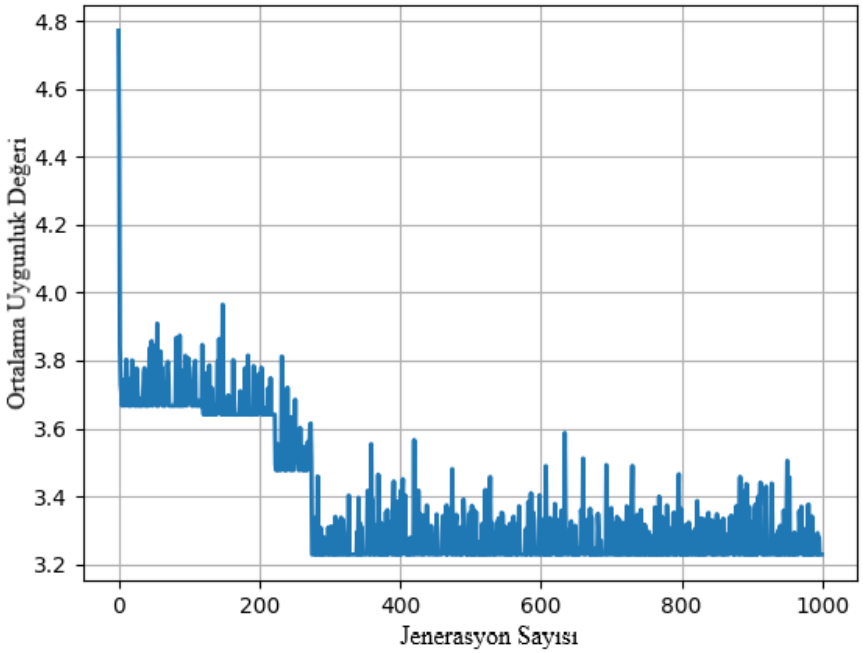
### 3.2.1. Genetik Algoritma Yaklaşımı

Bu yaklaşımda [78], tüm simülasyon deneyleri, Intel i7 işlemci ve 8 GB RAM'li bir bilgisayarda Ubuntu 16.04 işletim sistemi üzerinde gerçekleştirilmiştir. Çalışmanın kodları Python ile yazılmış ve derleyici olarak Pycharm platformundan yararlanılmıştır. GA'nın, KYP problemine uygulanabilmesi için popülasyondaki birey sayısı 100, ebeveyn yüzdesi 0.1, mutasyon oranı 0.05, rastgele seçim oranı 0.01 ve jenerasyon sayısı 1000 olarak belirlenmiştir. Çalışmada verilen haritaların birbirlerine olan bağlantılarının tutulması için bir matris oluşturulmuştur. Bu matrisin oluşturulmasında DA'dan yararlanılmıştır.

Her bir jenerasyon sonucunda sonucu optimum değere götüren yakınsama grafiği ULAKNET veri tabanı için Şekil 3.13'te, Colt veri tabanı için ise Şekil 3.14'te gösterilmiştir.




*Şekil 3.12.* KYP Problemine Genetik Algoritma Yaklaşımının ULAKNET Veri Tabanı için Yakınsama Grafiği

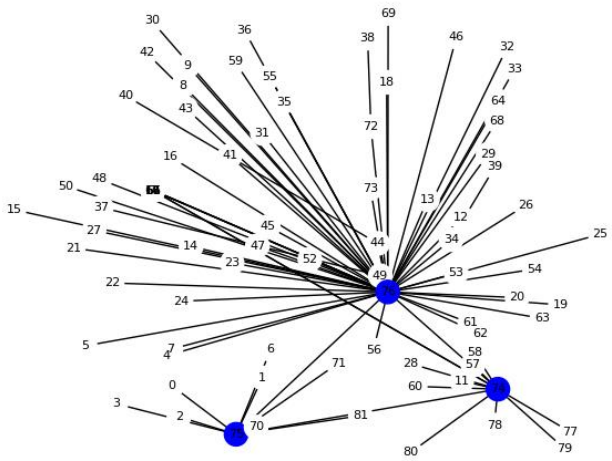



*Şekil 3.13.* KYP Problemine Genetik Algoritma Yaklaşımının Colt Veri Tabanı için Yakınsama Grafiği

1000 jenerasyon sonucunda GA kullanılarak elde edilen denetleyici yerleşim yerleri için elde edilen optimum değerler ULAKNET veri tabanı için sayısal olarak Şekil 3.15'te grafiksel olarak Şekil 3.16'da, Colt veri tabanı için sayısal olarak Şekil 3.17'de, grafiksel olarak Şekil 3.18'de verilmiştir.

```
Generation (980) ==> 1.0121951219512215
Generation (981) ==> 1.0121951219512215
Generation (982) ==> 1.0481707317073192
Generation (983) ==> 1.0121951219512215
Generation (984) ==> 1.0121951219512215
Generation (985) ==> 1.1073170731707338
Generation (986) ==> 1.0175609756097581
Generation (987) ==> 1.0121951219512215
Generation (988) ==> 1.0121951219512215
Generation (989) ==> 1.024146341463416
Generation (990) ==> 1.0121951219512215
Generation (991) ==> 1.0735365853658556
Generation (992) ==> 1.0913414634146361
Generation (993) ==> 1.0121951219512215
Generation (994) ==> 1.021951219512197
Generation (995) ==> 1.0697560975609774
Generation (996) ==> 1.0296341463414647
Generation (997) ==> 1.0121951219512215
Generation (998) ==> 1.0121951219512215
Generation (999) ==> 1.0121951219512215
[74, 75, 76] with a value of 1.0121951219512195
```

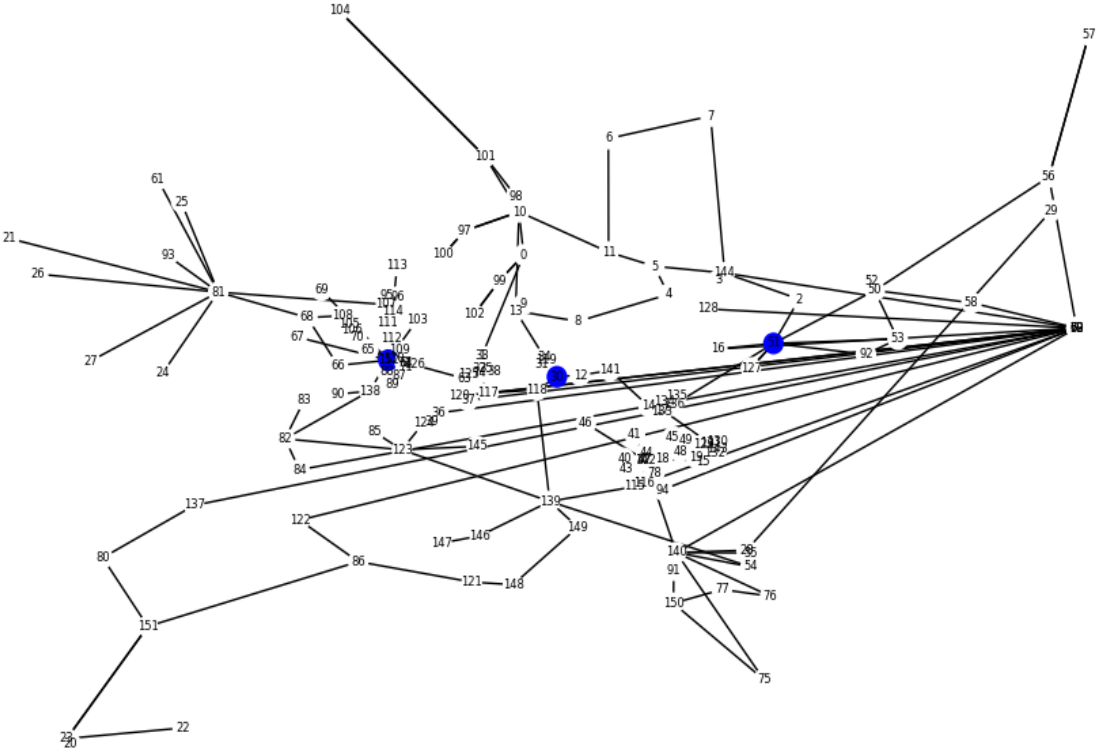
*Şekil 3.15.* KYP Problemine Genetik Algoritma Yaklaşımında ULAKNET Veri Tabanı için Denetleyici Yerleşim Yerlerinin Sayısal Değerleri



*Şekil 3.16.* KYP Problemine Genetik Algoritma Yaklaşımında ULAKNET Veri Tabanı için Denetleyici Yerleşim Yerlerinin Grafiksel Gösterimi

```
Generation (980) ==> 3.376405228758169
Generation (981) ==> 3.22875816993464
Generation (982) ==> 3.32797385620915
Generation (983) ==> 3.320915032679738
Generation (984) ==> 3.22875816993464
Generation (985) ==> 3.341176470588234
Generation (986) ==> 3.22875816993464
Generation (987) ==> 3.22875816993464
Generation (988) ==> 3.22875816993464
Generation (989) ==> 3.22875816993464
Generation (990) ==> 3.22875816993464
Generation (991) ==> 3.22875816993464
Generation (992) ==> 3.22875816993464
Generation (993) ==> 3.2918954248366017
Generation (994) ==> 3.22875816993464
Generation (995) ==> 3.278758169934639
Generation (996) ==> 3.22875816993464
Generation (997) ==> 3.22875816993464
Generation (998) ==> 3.22875816993464
Generation (999) ==> 3.22875816993464
[51, 30, 152] with a value of 3.2287581699346406
```

*Şekil 3.17.* KYP Problemine Genetik Algoritma Yaklaşımında Colt Veri Tabanı için Denetleyici Yerleşim Yerlerinin Sayısal Değerleri

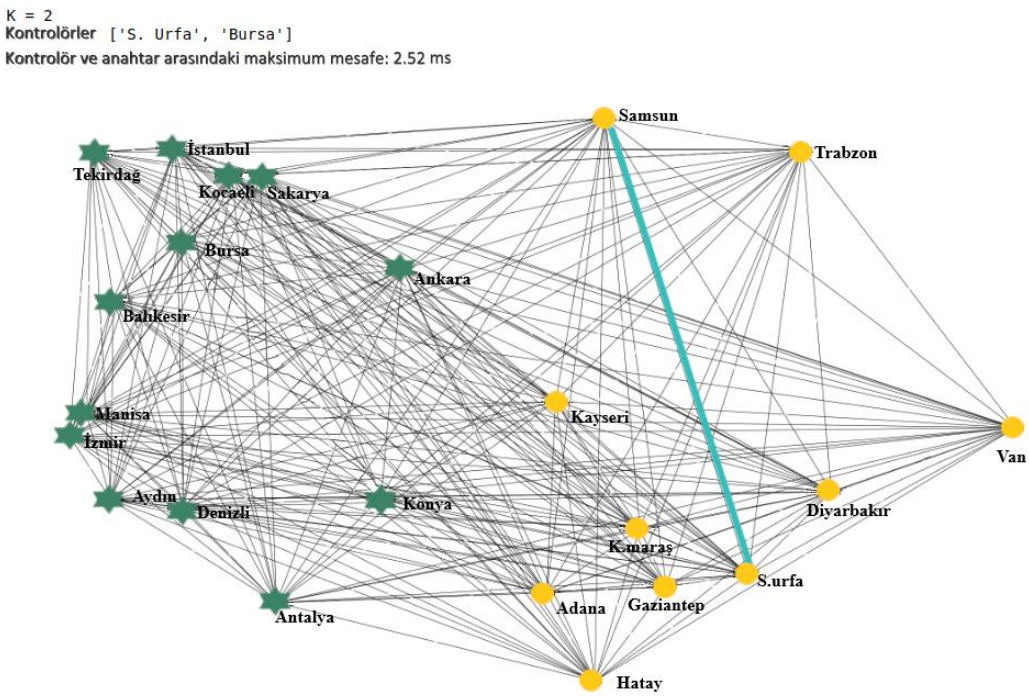


*Şekil 3.18.* KYP Problemine Genetik Algoritma Yaklaşımında Colt Veri Tabanı için Denetleyici Yerleşim Yerlerinin Grafiksel Gösterimi

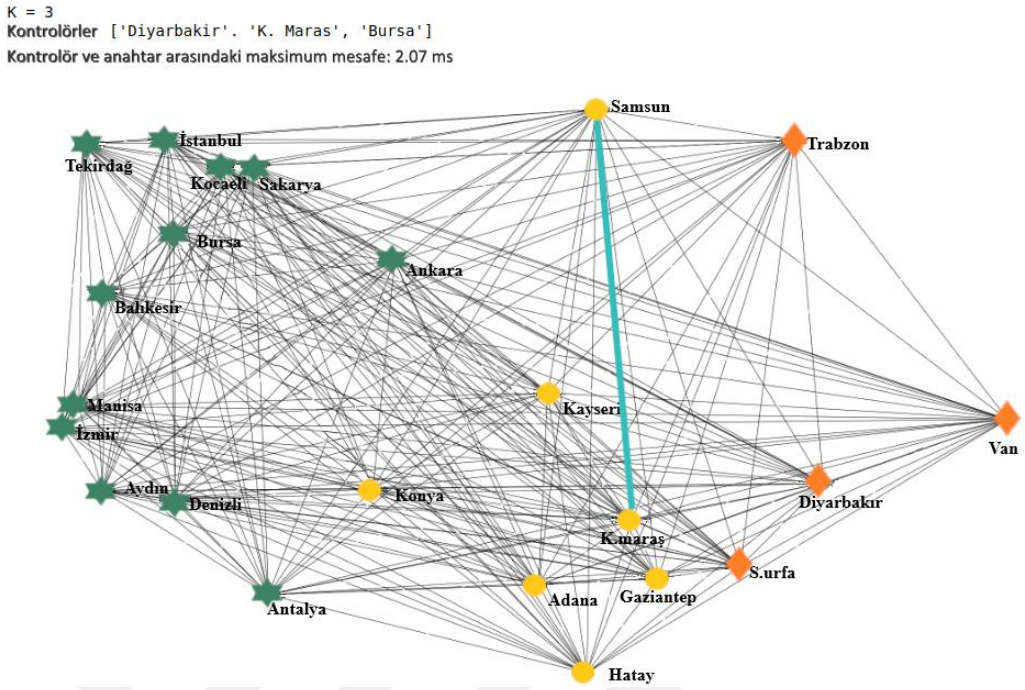
### 3.2.2. K-means ve Optimize Edilmiş K-means Algoritması Yaklaşımı

Bu yaklaşımda [79], tüm simülasyon deneyleri, Intel i7 işlemci ve 8 GB RAM'li bir bilgisayarda Ubuntu 16.04 işletim sistemi üzerinde gerçekleştirilmiştir. Çalışmanın kodları Python ile yazılmış ve derleyici olarak Jupyter Notebook platformundan yararlanılmıştır. YTA topolojisi bir mininet simülasyon ortamında oluşturulmuştur. YTA kontrolörü olarak Floodlight kullanılmıştır.

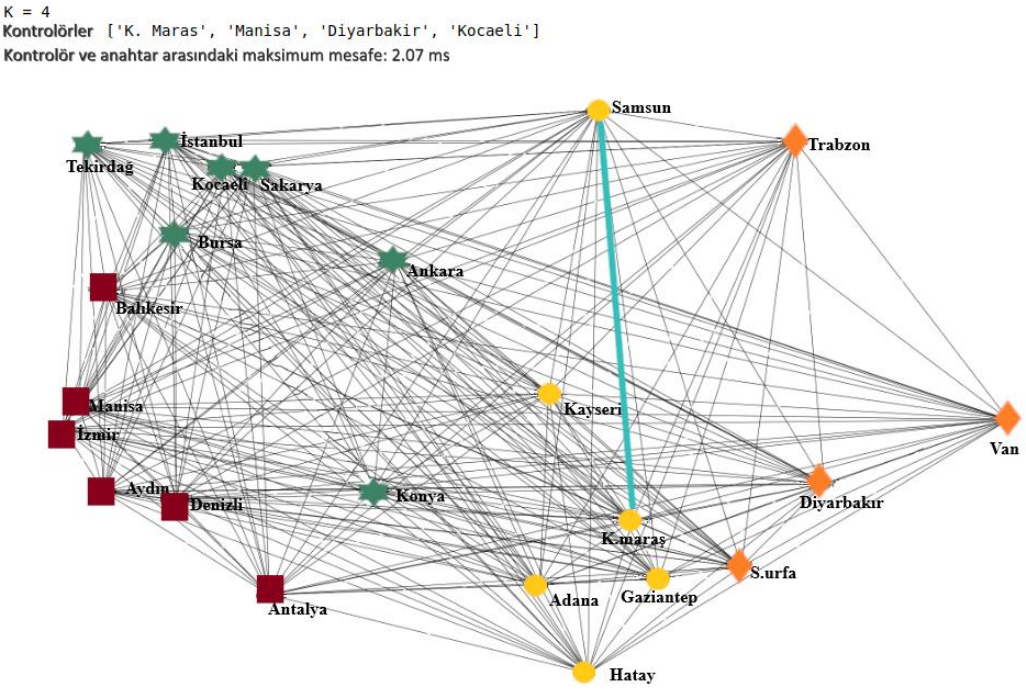
Bu yaklaşım kullanılarak elde edilen veriler grafik yöntemi ile haritalanmıştır. Şekil 3.19-3.24, standart k-means algoritması ile elde edilen sonuçları ve Şekil 3.25-3.29 ise, optimize edilmiş k-means algoritması ile elde edilen sonuçları göstermektedir. Şekillerde, kontrolör sayısı ve kontrolörler ile kontrolör ve anahtar arasındaki maksimum mesafe verilmiştir. Ayrıca denetleyicinin bulunduğu alan, diğer denetleyicilerden ayırt edilmesi için farklı bir renk ve şekilde verilmiştir.



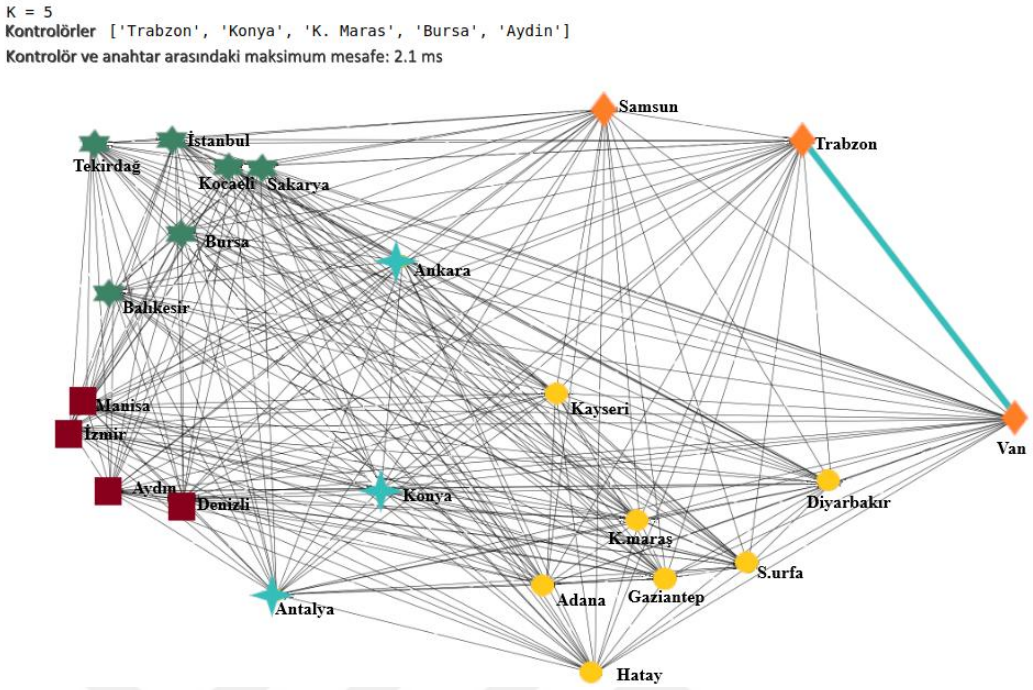

*Şekil 3.149.* KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=2)



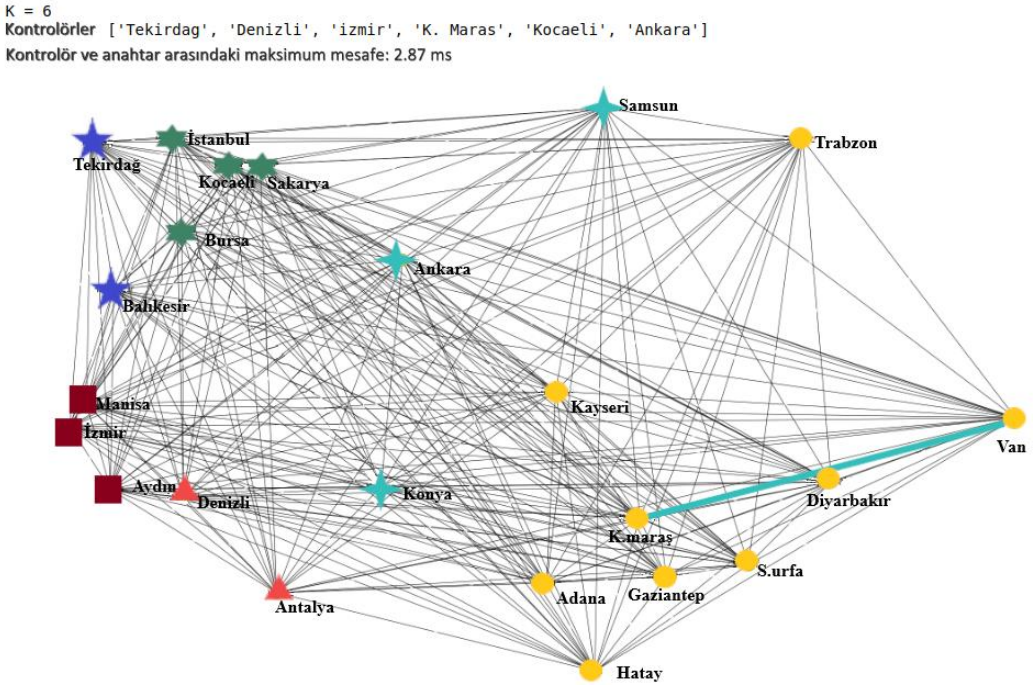

*Şekil 3.20.* KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=3)



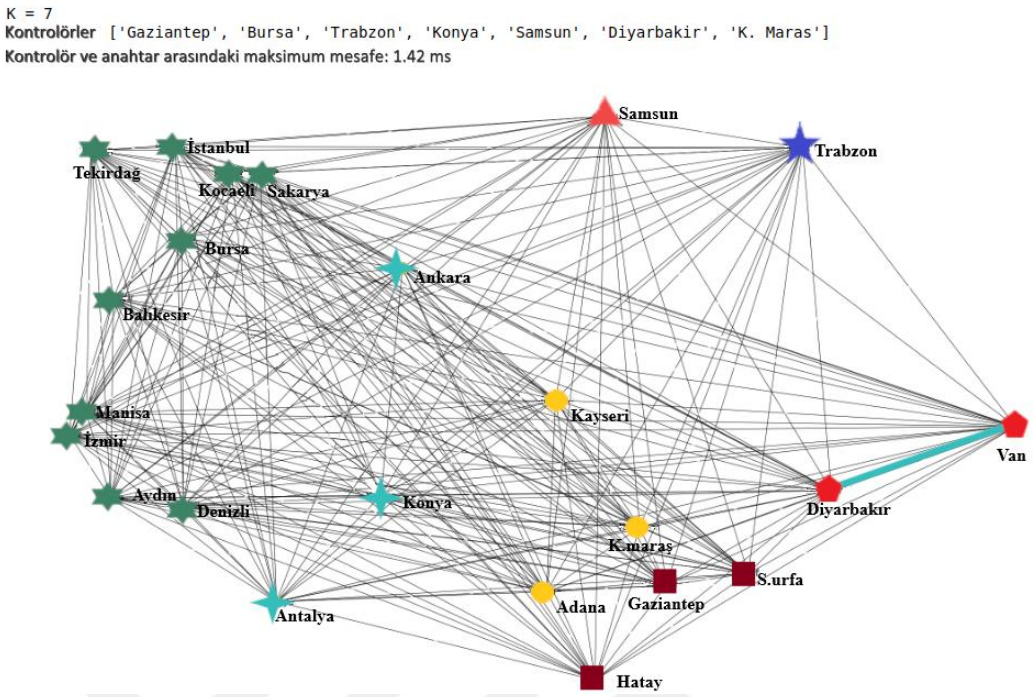

*Şekil 3.21.* KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=4)



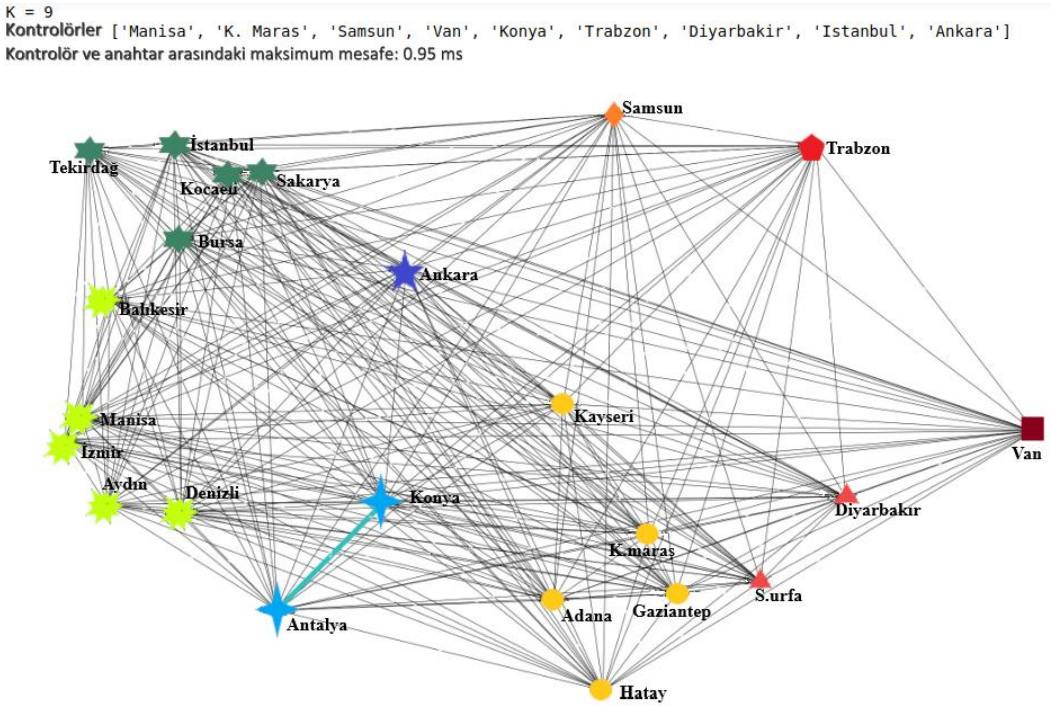

*Şekil 3.22.* KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=5)



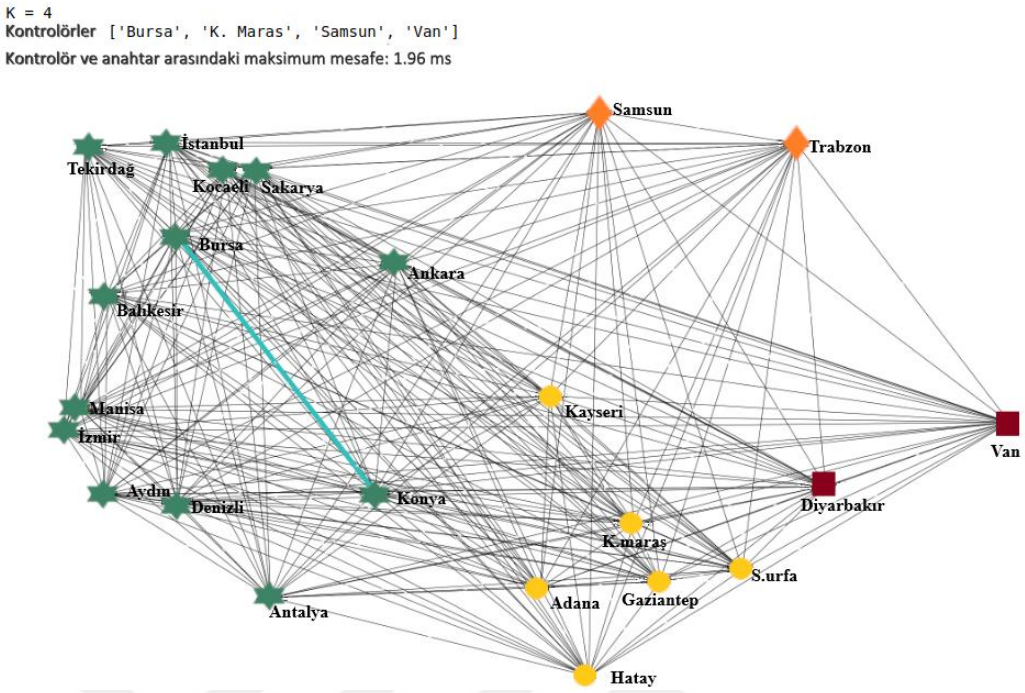

*Şekil 3.23.* KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=6)



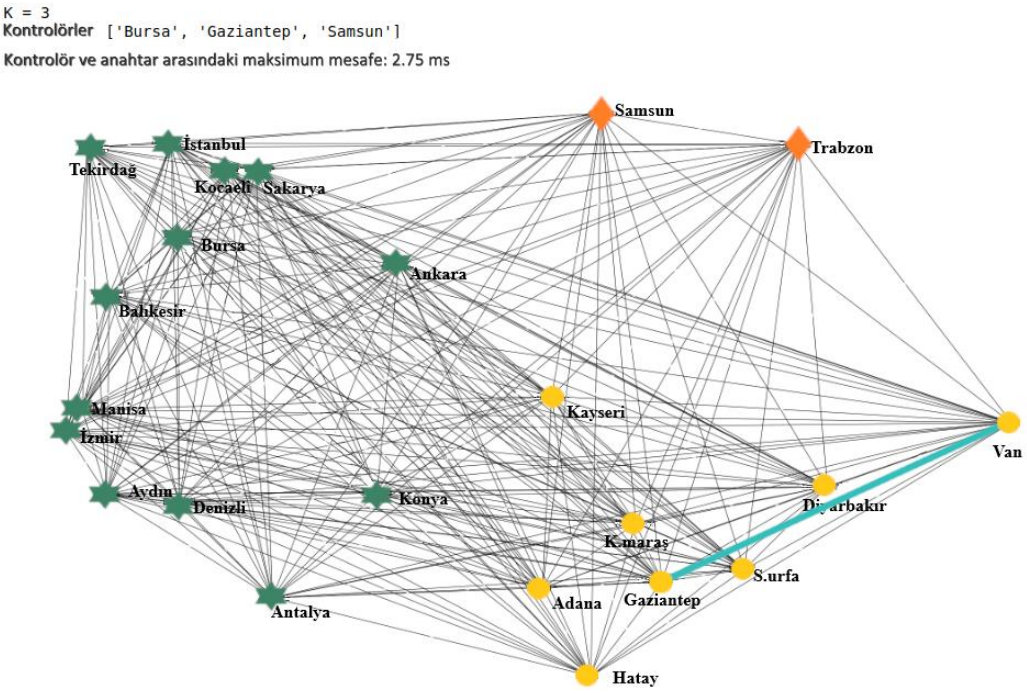

*Şekil 3.24.* KYP Problemine K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Kontrolör Sayısı=7)



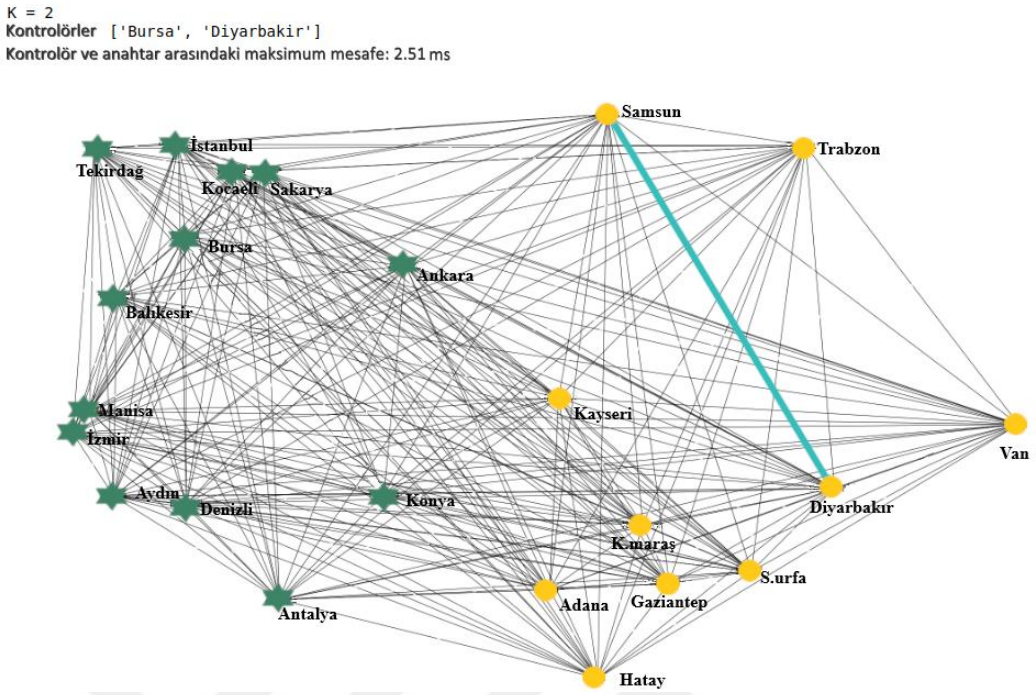

*Şekil 3.25.* KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=1 ms)



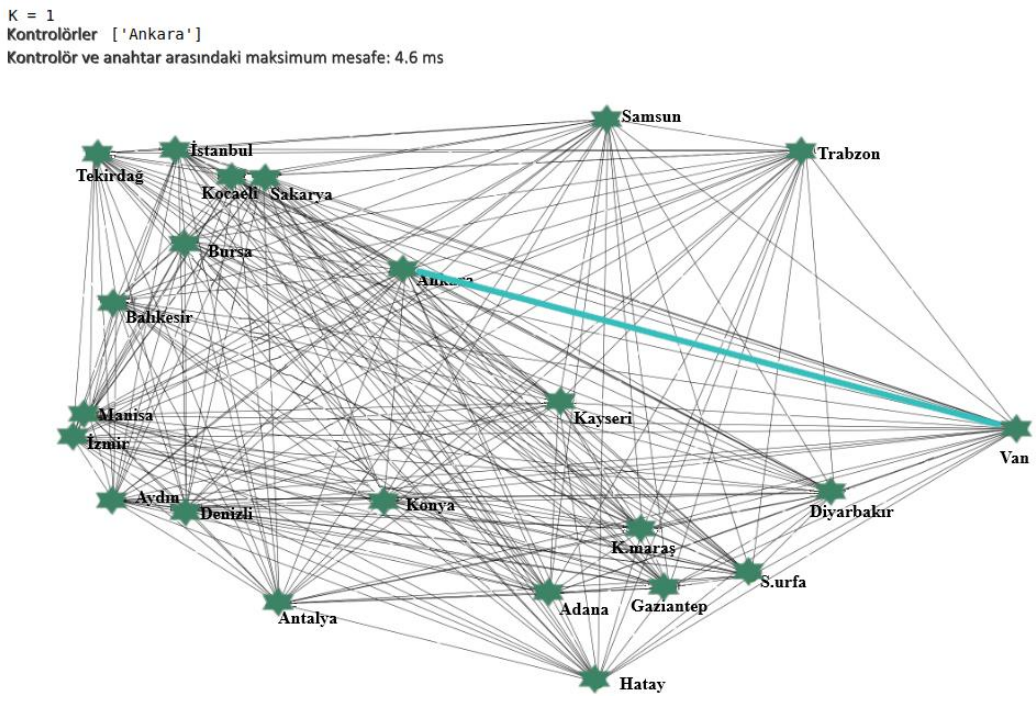

*Şekil 3.26.* KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=2 ms)




*Şekil 3.27.* KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=3 ms)



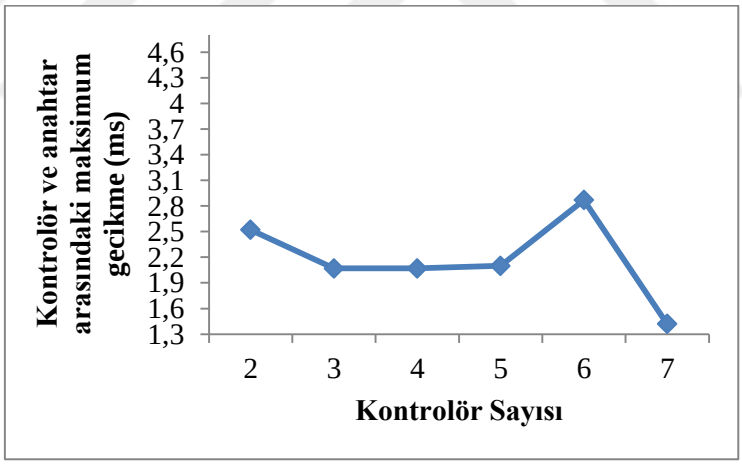

*Şekil 3.28.* KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=4 ms)



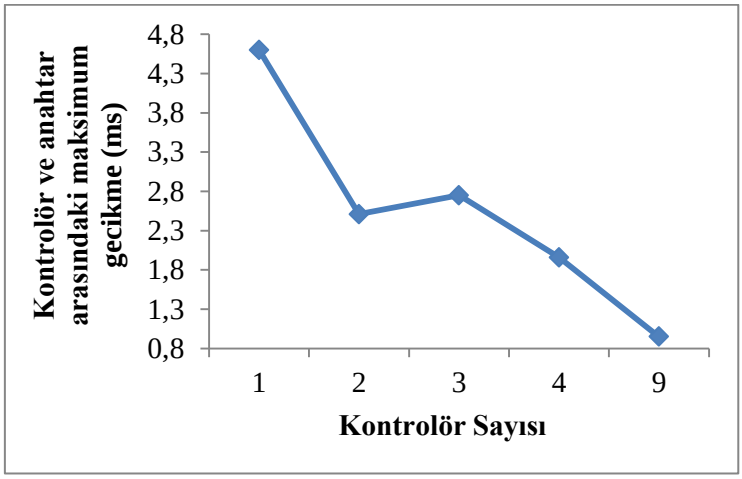

*Şekil 3.29.* KYP Problemine Optimize Edilmiş K-means Yaklaşımında Denetleyici Yerleşim Yerleri (Gecikme Süresi=5 ms)

Optimize edilmiş k-means algoritmasında *k* parametresi, algoritmadaki gecikmeyi sınırlandırıcı bir parametre olarak kullanıldığından kontrolör sayısı, maksimum gecikme süresine göre belirlenmiştir, ancak standart k-means algoritmasında *k* parametresi, algoritmadaki kontrolör sayısını sınırlandırıcı bir parametre olarak kullandığından kontrolör sayısı tarafımızca harici olarak belirlenmiştir.

Standart k-means algoritmasında kontrol düzlemi rastgele olacak şekilde belirlendiği için kontrolörler arasındaki gecikmeyle, kontrol sayısının değiştirilmesi arasında dengeli bir değişim söz konusu değildir. Bu değişim, Şekil 3.30'da verilmiştir. Ancak optimize edilmiş k-means algoritması yaklaşımında Şekil 3.31'de görüleceği gibi, kontrol düzlemi kontrolörler arasındaki mesafenin en az olacağı şekilde önce ikiye ve daha sonra alt ağlara bölünerek oluşturulduğu için, kontrolörler arasındaki gecikme ile kontrol sayısının değişmesi arasında daha dengeli bir değişim söz konusu olmuştur.




*Şekil 3.30.* KYP Problemine K-means Yaklaşımında Kontrolör Sayısı ile Maksimum Gecikme Arasındaki İlişki

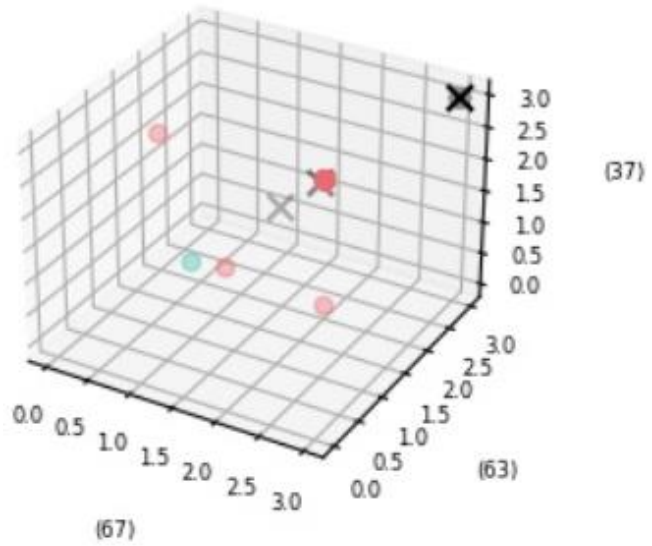



*Şekil 3.31.* KYP Problemine Optimize Edilmiş K-means Yaklaşımında Kontrolör Sayısı ile Maksimum Gecikme Arasındaki İlişki

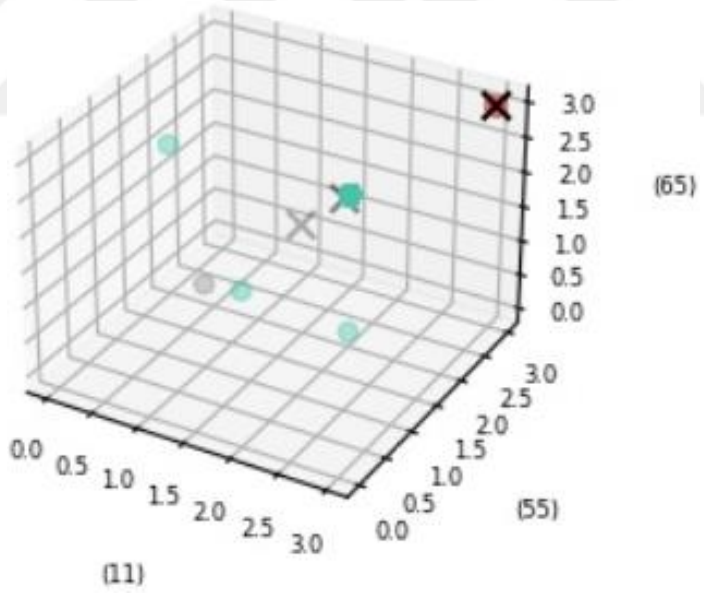
Optimize edilmiş k-means algoritmasında Şekil 3.31'de gösterildiği gibi, kontrolör sayısı ne kadar yüksek olursa, kontrolörler ile anahtarlar arasındaki ve kontrolörler ile kontrolörler arasındaki uçtan uca gecikme o kadar düşük olur sonucuna varılabilir.

### 3.2.3. YAK ve Hibrit-YAK Algoritması Yaklaşımı

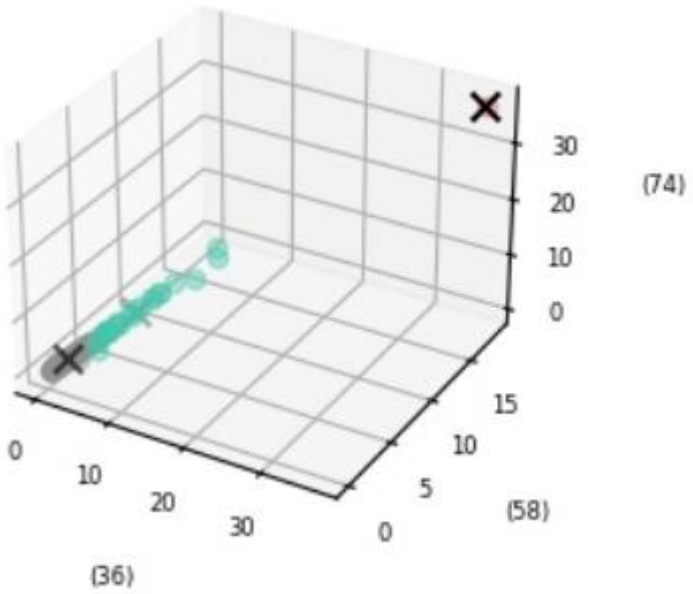
Bu yaklaşımda, tüm simülasyon deneyleri, Intel i7 işlemcili ve 8 GB RAM'li bir bilgisayarda Windows 10 işletim sistemi üzerinde gerçekleştirilmiştir. YAK ve Hibrit-YAK algoritması kodları Python dilinde yazılmış olup, Pycharm platformunda çalıştırılmıştır. YAK ve Hibrit-YAK algoritmaları için popülasyon sayısı; 100, limit değeri; 50, ve maksimum iterasyon sayısı; 50 olarak belirlendikten sonra hazırladığımız ilk ve ikinci veri setleri, algoritmalara verilmiş ve algoritmalar art arda 30 kez çalıştırılmıştır. Elde edilen verilerden en sık tekrarlayan kontrolör yerleri seçilmiştir. İlk harita sonucu Şekil 3.32 ve Şekil 3.33'te sırasıyla YAK ve Hibrit-YAK algoritmaları için verilmiştir. İkinci harita için ilgili sonuçlar Şekil 3.34 ve Şekil 3.35'te verilmiştir.



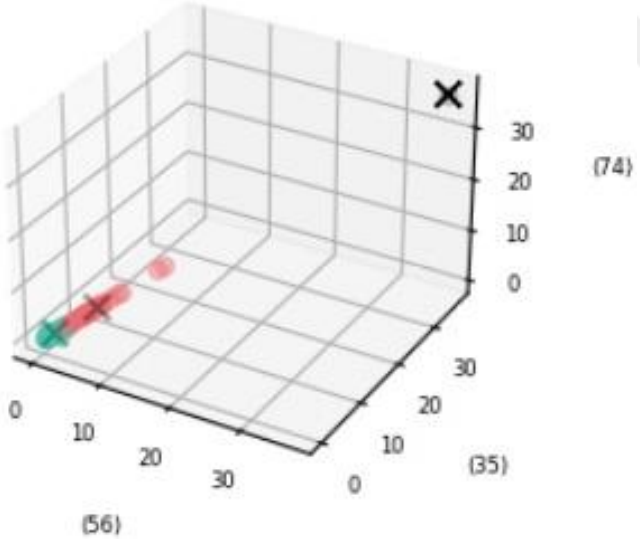

*Şekil 3.152.* KYP Problemine YAK Algoritması Yaklaşımında Denetleyici Yerleşim Yerleri




*Şekil 3.163.* KYP Problemine Hibrit-YAK Algoritması Yaklaşımında Denetleyici Yerleşim Yerleri




*Şekil 3.174.* KYP Problemine YAK Algoritması Yaklaşımında Ağırlıklandırılmış Denetleyici Yerleşim Yerleri




*Şekil 3.185.* KYP Problemine Hibrit-YAK Algoritması Yaklaşımında Ağırlıklandırılmış Denetleyici Yerleşim Yerleri

Tablo 3.7'de algoritmaların çözümlerinin kalitesi, standart sapma değerleri ve çalışma süreleri verilmiştir. Çözüm kalitesi kareler toplam hatası (KTH) ile ölçülmüştür.

Tablo 3.7. KYP Problemine YAK ve Hibrit-YAK Algoritması Yaklaşımında Algoritmaların Performansının Karşılaştırılması

| Veri Setleri | Algoritmalar | Çözüm Kalitesi (KTH) | Standart Sapma | Çalışma Süresi (s) |
|---|---|---|---|---|
| Veri Seti 1 | YAK | 228,7829 | 7,3100 | 242,80 |
| | Hibrit-YAK | 223,6031 | 2,9407 | 227,30 |
| Veri Seti 2 | YAK | 720,5150 | 7,2453 | 234,98 |
| | Hibrit-YAK | 715,7694 | 0,9549 | 233,95 |

Tablo 3.7'de görüldüğü gibi, k-means'in kullanılmasıyla veri setlerinde standart sapmanın azaldığı anlaşılmaktadır.

### 3.3. Tartışma

YTA tabanlı BVM'lerde yük dengelenmesinde, birinci ve ikinci senaryoların ortalama yanıt süre sonuçları incelendiğinde, YSA ve DÖ modellerinin diğer algoritmalara göre daha kısa süreye sahip olduğu görülmüştür. Ortalama yanıt süresi açısından, iki senaryo arasında en yüksek performansa sahip sınıflandırıcı YSA algoritmasıdır, ancak DÖ algoritması, YSA algoritmasına göre oldukça tutarlıdır. DÖ algoritmasını LR algoritması takip eder. DVM algoritması $C$=1 ve $C$=100 için yapılan analizde en düşük başarı performansına sahip sınıflandırıcı model olmuştur. Ayrıca, DVM $C$=100, $C$=1 ile karşılaştırıldığında daha iyi yanıt süresine sahip olduğu görülmüştür. Sınıflandırıcı sistemin doğruluk oranları açısından DVM $C$=100 en yüksek performansa sahip sınıflandırıcıdır. DVM $C$=100 ve DVM $C$=1'i ikinci sırada DÖ algoritması takip etmekte ve YSA algoritması en düşük başarı performansına sahip sınıflandırıcı model olmuştur. Hem ortalama tepki süresi hem de doğruluk oranları dikkate alındığında DÖ algoritması diğer algoritmalara göre oldukça tutarlıdır. Bu nedenle, YTA tabanlı BVM'lerde yük dengelenmesinde DÖ algoritması, diğer YZT algoritmaları ile karşılaştırıldığında en iyi sınıflandırıcı olarak kabul edilebilir.

YTA'da kontrolör yerleştirme problemine GA yaklaşımında, ULAKNET ve Colt olmak üzere farklı iki veri seti karşılaştırıldığında Colt veri setinde daha hızlı yakınsama olduğu görülmüştür. Colt veri setinde şehirlerin sayısı ve bağlantıların fazla olduğu göz önüne alındığında, büyük veri setinde yakınsamanın daha hızlı olduğu sonucuna

varılmıştır. Ancak kontrolörlerin optimum yerlerinin bulunması ve uçtan uca gecikmenin minimum seviyede olacak şekilde belirlenmesi konusunda, ULAKNET veri setinde GA algoritmasının performansının daha iyi olduğu anlaşılmıştır.

YTA'da kontrolör yerleştirme problemine k-means ve optimize edilmiş k-means yaklaşımında, sonuçlar incelendiğinde, optimize edilmiş k-means algoritmasının, standart k-means algoritmasına göre daha kararlı olduğu görülmüştür. İlk seçim ve sonraki seçimler, standart k-means algoritmasında rastgele seçildiğinden, her çalıştırma sonucunun çıktısı farklıdır. Ancak, optimize edilmiş k-means algoritması en kısa yolu bularak ilerler, böylece çalıştırılan tüm algoritmalarda çıktı aynı kalır. Ayrıca, standart k-means algoritmasında, kontrolör sayısı kullanıcı girişi olarak harici verilmelidir. Kontrolör sayısı KYP için çözülmesi gereken bir parametredir. Optimize edilmiş k-means algoritmasında, gecikme süresi belirlenerek en kısa mesafeler oluşturulmuş ve kontrolör sayısı optimum mesafe için belirlenmiştir. Böylece, KYP için uçtan uca gecikme optimal olarak belirlenmiştir.

YTA'da kontrolör yerleştirme problemine YAK ve Hibrit-YAK algoritması sonuçları karşılaştırıldığında görülmüştür ki, ilk veri seti ile algoritmalar anlamsız olarak en uygun noktaları bulurken, ikinci veri setinde algoritmalar, mantıklı bir dağılımla nüfus yoğunluklarının daha fazla olduğu bölgelerde kontrolör yerleri belirlemiştir. Ayrıca YAK algoritmasına göre Hibrit-YAK algoritmasının optimal çözüme yakınsamayı önemli ölçüde iyileştirdiği, çözüm kalitesini arttırdığı ve çalışma süresini azalttığı sonucuna varılmıştır.

# 4. BÖLÜM

# SONUÇ VE GELECEK ÇALIŞMALAR

## 4.1. Sonuç

Bu tez çalışmasında, yazılım tanımlı ağ (YTA) tabanlı büyük veri merkezleri (BVM) için, yük dengeleme problemi çeşitli yapay zekâ teknikleri ile değerlendirilmiş ve başarımı test edilmiştir. Ayrıca YTA'lar da çoklu kontrolör yerleştirme problemine (KYP) çözüm bulmak için farklı optimizasyon algoritmaları uygulanmış ve KYP için farklı iyileştirmeler gerçekleştirilmiştir.

İlk olarak bu tez çalışması ile YTA tabanlı BVM'lerin yük dengelemesi çalışması için literatürde ilk kez derin öğrenme (DÖ) algoritmasının kullanımı önerilmiştir. DÖ, BVM'lerde belirli yollar arasında değişken iletim maliyeti değerleri kullanılarak uygulanmıştır. DÖ tekniğinin etkinliği, yanıt süreleri ve doğruluk oranı sonuçları dikkate alınarak yapay sinir ağları, destek vektör makinesi ve lojistik regresyon gibi yapay zekâ teknikleri (YZT) ile karşılaştırılmıştır. Deneysel sonuçlar, önerilen DÖ tabanlı yük dengelemenin diğer YZT tabanlı yük dengeleme stratejilerinden daha iyi olduğunu göstermiştir.

İkinci olarak, YTA'daki KYP problemine çeşitli algoritma yaklaşımları önerilmiş ve KYP probleminin farklı unsurları üzerinde test edilmiştir. KYP problemine genetik algoritma (GA) yaklaşımında, KYP'ye Dijkstra algoritması (DA) tabanlı GA yaklaşımı ile çözüm üretilmiştir. GA'nın her bir çalışma zamanında en iyi çözümü bulma özelliğinden yararlanılarak uçtan uca gecikmeyi en aza indirecek bir sistem tasarlanmıştır. DA tabanlı GA yaklaşımı ULAKNET ve Colt veri setlerine uygulanmış ve elde edilen optimum çözümler grafla oluşturulan harita üzerinde gösterilmiştir. Ayrıca her iki veri tabanı için yakınsama grafikleri de verilmiştir.

KYP problemine k-means ve optimize edilmiş k-means yaklaşımında, KYP problemi için, kontrolörlerin optimal yerleşimini kontrolör sayısının yerleşime göre belirlemek

amacıyla optimize edilmiş k-means algoritması önerilmiş ve standart k-means algoritması ile karşılaştırılmıştır. Optimize edilmiş k-means algoritmasının, standart k-means algoritmasına göre daha kararlı olduğu görülmüştür. Ayrıca optimize edilmiş k-means algoritması uçtan uca gecikme süresini ve minimum kontrolör sayısını optimum olarak belirlemiştir.

KYP problemine yapay arı koloni (YAK) ve hibrit yapay arı koloni (Hibrit-YAK) yaklaşımında, YAK ve Hibrit-YAK algoritmaları Türkiye haritasında en optimum üç noktayı bulmak için kullanılmıştır. Ayrıca bu yaklaşımla ilk kez KYP için ağırlıklandırılmış bir harita kullanılmıştır, ilk kez KYP problemi YAK algoritması ile çözümlenmiş ve Hibrit bir YAK algoritması yaklaşımı önerilmiştir. Diğer yaklaşımlardan farklı olarak verilen haritalar nüfus yoğunluk oranlarına göre ağırlıklandırıldığı için kullanılan YAK ve Hibrit-YAK algoritmaları daha anlamlı sonuçlar vermiştir. Ayrıca, Hibrit-YAK algoritmasında standart sapmanın azaldığı, algoritmanın optimal çözüme yakınsamasının önemli ölçüde iyileştiği, çözüm kalitesinin arttığı ve çalışma süresini azaldığı görülmüştür.

### 4.2. Gelecek Çalışmalar

YTA tabanlı BVM'lerde yük dengeleme konusunda gelecek çalışma olarak, bu tez çalışmasında önerilen DÖ tabanlı yük dengeleme tekniği, bant genişliği, paket kaybı ve gecikme gibi farklı performans ölçütleri göz önünde bulundurularak YTA'daki tüm ağ topolojileri için incelenecektir. Gerçek zamanlı değerler elde edebileceğimiz deneysel bir test yatağı oluşturulacaktır. Daha sonra DÖ ve YZT teknikleri kullanılarak büyük veri merkezlerinde yük dengeleme yapılacaktır. Böylece DÖ'nün performansının doğrulanması için gerçek bir test ortamı oluşturulacaktır. Elde edilen veriler, en iyi yolu seçmek için DÖ ve YZT algoritmaları ile YTA kontrolörlerinde eğitilecektir. Yük dengelemesinin değerlendirilmesi için yanıt süresi ve doğruluk oranı parametrelerinden yararlanılacaktır.

# KAYNAKÇA


1. Xia, W., Zhao, P., Wen, Y., Xie, H., 2017. A survey on data center networking (DCN): infrastructure and operations. **IEEE Communications Surveys and Tutorials, 19**:640 – 656.

2. Khedkar, S. P., AroulCanessane, R., 2019. SDN enabled cloud, IoT and DCNs: A comprehensive survey. *5th International Conference On Computing, Communication, Control And Automation (ICCUBEA).*

3. Jain, S., Kumar, A., Mandal, S., Ong, J., Poutievski, L., Singh, A., Venkata, S., Wanderer, J., Zhou, J., Zhu, M., 2013. B4: experience with a globally-deployed software defined wan. **SIGCOMM Compututer Communication Review, 43**(4): 3-14.

4. Tavakoli, A., Casado, M., Koponen, T., Shenker, S., 2009. Applying NOX to the datacenter. *8th Workshop on Hot Topics in Networks ACM*.

5. Shirmarz, A., Ghaffari, A., 2020. Performance issues and solutions in SDN-based data center: a survey. **The Journal of Supercomputing, 76**:7545–7593.

6. Srishti, Singh, S., Deeban Chakravarthy, V., Kadiar, R., 2019. A survey on analysis of network traffic in data centers using controllers in SDN. **International Journal of Recent Technology and Engineering, 7** (6):1301–1304.

7. Hafeez, T., Ahmed, N., Ahmed, B., Malik, A.W., 2018. Detection and mitigation of congestion in SDN enabled data center networks: A survey. **IEEE Access, 6**:1730–1740.

8. Dai, B., Xu, G., Huang, B., Qin, P., Xu, Y., 2017. Enabling network innovation in data center networks with software defined networking: A survey. **Journal of Network and Computer Applications, 94**:33-49.

9. Xu, G., Yang, J., Dai, B., 2015. Challenges and opportunities on network resource management in DCN with SDN, 1785-1790. *IEEE International Conference on Big Data (Big Data)*, IEEE.

10. Marshoodulla, S.Z., Das, R.K., Saha, G., 2020. Big data issues in SDN based IoT: A review, 72-82. *International Conference on Big Data, Machine Learning, and Applications*, Springer.

11. Wang C., Zhang G., Xu H., Chen H., 2016. An ACO-based link load-balancing algorithm in SDN, 214–218. *7th International Conference on Cloud Computing and Big Data (CCBD).*

12. Stefano A.D., Cammarata G., Morana G., Zito D., 2015. A4SDN - adaptive alienated ant algorithm for software-defined networking, 344-350. *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC).*

13. Zhu R., Wang H., Gao Y., Yi S., Zhu F., 2015. Energy saving and load balancing for SDN based on multi-objective particle swarm optimization, 176-189. *Algorithms and Architectures for Parallel Processing,* Springer .

14. Al-Tam F., Correia N., 2019. On load balancing via switch migration in software-defined networking. **IEEE Access, 7**: 95998-96010.

15. Xue H., Kim K., Youn H., 2019. Dynamic load balancing of software-defined networking based on genetic-ant colony optimization. **Sensors, 19**(2): 311.

16. Prakash, S.W., Deepalakshmi, P., 2019. Artificial neural network based load balancing on software defined networking, 1-4. *IEEE International Conference on Intelligent Techniques in Control Optimization and Signal Processing (INCOS).*

17. Todorov, D., Valchanov, H., Aleksieva, V., 2020. Load balancing model based on machine learning and segment routing in SDN, 1-4. *International Conference Automatics and Informatics (ICAI).*

18. Fu, Q., Sun, E., Meng, K., Li, M., Zhang, Y., 2020. Deep Q-Learning for routing schemes in SDN-based data center networks. **IEEE Access, 8**:103491-103499.

19. Lei, K., Liang, Y., Li, W., 2020. Congestion control in SDN-based networks via multi-task deep reinforcement learning. **IEEE Network, 34**(4):28-34.

20. Babayigit B., Ulu B., 2018. Load balancing on software defined networks, 1-4. *2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT).*

21. Gebremariam, A.A. , Usman, M., Qaraqe, M., 2019. Applications of artificial intelligence and machine learning in the area of SDN and NFV: A survey, 545-549. *16th International Multi-Conference on Systems, Signals & Devices (SSD)*.

22. Latah, M., Toker, L., 2018. Artificial intelligence enabled software defined networking: A comprehensive overview, **CoRR**.

23. Latah, M., Toker, L., 2016. Application of artificial intelligence to software defined networking: A survey. **Indian Journal of Science and Technology, 9**(44):1-7.

24. Xie, J., Yu, F.R., Huang, T., Xie, R., Liu, J., Wang, C., Liu, Y., 2019. A survey of machine learning techniques applied to software defined networking (SDN): research issues and challenges. **IEEE Communications Surveys and Tutorials, 21**:393-430.

25. Alkhatib, A.A.A., Sawalha, T., Alzu'Bi, S., 2020. Load balancing techniques in software-defined cloud computing: An overview, 240-244. *Seventh International Conference on Software Defined Systems (SDS)*, IEEE.

26. Etengu, R., Tan, S.C., Kwang, L.C., Abbou, F.M., Chuah, T.C., 2020. AI-assisted framework for green-routing and load balancing in hybrid software-defined networking: proposal, challenges and future perspective. **IEEE Access, 8**:166384-166441.

27. Tamil Selvi, K., Thamilselvan, R., 2020. Deep learning based traffic classification in software defined networking –a survey. **International Journal of Scientific and Technology Research, 9**:2034–2041.

28. Mohammed, A.R., Mohammed, S.A., Shirmohammadi, S., 2019. Machine learning and deep learning based traffic classification and prediction in software defined networking, 1-6. *IEEE International Symposium on Measurements & Networking (M&N)*, IEEE.

29. Hamdan, M., Hassan, E., Abdelaziz, A., Elhigazi, A., Mohammed, B., Khan, S., Vasilakos, A.V., Marsono, M.N., 2021. A comprehensive survey of load balancing techniques in software-defined network. **Journal of Network and Computer Applications, 174**.

30. Belgaum, M.R., Musa, S., Alam, M.M., Su'Ud, M.M., 2020. A systematic review of load balancing techniques in software-defined networking. **IEEE Access, 8**:98612-98636.

31. Darade, S.A., Akkalakshmi, M., 2020. Extensive literature survey on load balancing in software-defined networking. **International Journal of Business Data Communications and Networking (IJBDCN), 16** (2):1-19.

32. Semong, T., Maupong, T., Anokye, S., Kehulakae, K., Dimakatso, S., Boipelo, G., Sarefo, S., 2020. Intelligent load balancing techniques in software defined networks: A survey. **Electronics, 9**(7):1-24.

33. Babbar, H., Rani, S., 2019. Emerging prospects and trends in software defined networking. **Journal of Computational and Theoretical Nanoscience, 16**(10):4236-4241.

34. Neghabi, A.A., Navimipour, N.J., Hosseinzadeh, M., Rezaee, A., 2018. Load balancing mechanisms in the software defined networks: A systematic and comprehensive review of the literature. **IEEE Access, 6**:14159-14178.

35. Bhatia, J., Mehta, R., Bhavsar, M., 2018. Variants of software defined network (SDN) based load balancing in cloud computing: A quick review, 164-173. *International Conference on Future Internet Technologies and Trends*, Springer.

36. Li, L., Xu, Q., 2017. Load balancing researches in SDN: A survey, 403-408. *7th IEEE International Conference on Electronics Information and Emergency Communication (ICEIEC)*, IEEE.

37. Raghul, S., Subashri, T., Vimal, K.R., 2017. Literature survey on traffic-based server load balancing using SDN and open flow, 1-6. *Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*, IEEE.

38. Wang, G., Lu, G., Jia, W., Jia, C., Zhang, J., 2020. A Review on the application of machine learning in SDN routing optimization. **Journal of Computer Research and Development, 57**(4): 688-698.

39. Zhao, Y., Li, Y., Zhang, X., Geng, G., Zhang, W., Sun, Y., 2019. A survey of networking applications applying the software defined networking concept based on machine learning. **IEEE Access, 7**:95397-95417.

40. Nguyen, T.N., 2019. The challenges in ML-based security for SDN. *2nd Cyber Security in Networking Conference (CSNet)*, IEEE.

41. Herrera, J., Camargo, J., 2019. A survey on machine learning applications for software defined network security, 70-93. *International Conference on Applied Cryptography and Network Security*, Springer.

42. Nguyen, T.N., The challenges in SDN/ML based network security : A survey. **CoRR**.

43. Sultana, N., Chilamkurti, N., Peng, W., Alhadad, R., 2018. Survey on SDN based network intrusion detection system using machine learning approaches. **Peer-to-Peer Networking and Applications**, 12:1-9.

44. Alsaeedi, M., Mohamad, M.M., Al-Roubaiey, A.A., 2019. Toward adaptive and scalable OpenFlow-SDN flow control: A survey. **IEEE Access, 7**:107346-107379.

45. Ahmadi V., Khorramizadeh M., 2018. An adaptive heuristic for multiobjective controller placement in software-defined networks. **Computers and Electrical Engineering, 66**: 204-228.

46. Huang V., Chen G., Fu Q., Wen E., 2019. Optimizing controller placement for software-defined networks, 224-232. *Symposium on Integrated Network and Service Management (IM)*, IEEE.

47. Sanner J., Hadjadj-Aoul Y., Ouzzif M., Rubino G., 2017. An evolutionary controllers' placement algorithm for reliable SDN networks, 1-6. *13th International Conference on Network and Service Management (CNSM)*, IEEE.

48. Liao L., Leung V.C., 2017. Genetic algorithms with particle swarm optimization-based mutation for distributed controller placement in SDNs, 1-6. *Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, IEEE.

49. Sahoo K.S., Sahoo S., Sarkar A., Sahoo B., Dash R., 2017. On the placement of controllers for designing a wide area software defined networks, 3123-3128. *Region 10 Conference*, IEEE.

50. Wang G., Zhao Y., Huang J., Duan Q., Li J., 2016. A k-means-based network partition algorithm for controller placement in software defined network, 1-6. *Communications Software, Services and Multimedia Applications Symposium (ICC)*, IEEE.

51. Das, T., Sridharan, V., Gurusamy, M., 2019. A survey on controller placement in SDN. **IEEE Communications Surveys and Tutorials, 22**:472-503.

52. Zhang, Y., Cui, L., Wang, W., Zhang, Y., 2018. A survey on software defined networking with multiple controllers. **Journal of Network and Computer Applications, 103**:101-118.

53. Hu, T., Guo, Z., Yi, P., Baker, T., Lan, J., 2018. Multi-controller based software-defined networking: A survey. **IEEE Access, 6**:15980-15996.

54. Lu, J., Zhang, Z., Hu, T., Yi, P., Lan, J., 2019. A survey of controller placement problem in software-defined networking. **IEEE Access, 7**:24290–24307.

55. Isong, B., Molose, R.R.S., Abu-Mahfouz, A.M., Dladlu, N., 2020. Comprehensive review of SDN controller placement strategies. **IEEE Access, 8**:170070-170092.

56. Kumari, A., Sairam, A.S., 2019. A survey of controller placement problem in software defined networks. **CoRR**.

57. Killi, B.P.R., Rao, S.V., 2019. Controller placement in software defined networks: A comprehensive survey. **Compututer Networks, 163**.

58. Choumas, K., Giatsios, D., Flegkas, P., Korakis, T., 2019. The SDN control plane challenge for minimum control traffic: distributed or centralized, *16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, IEEE.

59. Adebayo, I.O., Adigun, M.O., Mudali, P., 2018. Feature selection strategies for the controller placement problem in SDNs: A review, 1-5. *International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, IEEE.

60. Singh, A.K., Srivastava, S., 2018. A survey and classification of controller placement problem in SDN. **International Journal of Network Management, 28**(3).

61. Wang, G., Zhao, Y., Huang, J., Wang, W., 2017. The controller placement problem in software defined networking: A survey, **IEEE Network, 31**(5):21–27.

62. Gao, X.M., Wang, B.S., Deng, W.P., Tao, J., 2017. Survey of controller placement problem in software defined network. **Journal on Communications, 38** (7):155-164.

63. Yoon, S.K., Khalib, Z., Yaakob, N., Amir, A., 2017. Controller placement algorithms in software defined network - a review of trends and challenges. *MATEC Web Conferences*.

64. Yu, T., Hong, Y., Cui, H., Jiang, H., 2018. A survey of multi-controllers consistency on SDN,1-6. *4th International Conference on Universal Village (UV)*, IEEE.

65. Babayigit, B., Ulu, B., 2021. Deep learning for load balancing of SDN based data center networks. **International Journal of Communication System, 34**(7).

66. McKeown, N., 2009. Software-defined networking. **INFOCOM Keynote Talk, 17**:30–32.

67. Lopes, F.A., Santos, M., Fidalgo, R., Fernandes, S., 2016. A software engineering perspective on SDN programmability. **IEEE Communications Surveys and Tutorials, 18**:1255-1272.

68. Xia, W., Wen, Y., Foh, C.H., Niyato, D., Xie, H., 2015. A survey on software-defined networking. **IEEE Communications Surveys and Tutorials, 17**:27–51.

69. Kreutz, D., Ramos, F.M.V., Veríssimo, P.E., Rothenberg, C.E., Azodolmolky, S., Uhlig, S., 2015. Software-defined networking: A comprehensive survey. **Proceedings of the IEEE, 103**:14–76.

70. Hangloo, S., Kumar, S., 2018. Software-defined networking: A survey. **Computer Networks, 4**:282-290.

71. Trois, C., Del Fabro, M.D., de Bona, L.C.E., Martinello, M., 2016. A survey on SDN programming languages: toward a taxonomy. **IEEE Communications Surveys and Tutorials, 18**:2687-2712.

72. Anonim, 2013. An instant virtual network on your laptop, Mininet. (http://mininet.org/), (Erişim Tarihi: 10 Mart 2021).

73. Anonim. Network Simulator, NS3. (https://www.nsnam.org/), (Erişim Tarihi: 10 Haziran 2021).

74. Büyük veri. Büyük veri nedir. (https://tr.wikipedia.org/wiki/Buyuk_veri), (Erişim Tarihi: 10 Haziran 2021).

75. Akhtar N., Mian A., 2018. Threat of adversarial attacks on deep learning in computer vision: A survey. **IEEE Access, 6**:14410-14430.

76. Zhang C., Patras P., Haddadi H.,2019. Deep learning in mobile and wireless networking: A survey. **IEEE Communications Surveys Tutorials, 21**(3):2224-2287.

77. Otter D.W., Medina J.R., Kalita J.K., 2020. A survey of the usages of deep learning for natural language processing. **IEEE Transactions on Neural Networks and Learning Systems**, 1-21.

78. MacQueen J.B., 1967. Some methods for classification and analysis of multivariate observations, 281-297. *5-th Berkeley Symposium on Mathematical Statistics and Probability, University of California Press*.

79. Karaboga, D., 2005. An idea based on honey bee swarm for numerical optimization, technical report. Erciyes University, Engineering Faculty,Computer Engineering Department.

80. Anonim, 2010. TopologyZoo-Ulaknet veri seti. (http://www.topology-zoo.org/files/Ulaknet.gml.), (Erişim Tarihi: 10 Mart 2021).

# ÖZGEÇMİŞ

## KİŞİSEL BİLGİLER

**Adı Soyadı:** Banu ULU

**Uyruğu:** Türkiye (T.C)

## EĞİTİM

| Derece | Kurum | Mezuniyet Tarihi |
|---|---|---|
| Yüksek Lisans | Erciyes Üniversitesi. Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği ABD, KAYSERİ | 2016 |
| Lisans | Erciyes Üniversitesi, Bilgisayar Mühendisliği Bölümü, KAYSERİ | 2013 |
| Lise | Nuh Mehmet Küçükçalık Anadolu Lisesi, KAYSERİ | 2008 |

## YABANCI DİL

İngilizce

## YAYINLAR

### SCI, SSCI, AHCI ˙Indekslerine Giren Dergilerde Yayımlanan Makaleler

1. Babayiğit, B., Ulu, B., 2021. Deep learning for load balancing of SDN-based data center networks, **International Journal of Communication Systems, 34**(7).

**SCI, SSCI, AHCI ˙Indekslerine Giren Dergilerde İncelemede Olan Makaleler**

1. Babayiğit, B., Ulu, B., 2021. A taxonomy of survey studies on software-defined networking. *İncelemede*.

2. Babayiğit, B., Ulu, B., 2021. A new modified ABC clustering algorithm for capacitated controller placement in SDNs. *İncelemede*.

**TR Dizinli Dergilerde Yayımlanan Makaleler**

1. Babayiğit, B., Ulu, B., 2021. A high available multi-controller structure for SDN and placement of multi-controllers of SDN with optimized k-means algorithm. Iğdır Üniversitesi Fen Bilimleri Enstitüsü Dergisi. *Yayın aşamasında*.

**Hakemli Kongre / Sempozyum Bildiri Kitaplarında Yer Alan Yayınlar**

1. Babayiğit, B., Ulu, B., Hoşçokadar, E., 2019 . Solving multi-controller placement problem in software defined networks with a genetic algorithm, Tam Metin Bildiri. *2019 4th International Conference on Computer Science and Engineering (UBMK)*, 11 Eylül 2019, 15 Eylül 2019.

2. Babayiğit, B., Ulu, B., 2018. Load balancing on software defined networks, Tam Metin Bildiri. *2nd International Symposium on Multidisciplinary Studies and Innovative Technologies*, ISMSIT 2018, 19 Ekim 2018, 21 Ekim 2018.

3. Babayiğit, B., Karakaya, S., Ulu, B., 2018. Raspberry Pi ile bir yazlm tabanlı ağ gerçekleştirilmesi, Tam Metin Bildiri. *26. Ieee Sinyal Işleme Ve Iletişim Uygulamaları Kurultayı,* 02 Mayıs 2018, 05 Mayıs 2018.

4. Babayiğit, B., Ulu, B., 2017., Nesnelerin interneti ve büyük veri mimari ve teknolojileri, Özet Bildiri. *International Mediterranean Science And Engineering Congress (imsec 2017)*, 25 Ekim 2017, 27 Ekim 2017.