BİYOLOJİK VERİTABANLARINDA ETKİN BENZERLİK HESAPLAMA

ARDA SÖYLEV

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

TEMMUZ 2013
ANKARA

Fen Bilimleri Enstitü onayı

_____

Prof. Dr. Necip CAMUŞCU
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

_____

Doç. Dr. Erdoğan DOĞDU
Anabilim Dalı Başkanı

Arda Söylev tarafından hazırlanan BİYOLOJİK VERİTABANLARINDA ETKİN BENZERLİK HESAPLAMA adlı bu tezin Yüksek Lisans tezi olarak uygun olduğunu onaylarım.

_____

Doç. Dr. Osman ABUL
Tez Danışmanı

Tez Jüri Üyeleri

Başkan :Yrd. Doç. Dr. Mehmet TAN _____

Üye : Doç. Dr. Osman ABUL _____

Üye : Yrd. Doç. Dr. Ersin Emre ÖREN _____

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, ayrıca tez yazım kurallarına uygun olarak hazırlanan bu çalışmada orijinal olmayan her türlü kaynağa eksiksiz atıf yapıldığını bildiririm.

Arda SÖYLEV

Üniversitesi : **TOBB Ekonomi ve Teknoloji Üniversitesi**
Enstitüsü : **Fen Bilimleri**
Anabilim Dalı : **Bilgisayar Mühendisliği**
Tez Danışmanı : **Doç. Dr. Osman ABUL**
Tez Türü ve Tarihi : **Yüksek Lisans – Temmuz 2013**

Arda SÖYLEV

BİYOLOJİK VERİTABANLARINDA ETKİN BENZERLİK HESAPLAMA

ÖZET

Canlının temel özelliklerini taşıyan en küçük birim olan hücrenin içerisinde meydana gelen olayların açıklanması biyolojik ağlarının incelenmesiyle mümkün olur. Bu inceleme için kullanılan tekniklerden biri benzerlik tabanlı analizdir. Bu kapsamda, bir sorgu ağıyla biyolojik ağlardan oluşan bir biyolojik veritabanı karşılaştırılmakta, sorgu ağıyla benzerliği belli bir eşik değerinin üzerinde ve aşağısında olan ağlar ayrışmaktadır. Bu problemin çözümü, iki ağın benzerliğinin bulunmasını gerektirir. Literatürde NP-tam olarak geçen alt çizge eşleniği problemi sebebiyle problemin çözümü hesaplamsal olarak çok maliyetlidir. Çözüm için literatürde çeşitli yöntemler geliştirilmiştir. Bu yöntemlerden biri olan QNET yöntemi, bu tez çalışması kapsamında Java diliyle ve Hadoop çatısında kodlanmıştır. 7 düğümlü sorgu ağları için Hadoop gerçekleştirimi 10 makinalı (18 çekirdekli) bir öbekte 11,42 hızlanma sağlamıştır.

Ayrıca literatürde yer alan "referans tabanlı indeksleme yöntemi" incelenerek ESBiD yöntemi geliştirilmiş, bir referans tabanlı indeksleme yöntemi olan RINQ' nun zayıflıkları üzerine çalışmalar yapılmıştır. Bu kapsamda sezgisel yöntemler kullanılarak belirsizlik setindeki ağ sayısı %29,85 oranında, %93,22 doğruluk payıyla azaltılmış, referans ağların seçim yöntemi değiştirilmiş ve belirsizlik setinde biriken ağların daha hızlı hizalanması için "en yüksek dereceli düğüm" tekniği geliştirilmiştir. Bu teknik, QNET' le yapılan tam hizalamanın %89,76 etkinliğine %51,14 daha kısa sürede ulaşmıştır .

Anahtar Kelimeler: Biyolojik ağ hizalama, biyolojik veritabanı hizalama, çizge hizalama, referans tabanlı indeksleme, QNET, Hadoop, ESBiD, en yüksek dereceli düğüm

University : **TOBB Economics and Technology University**
Institute : **Institute of Natural and Applied Sciences**
Science Programme : **Computer Engineering**
Supervisor : **Assoc. Prof. Dr. Osman ABUL**
Degree Awarded and Date : **M.Sc. – July 2013**

Arda SÖYLEV

EFFECTIVE SIMILARITY CALCULATION IN BIOLOGICAL DATABASES

ABSTRACT

It is possible to explain the events occurring inside the cell, the smallest unit in living things, by observing biological networks. Similarity-based analysis is one of the techniques for biological network analysis. In this context, a database consisting of biological networks is aligned with a query network, and the networks having a similarity score higher and lower than a predefined cut-off value are separated. The exact similarity score of two networks needs to be known in the solution of this problem. Unfortunately, because of the NP-complete sub-graph isomorphism problem, this is computationally too expensive. Several methods are proposed in the literature to solve the graph alignment problem. QNET, which is one of these methods, is coded in Java using Hadoop framework in the scope of this thesis. For query networks with 7 nodes, Hadoop implementation with 10 machine cluster (18 cores) achieved 11,42 speedup.

A new method called ESBiD, taking the "reference based indexing method" approach has been developed. Particularly, ESBiD focused on the weaknesses of RINQ, another reference based indexing method. To this end, by using heuristics, the number of networks in the twilight zone has been reduced by 29,85% with 93,22% accuracy, the reference network selection strategy has been changed and a new technique called "highest degree node" has been proposed in order to align the networks in the twilight zone faster. This technique reached 89,74% effectiveness in 51,14% runtime with respect to the QNET's exact alignment method.

Keywords: Biological network alignment, biological database alignment, graph alignment, reference based indexing, QNET, Hadoop, ESBiD, highest degree node

TEŞEKKÜR

Öncelikle bu çalışmayı birlikte yürüttüğümüz, her adımda desteğini esirgemeyen, beni yönlendiren sevgili hocam Osman Abul' a, RINQ yöntemini uygulamamda bana büyük destek veren, gerekli bilgileri, verileri sağlayan Günhan Gülsoy'a ve Hadoop gerçekleştiriminde bana yardımcı olan sevgili Yiğit Çetin'e çok teşekkür ederim.

TOBB Ekonomi ve Teknoloji Üniversitesi'ndeki tüm hocalarıma, özellikle biyoenformatik alanında çalışmama vesile olan Mehmet Tan Hocam' a ve bana imkanlarını açan, çalışmalarıma destek olan KTO Karatay Üniversitesi'ne, mühendislik fakültesi dekanı Ali Okatan Hocam' a da çok teşekkür ederim.

Her an benimle olduğunu bildiğim, beni benden iyi bilen dostlarıma, desteklerini her zaman hissettiğim aileme de sonsuz teşekkür ederim.

İÇİNDEKİLER

ÖZET iv

ABSTRACT v

TEŞEKKÜR vi

ÇİZELGELERİN LİSTESİ ix

ŞEKİLLERİN LİSTESİ x

KISALTMALAR xi

SEMBOL LİSTESİ xii

1 GİRİŞ 1

2 İLGİLİ ÇALIŞMALAR 6

2.1 BLAST 6

2.2 QNET 8

2.3 HADOOP 9

2.3.1 HDFS 10

2.3.2 MapReduce 11

2.4 Öbekleme 13

3 RINQ 16

3.1 Referans Tabanlı İndeksleme Yönteminin Genel Yapısı 16

3.2 Referans Seçimi 17

3.2.1 Birinci Aşama 17

3.2.2 İkinci Aşama 18

3.3 Sınırların Hesaplanması 20

3.3.1 Alt Sınır Hesabı 20

3.3.2 Üst Sınır Hesabı 21

4 QNET HADOOP GERÇEKLEŞTİRİMİ 24

5 REFERANS SEÇİMİ 31

6 İKİNCİ ÜST SINIR BELİRLEME VE SEZGİSEL YAKLAŞIM 36

6.1 İkinci Üst Sınır Hesabı 36
6.2 Sezgisel Yaklaşım 37
7 BELİRSİZLİK SETİ HESABI 43
7.1 Öbekleme Yöntemi 44
7.1.1 Tek Kademeli Öbekleme 44
7.1.2 İki Kademeli Öbekleme 44
7.2 En Yüksek Dereceli Düğüm Tekniği 45
7.3 Belirsizlik Seti Hesabı Sonuçları 46
8 ESBiD YÖNTEMİ 50
8.1 Çevrimdışı Adımlar 50
8.2 Çevrimiçi Adımlar 51
9 SONUÇ 53
KAYNAKLAR 55
ÖZGEÇMİŞ 60

ÇİZELGELERİN LİSTESİ

Çizelge 3.1 Veritabanındaki Ağların Setlere Dağılımı 19
Çizelge 4.1 QNET Çalışma Zamanı Karşılaştırması 25
Çizelge 4.2 QNET Hadoop: 7 Düğümlü Sorgu Ağı Sayısının Performansa Etkisi ... 26
Çizelge 4.3 QNET Hadoop: Düğüm Sayısındaki Artışın Etkisi 27
Çizelge 4.4 Hadoop Hızlanma ve Verimlilik Tablosu 29
Çizelge 6.1 Sezgisel Yaklaşım Sonuçları..... 39
Çizelge 7.1 Belirsizlik Seti Hesabı Sonuçları 46
Çizelge 7.2 Belirsizlik Seti Süre ve Puan Kazancı 46

ŞEKİLLERİN LİSTESİ

Şekil 2.1 Örnek DNA Dizisi Hizalama 7
Şekil 2.2 Çok Düğümlü Hadoop Kümesi [22] 10
Şekil 2.3 MapReduce Çalışması [23] 12
Şekil 2.4 MapReduce Kelime Sayma Örneği [24] 12
Şekil 2.5. Öbekleme Örneği [25] 13
Şekil 3.1 Alt Sınır Hesabı [5] 20
Şekil 3.2 Üst Sınır Hesabı [5] 22
Şekil 4.1 Hadoop Makina Sayısının Çalışma Süresine Etkisi 26
Şekil 4.2 Hadoop Sorgu Ağı Sayısının Çalışma Süresine Etkisi 27
Şekil 4.3 Düğüm Sayısındaki Artışın Çalışma Zamanına Etkisi 28
Şekil 6.1 Eklenen Ağın Belirsizlik Setine Oranı 40
Şekil 6.2 Yeni Eklenen Ağın Doğruluğu 41
Şekil 6.3 Doğru Eklenen Ağ Sayısının Belirsizlik Setine Oranı 41
Şekil 7.1 EYDD Tekniği Süre Kazanımı 48
Şekil 7.2 EYDD Tekniği Puan Kazanımı 49

KISALTMALAR

Kısaltmalar	Açıklama
BLAST	Basic Local Alignment Search Tool
C	Aday Referans Ağı Seti
D	Biyolojik Veritabanı
EYDD	En Yüksek Dereceli Düğüm
ESBiD	Effective Similarity in Biological Databases
HDFS	Hadoop File System
İKÖ	İki Kademeli Öbekleme
LB	Alt Sınır
Q	Sorgu Ağı
RINQ	Reference Based Indexing Of Biological Databases
R	Referans Ağı Seti
TKÖ	Tek Kademeli Öbekleme
UB	Üst Sınır

SEMBOL LİSTESİ

Simgeler	Açıklama
d[i]	Veri tabanı ağının i. düğümü
d_i	Biyolojik veritabanının i. biyolojik ağı
c_i	Aday referans ağı setinin i. aday referans ağı
E_p	P işlemcili paralel hesaplama için verimlilik
$\boldsymbol{E_q}$	q ağının kenarı
$\boldsymbol{EW_q}$	q ağının kenar ağırlığı
LB($\boldsymbol{q, d_i}$)	q ve d_i ağları arasındaki alt sınır değeri
q[i]	Sorgu ağı q' nun i. düğümü
r[i]	Referans ağı r'nin i. düğümü
r_i	Referans ağı setinin i. referans ağı
S_p	P işlemcili paralel hesaplama için hızlanma
sim(q, d_i)	q ve d_i ağları arasındaki benzerlik
T_1	Sıralı algoritmanın çalışma süresi
T_p	P işlemcili paralel algoritmanın çalışma süresi
UB($\boldsymbol{q, d_i}$)	q ve d_i arasındaki üst sınır değeri
ε	Benzerlik eşik değeri
$\boldsymbol{\beta(q[i])}$	Sorgu ağı q' nun i. düğümüyle veritabanı ağı d' nin eşleştiği düğüm
$\boldsymbol{\psi_i}$	Sorgu ağı q ve sorgu ağı r_i arasındaki eşleşme
$\boldsymbol{\delta_i}$	Referans ağı r_i ve veritabanı ağı d arasındaki eşleşme
$\boldsymbol{\phi}$	Sorgu ağı q ve veritabanı ağı d arasındaki eşleşme

1 GİRİŞ

Canlının yapısal ve işlevsel özelliklerini gösteren en küçük birim hücredir. Bir canlının anlaşılması, öncelikle hücrenin ve içerisindeki karmaşık yapının [1] anlaşılmasını gerektirir. Sayısı bazı canlılarda tek, bazılarında yüzlerce, insanda ise trilyonlarca olan hücreler, farklı şekil, büyüklük ve özelliklere sahip olmalarına rağmen, birçok ortak davranış ve özellik taşırlar. Her canlının hücreleri moleküllerden oluşur. Bu moleküller ise her an birbirleriyle etkileşim ve iletişim halindedir. Hayat ise moleküller arasındaki bu etkileşimler sayesinde devam eder. Örneğin hücre içinde meydana gelen metabolik yolaklar enzimleri içeren moleküller arası etkileşim zincirlerini, sinyal yolakları ise, hücre zarı yoluyla iletişim sağlayan moleküler etkileşimleri gösterir [2].

Hücre biyolojisi, binlerce gen ve proteinin farklı seviyelerdeki etkileşimine imkan vermektedir, bu da biyolojik ağların ortaya çıkmasına sebep olur. Organizmada meydana gelen biyolojik süreçleri, moleküller arasındaki etkileşimleri tanımlayan biyolojik ağların anlaşılması, hücreler arasındaki ve hücre içerisindeki karmaşık ancak yaşamın temelini oluşturan yapının anlaşılmasının gereğidir. Birçok önemli biyolojik ağ, DNA, RNA, protein, metabolit gibi moleküller ve bu moleküllerin birbiriyle olan ilişkileriyle ilgilidir. Gen düzenleyici ve sinyal transdüksiyon ağları genlerin nasıl aktive edildiğini ve belli bir zamanda hücrede hangi proteinin oluştuğunu açıklar [3]. Protein-protein etkileşim ağları proteinler arasındaki etkileşimi, metabolik ağlar ise glikoz üretimi gibi metabolizmayı oluşturan birtakım kimyasal tepkimeleri açıklar. Genetik, biyokimyasal ve moleküler biyoloji teknikleri yıllardır bu etkileşimlerin incelenmesinde kullanılmaktadır. Bugün ise yeni geliştirilen tekniklerin bilgisayar teknolojisiyle birleştirilmesi sayesinde hücre içerisindeki ilişkiler ve bunların sebepleri ortaya çıkmaktadır [4].

Bir organizmanın nasıl çalıştığının anlaşılması, biyolojik ağların anlaşılmasıyla olur. Bu gereklilik, ağların farklı yöntemlerle incelenmesi çalışmalarına yol açmıştır.

İnceleme yöntemlerinden bir tanesi karşılaştırma temelli analizdir. İki biyolojik ağ arasındaki benzer kısımların hizalanarak belirlendiği bu yöntem, fonksiyonel benzerlik, ilaç keşfi, metabolik ağın yeni hizalanan genomla tekrar oluşturulması ve filogenetik ağaç oluşturma gibi alanlarda başarılı bir şekilde kullanılmaktadır [5]. Bilgisayar bilimleri vasıtasıyla yapılan bu tip analiz ve incelemeler kapsamında biyolojik ağlar çizge olarak kullanılmakta, moleküller düğümlerle, moleküller arasındaki etkileşimler ise kenarlarla gösterilmektedir [6]. İki biyolojik ağın hizalanması ise çizge hizalama yönteminin kullanılmasından ibarettir. Bu yönteme göre girdi olarak G ve H çizgeleri verildiğinde, G çizgesinin H'ye alt eşlenik bir çizge barındırıp barındırmadığı bulunmaya çalışılmaktadır. Ancak bu problem, teorik bilgisayar bilimleri kapsamında yönsüz Hamilton yolu problemi ve alt küme toplamı problemi gibi bir NP-tam [7] problemdir ve hesaplanması çok maliyetlidir. Henüz polinom zamanda optimal çözümü bulan bir algoritma yoktur.

İki biyolojik ağın hizalanmasının hesaplamsal zorluğunun yanında, biyolojik verilerin eksik ve gürültülü olduğu da bir gerçektir. Bu durum biyolojik ağlarla çalışmayı daha da zorlaştırmaktadır. Biyolojik ağlarla çalışmak ayrıca, biyolojik ağları barındıran veritabanlarıyla çalışmayı gerektirmektedir. Biyolojik veritabanları yüzlerce ağı barındıran yapılardır. Bir ağın, biyolojik veritabanıyla karşılaştırılması, veritabanında yer alan tüm ağlarla teker teker karşılaştırılması demektir. Bu da çalışma zamanını veritabanında yer alan ağ sayısına orantılı olarak arttırmaktadır.

Giriş niteliğinde verilen bu bilgiler ışığında, bu tez kapsamında çözülmek istenen problemin tanımı şu şekildedir; Bir sorgu ağıyla, biyolojik ağlardan oluşan bir biyolojik veritabanı benzerlik temelli karşılaştırılmak istenmekte, bu karşılaştırmanın sonucunda sorgu ağıyla benzer olan ve benzer olmayan biyolojik ağlar ayrışmaktadır. Benzerlik hesaplanırken bir eşik değeri kullanılmakta ve iki ağın benzerliği bu eşik değerinden yüksekse iki ağ benzer, düşükse iki ağ benzer değil denmektedir. Bu kapsamda ağların benzerlikleri hesaplanırken temel problem olan zaman kısıtı göz önünde bulundurulmuş, benzerlik hesaplama süresi azaltılmaya çalışılmıştır.

Literatürde ağların benzerliğinin hesaplanması üzerine yapılmış çeşitli çalışmalar mevcuttur. Ancak bu çalışmalar benzerlik hesaplamanın maliyetli olması sebebiyle belli topolojilerle ve ağ boyutuyla sınırlandırılmış, farklı kısıtlamalara sahip yöntemlerdir [8, 9, 10, 11]. Ayrıca bu çalışmalar yüksek zaman ve hafıza karmaşıklığına sahiptir. Bunların dışında, kullanıcının tanımladığı sorgu ağlarıyla benzerlik hesabı yapan ve yaklaşık sonuçlar bulan algoritmalar da vardır [12, 13].

Literatürde indeksleme yaparak benzerlik bulan yöntemler de görülmektedir. Bu yöntemler üç farklı sınıfta incelenir; özellik tabanlı yöntemler, ağaç tabanlı yöntemler ve referans tabanlı yöntemler. Özellik tabanlı indeksleme yöntemleri ağın belli bir özelliğini temel alarak indeksleme yapar. Sorgu ağındaki bir özelliği veritabanındaki ağlarda arar. SAGA [8], GraphGrep [9], gIndex [10] ve Grafill [11] özellik tabanlı indeksleme yöntemini kullanır. Ağaç tabanlı indeksleme yöntemleri ise veritabanındaki ağları ağaçtaki yapraklara yerleştirir ve şu şekilde çalışır; verilen bir sorgu ağını ağacın kökünden başlayarak her düğümle hizalar ve bu şekilde ilerleyerek ağacın alt dallarına doğru iner. Bu esnada kullanılan bir eşik değerine bağlı olarak ağaçta budama yapar [12]. Ağaç tabanlı indeksleme yöntemini kullanan Closure Tree (CTree) [13] metodu bilinmektedir. Bu anlatılan iki yöntem biyolojik ağlar için çok uygun değildir. Bunun sebebi biyolojik ağların çok büyük olması ve bahsedilen yöntemler kullanıldığında benzerlik hesabının çok uzun sürmesidir [5]. Referans tabanlı indeksleme yöntemine göre ise biyolojik veritabanı içinden bir referans ağlar kümesi oluşturulur. Seçilen referans ağların boyutu, veritabanındaki ağlara göre çok daha küçüktür. Ancak bu ağlar seçilirken referans ağların veritabanındaki ağları temsil etmesi istenir ve oluşturulan sorgu ağı referans ağlarla hizalanır. Hizalamanın sonucunda ortaya çıkan eşleşmeler kullanılarak veritabanındaki ağlarla sorgu ağının benzerliğine dair bir alt ve üst sınır bulunur. Bu sayede daha önceden belirlenen bir eşik değerine göre, sorgu ağıyla veritabanında benzer olan ve olmayan ağlar ayrılır. Arada kalan ve benzerliğine bu yöntemle kesin karar verilemeyen bazı ağlarla da uzun süren tam hizalama işlemi yapılır. Bu işleme tabi tutulan ağ sayısının az olması yöntemin başarısının en önemli göstergelerindendir. Referans tabanlı indeksleme yöntemini RINQ kullanmaktadır.

Bu çalışma kapsamında da biyolojik ağların benzerliğinin hesaplanması için şu anda en verimli ve yeni yöntem olduğuna karar verilen "referans tabanlı indeksleme yöntemi" kullanılmıştır. Bu yöntemi ortaya çıkartan RINQ, detaylı olarak incelenmiş ve yöntemin temel yapısı sabit kalmak üzere ESBiD adında farklı bir yöntem oluşturulmuştur.

Ayrıca bu çalışmada QNET [14] metodu Hadoop [15] altyapısına uygun olarak kodlanmış ve farklı sayılarda makinalar kullanılarak dağıtık yapıda çalıştırılmıştır. 4, 7, 9 ve 10 makinalı öbekler oluşturulmuş, algoritmanın performansı ölçülmüştür.

Bu çalışmalarda ve analizlerde kullanılmak üzere KEGG [16] veritabanında bulunan ve 15'ten fazla düğüme sahip gen düzenleyici ağlardan 297 tanesi seçilerek biyolojik veritabanı oluşturulmuştur.

Özet olarak tez kapsamında yapılanlar şunlardır;

Öncelikle RINQ yöntemi JAVA diliyle kodlanmış ve performans analizi yapılmıştır. Bunun haricinde, yapılan çalışmalar iki kısma ayrılabilir. Bunlardan ilki performansı arttırmak için yapılanlar, ikincisi ise yöntemde yapılan geliştirmelerdir.

Yöntemin performansını arttırmak için yapılanlar şunlardır;

- C++ diliyle RINQ 'da kullanılan QNET algoritması JAVA diliyle tekrar kodlanmıştır.

- QNET, Hadoop çatısında programlanmış ve dağıtık olarak farklı sayıda makinada çalıştırılmıştır.

Yapılan bu iki geliştirmenin performansa etkisi, eski yöntem de dahil edilerek karşılaştırılmış, sonuçlar tablo ve grafiklerle Bölüm 4'te incelenmiştir.

Yöntemde yapılan geliştirmeler ise şunlardır;

- Üst sınır hesabında kullanılmak üzere sezgisel bir yaklaşım oluşturulmuş, belirsizlik setindeki ağ sayısı azaltılarak doğru sonuçların sonuç setine aktarılması sağlanmıştır.

- Belirsizlik setindeki ağların hizalanması için öbekleme ve bu kapsamda oluşturulan, "en yüksek dereceli düğüm" tekniği denenmiş, sonuçlar analiz edilmiştir.

- Referans Seti oluşturma adımında değişiklik yapılmış, ağların biyolojik özellikleri de dikkate alınarak referansların oluşturulması sağlanmıştır.

Yöntemde yapılan değişikliklerin detayları Bölüm 5, 6 ve 7'de anlatılmakta, etkileri incelenmektedir.

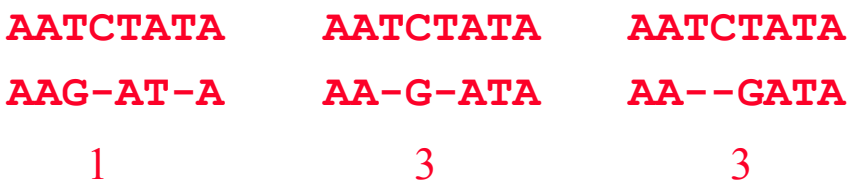
2 İLGİLİ ÇALIŞMALAR

2.1 BLAST

Bir biyolojik ağın yapısında barındırdığı birçok molekül vardır. Bu moleküllerin her birinin farklı nükleotid, amino asit dizileri vardır. Bu bağlamda, iki DNA veya protein dizisinde benzerlik, homoloji bulmak için bu dizilerin hizalanması gerekir. Hizalama sonucunda ise önemli bulgular elde edilebilir. Bu bulguların bazıları şunlardır;

- Dizilerde motif olarak adlandırılan önemli bölgeler bulunur
- Fonksiyonel benzerlikler belirlenir
- Evrimsel yakınlıklar ortaya çıkartılır.

Benzerlik bulmak için dinamik programlama algoritmaları sıklıkla kullanılmaktadır. Bunlardan en çok bilinenleri global hizalama için Needleman & Wunsch [17], lokal hizala için ise Smith & Waterman' dır [18]. Bu yöntemlere göre iki farklı DNA veya protein dizisi alt alta koyulur. Karşılıklı gelen her sembolün belli bir puanı vardır. Puanın yüksekliği benzerliğin fazlalığını gösterir. Bu puanlar biyolojik çalışmalar sonunda elde edilen PAM [19], BLOSUM [20] gibi benzerlik dizeylerine göre belirlenmektedir. İki diziden en yüksek puanlı karşılıklı gelen dizilimler, bahsedilen algoritmalarla oluşturulur. Bunu yaparken bazı sembollerin karşısına hiçbir sembol de gelmeyebilir ancak böyle olursa bir ceza puanı toplam benzerlik puanından düşülür. Bu sayede iki dizi için en uygun benzerlik puanı ortaya çıkar. Toplam benzerlik puanı ne kadar yüksekse iki dizi o kadar benzerdir denilebilir. Şekil 2.1'de iki farklı dizinin farklı şekillerde hizalanması gösterilmektedir. Hizalama yapılırken benzer semboller için +1 puan ve boşluklar için -1 puan verilmiştir. Üç farklı hizalamadan her birinin benzerlik puanı altlarına yazılmıştır. Dizi hizalamada amaç en yüksek puanlı hizalamayı elde etmektir.



Şekil 2.1 Örnek DNA Dizisi Hizalama

Bahsedilen dizi hizalama algoritmaları en doğru benzerlik puanını bulduklarından süre ve yer karmaşıklıkları yüksektir. Tek bir hizalamada ihmal edilebilecek yüksek karmaşıklık, iki büyük genom dizisi hizalandığında veya birden fazla dizi arka arkaya hizalandığında göz ardı edilemeyebilir. Bu tip durumlarda sezgisel bir yöntem olan BLAST [21] yaygın olarak kullanılmaktadır. BLAST yöntemi 1990 yılında ortaya çıkmıştır. Daha önce kullanılan Smith & Waterman lokal hizalama yöntemine göre çok daha hızlı çalışmaktadır. Verilen bir sorgu ağıyla veritabanındaki ağları kısa bir sürede hizalayıp, veritabanında sorgu ağına benzer olan tüm dizileri ayırır. Bunu yaparken sonuçların doğruluğunu arttırmak için bir takım istatistiksel yöntemler de kullanır. BLAST' ın bu derece hızlı sonuç vermesinin bir sonucu olarak, en doğru sonucu veremeyebileceği göz ardı edilmemelidir.

Bu çalışmada BLAST metodu kullanılmaktadır. Her sorgu, referans ve biyolojik ağın veritabanında bir benzerlik dizeyi bulunmaktadır. Dizeylerde, ağlardaki her molekülün veritabanındaki diğer tüm moleküllerle olan benzerlik puanları yer alır. Bu benzerlik puanları sayesinde de çizgeler hizalanır. Dizeylerdeki bu benzerlikler BLAST metodu kullanılarak hesaplanmaktadır. Her molekülün bir dizisi vardır ve iki dizi arasındaki benzerliği bulmak için BLAST yöntemi kullanılır. BLAST puanı olarak da e değerinin[1] ters logaritmasının 10 katı, 0 ve 100 değerleri arasına oranlanarak alınır.

[1] ing: Expectation Value

2.2 QNET

QNET, renk kodlama algoritması kullanarak ağaç veya ağaç genişlikli ağları[2] herhangi bir sorgu ağıyla belli bir güven değerine göre hizalar [14]. Ancak bu hizalama işlemi hesaplamsal olarak çok maliyetlidir. Herhangi bir 7 düğümlü sorgu ağıyla 297 ağdan oluşan tüm biyolojik veritabanını hizalamak 4 işlemcili 3.00 GHz Intel Core i5 bilgisayarla yaklaşık 6,5 saat sürmektedir. Bu süre, ağdaki düğüm sayısı 8'e yükseldiğinde 56,5 saati bulmaktadır.

Bu çalışmada QNET algoritması çok önemli bir yere sahiptir ve üzerinde çalışılan uygulamaların birçok adımında kullanılmaktadır. Bu kapsamda QNET' in kullanımını çevrimiçi ve çevrimdışı kullanımlar olarak ikiye ayırabiliriz. Çevrimdışı kullanımlar sadece bir kere uygulanır ve sonuçları ileride tekrar kullanılacaksa kaydedilir. Bu işlemin uzun sürmesi göz ardı edilebilir. Çevrimdışı olarak QNET algoritması iki yerde kullanılmaktadır. Bunlardan ilki referans ağların seçimi esnasındadır. Bu seçimde referansların birbirleriyle olan benzerliklerinin belli bir eşik değerinin altında olması, referansların farklılığını korumak amacıyla istenir. Bu da QNET vasıtasıyla sağlanır. Aday referans ağ, seçilmiş referanslarla tek tek hizalanır ve tüm referanslarla benzerliğinin belirlenen eşiğin altında olması istenir. QNET' in ikinci kullanımı ise referans ağlarıyla veritabanındaki büyük biyolojik ağların hizalanmasıdır. Bu çok maliyetli ve her referans ağı için yaklaşık 6,5 saat süren bir işlemdir. Bu benzerlikler kaydedilir ve sadece bir kez yapılıp tekrar tekrar kullanılır. QNET' in çevrim içi kullanımlarından biri sorgu ağıyla referans ağlarının hizalanması esnasındadır. Bu hizalama iki tarafın da düğüm sayısının az olması sebebiyle çok maliyetli değildir. İkincisi ise belirsizlik setindeki ağların sorgu ağıyla hizalanması esnasındadır. Bu işlem belirsizlik setinde biriken ağ sayısına göre uzun ve çok maliyetli olabilmektedir. Bunun için önerilen yaklaşım Bölüm 7' de anlatılmaktadır.

[2] ing: treewidth networks

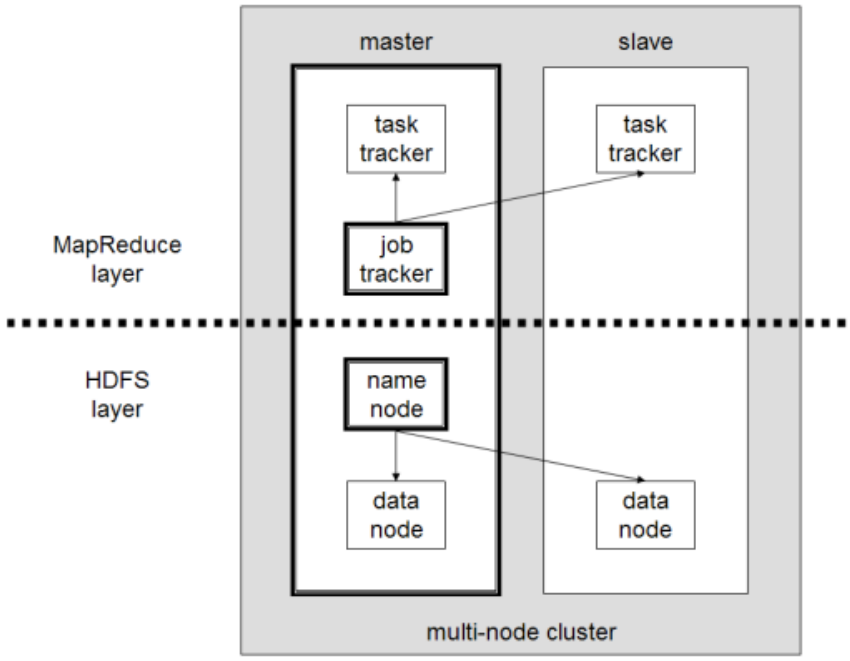
Girdi olarak ilgili ağların benzerlik dizeylerini alan ve bu sayede iki çizgenin benzerliğini ve hangi düğümlerin birbirleriyle eşleştiğini bulan QNET algoritması sezgisel bir yöntemdir. Algoritmanın performansıyla ilgili yapılan çalışmalar Bölüm 4'te anlatılmaktadır.

2.3 HADOOP

Apache Hadoop [15], büyük verileri işlemek için Google tarafından açık kaynak koduyla, Java dili kullanılarak geliştirilen, binlerce bilgisayarın bir arada, tek bir makina gibi dağıtık olarak çalışmasını sağlayan güvenilir ve ölçeklenebilir bir yapıdır.

Hadoop sistemi sayesinde, dağıtık sistemlerde binlerce bilgisayar olması sebebiyle çokça ortaya çıkan hatalar, makinalardaki bozulmalar tüm sistemi etkilememekte, kolaylıkla baş edilebilmektedir. Ayrıca bu yapıyı kullanarak büyük verileri işleyen yazılımlar kolaylıkla oluşturulabilmektedir. Yazılımcılar dağıtık sistem mimarisinin karmaşık yapısıyla uğraşmak yerine basit bir programlama modeli kullanırlar. Böylelikle sadece makinaların yapacağı işi ve bu işlerin sonuçlarının bir araya gelmesiyle yapılması gereken işin kodunu yazarak istedikleri işin Hadoop tarafından yapılmasını sağlayabilirler. Sistem şu anda küçük, tek bir kümeden oluşan ve sadece birkaç bilgisayarı barındıran yapılardan, birçok kümeye sahip ve içinde binlerce bilgisayarı barındıran yapılara kadar kullanılabilmektedir. Yani Hadoop sistemi günümüzde bireysel bir yazılımcıdan, Google, Yahoo, Amazon, Ebay, Facebook gibi şirketlere kadar kullanılmaktadır.

Hadoop' un yapısı iki parçadan oluşur, bunlardan ilki veriyi binlerce bilgisayara dağıtan bir dosya sistemi, HDFS, ikincisi ise bu makinaların birlikte iş yapmasını sağlayan, işi küçük parçalara bölerek makinalara, paralel olarak çalışması için dağıtan MapReduce' dur.



Şekil 2.2 Çok Düğümlü Hadoop Kümesi [22]

2.3.1 HDFS

Veriyi genellikle 64 MB veya 128 MB boyutlarında bloklara bölerek makinalara dağıtan dosya sisteminin Hadoop' daki adı HDFS' dir. HDFS' in özelliği gigabayt, terabayt hatta petabayt boyutundaki çok büyük veriyi saklayabilme kapasitesidir. Genellikle veriyi 2 ya da 3 kopya[3] bloklar halinde saklar. Şekil 2.2'de gösterildiği gibi HDFS, içinde NameNode ve DataNode' u barındırır.

- **NameNode**: HDFS' i idare eden merkez konumundadır. Bloklar halinde bölünmüş veri parçalarının hangi makinalarda saklandığını bilir ve bunun koordinasyonunu yapar. Sistemde bir tane NameNode ve herhangi bir problem olursa yerine geçmesi için bir tane de yedek NameNode vardır. NameNode olmadan HDFS sistemi çalışamaz.

[3] ing: replication

- **DataNode**: Sistemde birçok DataNode vardır. Bunlar veri bloklarını depolayan makinalardır. Datanode' lara emirleri NameNode verir ve DataNode' lar bu doğrultuda hareket ederler. Ayrıca belli aralıklarla sahip oldukları blokların bilgisini NameNode' a gönderirler.

2.3.2 MapReduce

Veri üzerindeki işlemler MapReduce sayesinde yapılır. Bir MapReduce işi[4], kullanıcının yapılmasını istediği bir görev parçacığıdır. Bu işin içinde girdi verisi, MapReduce programı ve konfigürasyon bilgileri vardır. Hadoop bu işi görevlere ayırır, bu görevler de Map ve Reduce olarak ikiye ayrılır [23].

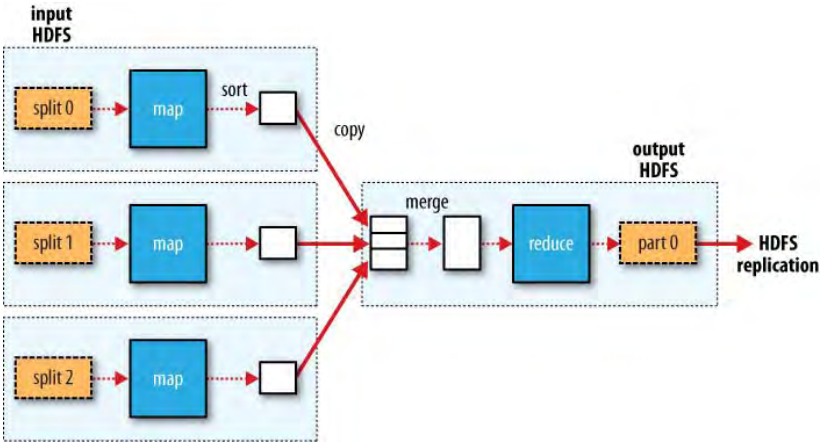
Öncelikle veri, split adımında, boyutuna göre parçalara ayrılır ve her parça farklı bir map görevi olarak adlandırılır. Bu işlerin yapılmasını Şekil 2.2'de gösterildiği gibi iki farklı birim sağlar, bunlar JobTracker ve TaskTracker' dır.

- **JobTracker**: Bütün işlerin koordinasyonunu yaparak, iş parçaları olan map görevlerini TaskTracker' lara iletir.

- **TaskTracker**: İşleri yapan birimlerdir. Verilen görevi yapar ve sonucu tekrar JobTracker' a iletirler. Eğer yapılan işte veya sonucun iletilmesinde herhangi bir aksaklık olursa JobTracker işi başka bir TaskTracker' a gönderir.

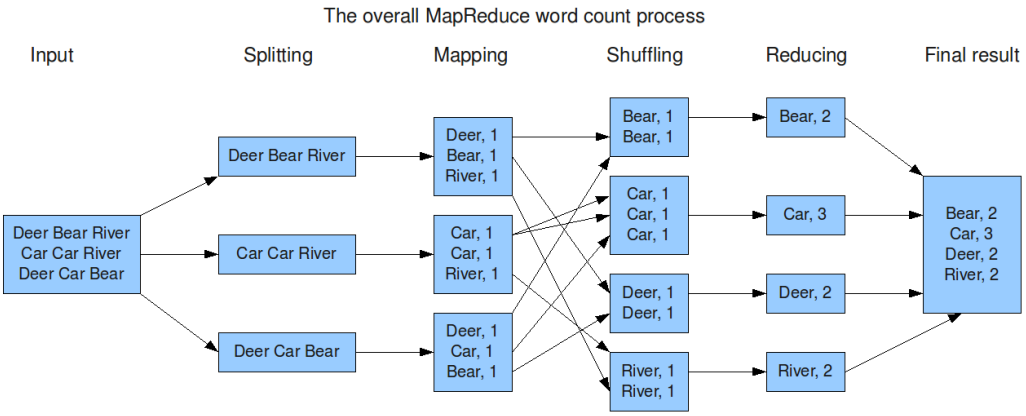
Mapper' lardan çıkan sonuçlar, eğer kullanılıyorsa, Reduce görevine gider. Burada, gelen sonuçlar birleştirilerek çıktı olarak tek bir sonuç oluşturulur. Bazı durumlarda Reduce adımına ihtiyaç olmayabilir. Bu tip durumlarda TaskTracker' lar sonuçları doğrudan HDFS' e yazarlar. Anlatılanlar Şekil 2.3'de gösterilmektedir.

[4] ing: MapReduce job



Şekil 2.3 MapReduce Çalışması [23]

Daha ayrıntılı bir anlatımla; MapReduce, anahtar ve değer ikilileri[5] şeklinde yapılanan veriyle çalışır. Her Map görevi anahtar/değer (K/V) ikilisini girdi olarak alır ve çıktı olarak yeni bir anahtar/değer ikilisi üretir. Tüm Map görevlerinin sonuçlarından gelen anahtar/değer ikilileri anahtarlara göre sınıflandırılır, böylelikle aynı anahtara sahip tüm değerler bir araya toplanır. Bu değerler de Reduce adımına gönderilir. Reduce adımında ise gelen tüm değerlerden tek bir sonuç değeri üretilir.



Şekil 2.4 MapReduce Kelime Sayma Örneği [24]

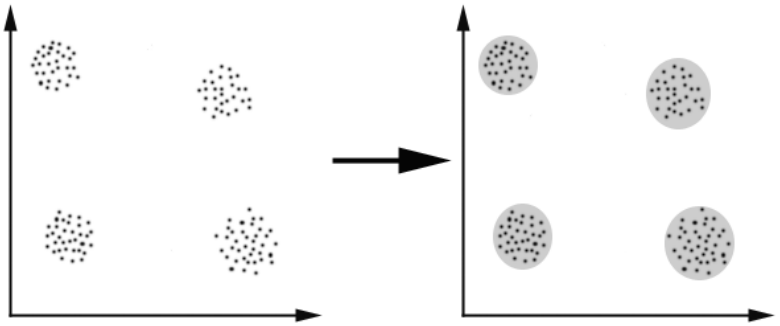
[5] ing: key-value pair

Şekil 2.4'te, sıkça rastlanan bir kelime sayma uygulamasının örneği vardır. Büyük bir dokümanın içinde hangi kelimenin kaç kere geçtiğini hesap eden bir uygulamadır. Şekilde Hadoop adımları açıkça gözükmektedir.

Bu çalışmada Hadoop, QNET uygulamasının dağıtık mimaride çalıştırılıp, hızlandırılması için kullanılmıştır. Deneylerde 4, 7, 9 ve 10 makinanın olduğu 4 farklı Hadoop öbeği kullanılmış ve bu farklı yapılandırmalarda ortaya çıkan performanslar karşılaştırılmıştır. Buna ek olarak birden çok sorgu ağının veritabanıyla aynı anda karşılaştırılması esnasında Hadoop' un performansı ölçülmüştür. QNET' te sorgu ağındaki düğüm sayısının artması, algoritmanın çalışma hızını çok çabuk arttırmaktadır. Bu durumun etkileri de tablo ve grafiklerle incelenmektedir. Son olarak hızlanma ve verimlilik hesapları da yapılmıştır. Tüm bu çalışmaların detayları Bölüm 4'te verilmektedir.

2.4 Öbekleme

Veri madenciliği tekniklerinden olan öbekleme kısaca, objeleri öbek adı verilen anlamlı alt gruplara ayırma işlemidir. Bu gruplandırma objelerin benzerliklerine göre yapılmaktadır. Aynı öbekte yer alan objelerin benzer olması ve bu öbeklerin diğer öbeklerdeki objelerden farklı olması amaçlanır. Bir gözetimsiz öğrenme[6] yöntemidir.



Şekil 2.5. Öbekleme Örneği [25]

[6] ing: unsupervised learning

Şekil 2.5'de uzaklık özelliği öbeklemede kullanılmış, birbirine yakın olan noktalar aynı öbekte yer almış ve doğal olarak 4 farklı öbek ortaya çıkmıştır. Öbekleme denetlenmeyen bir sınıflandırmadır. Bunun anlamı ise ilk başta ne kadar sınıfa ayrım yapılması gerektiği, oluşacak sınıflara ait özellikler gibi bilgilerin olmamasıdır [26].

Öbekleme metotları 5 farklı kategoriye ayrılmaktadır [27].

1. Bölümleme Algoritmaları
2. Hiyerarşik Algoritmalar
3. Yoğunluk Tabanlı Algoritmalar
4. Grid Tabanlı Algoritmalar
5. Model Tabanlı Algoritmalar

Bölümleme algoritmaları n adet nesnesi k adet öbeğe ayırır. Bu tip algoritmalar genel olarak ilk başta rastgele bir ilk dağıtım yaparlar. Sonra da tekrarlı olarak nesneleri genel sonuç optimize edilecek şekilde öbekler arasında taşırlar. Ancak bu tarz algoritmaların sıkıntısı ilk dağıtıma ve yerel en uygun değere çok bağımlı olmalarıdır. Sonuçta önsel bir bilgi genelde yoktur, bu sebeple gözlemlenen veri kümesi üzerinden öbek sayısını hesaplamak gerekir. Bu problem öbek doğrulama problemiyle alakalıdır [26]. Bu tip algoritmaların en çok bilineni k-means ve k-medoids (PAM)'dir.

Hiyerarşik algoritmalar ise tümden gelim ve tüme varım şeklinde iki farklı şekilde çalışırlar. Tüme varımda her obje, bir öbek olarak düşünülür ve kendisine en yakın olan diğer objelerle birleşir. Böylelikle öbekler büyür, öbek sayısı azalar. Tümden gelim metodunda ise tek bir öbek olduğu varsayılır ve bu öbek her adımda bölünür. Bu sayede de öbek sayısı artar. Algoritmanın sonunda bir dendrogram, yani ağaç şeklinde çizilmiş bir şekil oluşur.

Yoğunluk tabanlı algoritmalar yoğunluğa göre öbekleme yaparlar. Her öbeğin yoğunluğu, öbeğin dışına göre yüksektir. Bu algoritmalardan en popüler olanları DBSCAN [28] ve OPTICS [29]' dir.

Grid tabanlı öbekleme algoritmalarının en çok bilinenleri STING [30], WaveCluster [31] ve CLIQUE' dir. Bu algoritmalar kümeleme yapmak için çok çözünürlüklü kümeleme yaklaşımını ve yoğun grid hücrelerini kullanırlar [32].

Son olarak öbekler için belli modellerin kullanıldığı ve veriyle model arasındaki uyumluluğun optimize edilmeye çalışıldığı model tabanlı öbekleme yöntemi vardır. SOM [33], çok kullanılan model tabanlı öbekleme yöntemidir.

Bu çalışmada öbekleme, belirsizlik seti hesabında kullanılmaktadır. Kullanılan yaklaşımın detayları ve sonuçları Bölüm 7'de incelenmektedir.

3 RINQ

Bu bölümde RINQ [5] yöntemi detaylı olarak anlatılmaktadır. Açık kaynak kodu bulunmaması sebebiyle RINQ, bu çalışma kapsamında JAVA diliyle kodlanmıştır.

3.1 Referans Tabanlı İndeksleme Yönteminin Genel Yapısı

Bir referans tabanlı çizge hizalama yöntemidir. Yöntemin en temel özelliği, tüm veritabanını temsil etmesi beklenen bir referans setinin oluşturulmasıdır. Sorgu ağı, veritabanındaki tüm ağlarla tek tek hizalanmaz, bunun yerine daha küçük boyutlu olan, bu sebeple daha hızlı hizalanabilen, referans ağlarla hizalanır. Bu hizalamaların sonuçları ve daha önceden yapılan referans ağı - sorgu ağı eşleşmelerinin sonuçları kullanılarak sorgu ağıyla veritabanı ağları arasında bir eşleşme ortaya çıkar. Bu eşleşme aranan eşleşmedir.

Yukarıda bahsedilen, referans ağlarla veritabanı ağlarının eşleşmesi maliyetli bir işlemdir. Bu hizalama işlemi çevrimdışı olarak sadece bir kez yapılır ve eşleşme sonuçları saklanarak tekrar tekrar kullanılır. Bu sebeple, işlemin uzun sürmesinin uygulamanın performansına olumsuz bir etkisi yoktur.

Bir sorgu ağı q' yla, biyolojik veritabanının hizalanması şu şekilde olur; kullanıcı öncelikle bir eşik değeri, ε belirler ve sorgu ağı tüm referans ağlarla tek tek hizalanır. Bu hizalamalar sorgu ve referans ağlarındaki düğüm sayılarının az olması sebebiyle çok maliyetli değildir. Sorgu - referans ve daha önceden hesaplanan referans - veritabanı eşleşmeleri kullanılarak veritabanındaki her ağ için sorgu ağıyla benzerliğine dair bir alt sınır, $LB(q, d_i)$ ve bir üst sınır, $UB(q, d_i)$ belirlenir. Bu işlemlerin sonunda üç farklı durum ortaya çıkar:

- Filtre Seti: $UB(q, d_i) < \varepsilon$
- Sonuç Seti: $\varepsilon \leq LB(q, d_i)$
- Belirsizlik Seti: $\mathrm{LB}(q, d_i) < \varepsilon \leq UB(q, d_i)$

Filtre setindeki ağlar sorgu ağına benzemediği için elenir, sonuç setindeki ağların da sorgu ağına benzerliği eşik değerinden yüksek olduğu için alınır. Burada en büyük problem belirsizlik setidir. Bu setteki ağlar için bir karar verilememiştir. Bundan dolayı bu setteki ağlarla sorgu ağının hizalanması uzun ve maliyetli çizge hizalama yöntemiyle tekrar yapılır.

Bu yöntemde kullanılan hizalama, iki ağdaki düğümlerin birbirleriyle eşleştirilmesidir. Q ve d arasındaki eşleşme β ile gösterilirse, $\beta(q[i]) = d[j]$ demek, q[i], d[j]' yle eşleşiyor demektir. Bu eşleşmeler bire bir ve simetriktir. Bazı durumlarda hiçbir düğümle eşleşmeyen düğümler de bulunur, bu da $\beta(q[i]) = \emptyset$ şeklinde gösterilir. Bu tür durumlar ekleme veya silme[7] olarak kabul edilir. Bunların bir ceza puanı vardır. Ayrıca E_d ve E_q'yu q ve d'nin kenarları olarak kabul edersek E_d kenarının ağırlığını EW_d olarak gösteririz. Bu bilgiler ışığında q sorgu ağıyla d veritabanı ağının benzerliği şu şekilde hesaplanır;

$$\begin{aligned} \text{sim(q, d)} = & \sum_{\beta(q[i])\neq 0} sim(q[i], \beta(q[i])) \\ & + \sum_{\beta(q[i])=0} Indel\ Penalty \\ & + \sum_{\substack{(\beta(q[i]),\beta(q[j]))\in E_d \\ AND\ (q[i],q[j])\in E_q}} EW_d(\beta(q[i]), \beta(q[j])) \end{aligned} \qquad (3.1)$$

3.2 Referans Seçimi

Doğru referansların seçimi bu metodun yapı taşlarındandır. Referanslar iki aşamada seçilir.

3.2.1 Birinci Aşama

Bu aşamada üç özellik göz önünde bulundurularak bir aday referans seti, C oluşturulur. Bu özelliklerden ilki seçilen referans ağının küçük, az sayıda düğüme

[7] ing: insertion / deletion

sahip olmasıdır. İkincisi ise kapsamlı olması, yani sorgu ağının veritabanındaki en az bir ağla iyi, yüksek puanlı eşleşmesidir. Sonuncu özellik ise seçilen referansın diğer seçilmiş referanslarla benzer olmamasıdır. Böylelikle gereksiz referans ağlarının varlığı performans düşmesine sebep olmayacaktır.

Bu hedefler dahilinde ilk aşamada veritabanından rastgele bir ağ ve bu ağdan rastgele bir düğüm seçilir. Bu düğüme de komşu düğümlerinden eklemeler yapılarak istenilen sayıda düğüme sahip bir ağ oluşturulur. Bu oluşturulan ağ ikinci aşamada, daha önce seçilmiş olan ağlarla hizalanır. Bu hizalama belirlenen bir eşik değerine göre yapılır ve seçilen ağın daha önce seçilmiş herhangi bir ağla benzerliği bu eşik değerinden yüksekse bu ağ referans ağı olarak seçilmez, başka bir aday referans ağı oluşturularak işlem devam eder. Bu şekilde aday referans ağları seti oluşturulur.

3.2.2 İkinci Aşama

Birinci aşamada seçilen aday referanslar, referans seçmenin üç temel şartını sağladığı için referans setine eklenebilir. Eğer daha hassas referans seçimi yapılmak istenirse, referans seti optimize edilebilir. Bunun için aşağıda sözde kodu verilen Algoritma 1 kullanılır.

Algoritma 1: *Referans Seti Optimizasyonu*

R referans setini oluştur
Q ve C'deki ağları D'deki ağlarla hizala
C'den rastgele k tane ağı sil ve bunları R'ye ekle
repeat
    Doğruluğu, R'nin doğruluğu olarak belirle
    R'den $R-\{r_i\}$' yi maksimum yapan r_i 'yi sil
    for all C'deki c_j *için* **do**
        $R \cup \{c_j\}$' yi kullanarak doğruluğu hesapla
    end for
    if Çıkan doğruluk değeri mevcut olandan daha iyiyse **then**
        R'ye $R \cup \{c_j\}$ ' yi en büyük yapan c_j 'yi ekle
        C'den c_j ' yi sil
    else
        R'ye tekrar r_i ' yi ekle ve döngüden çık
    end if
until Doğruluk daha fazla geliştirilemez veya C boş

Algoritma 1, en yüksek doğruluk değerine sahip referans setini tepe tırmanma metoduyla bulmaya çalışır.

Öncelikle rastgele bir referans seti oluşturur, ardından her iterasyonda R'deki bir referans ağı C - R'deki bir ağ ile doğruluğu arttırmak için değiştirilir. Bunun için ilk olarak referans setinden doğruluk değerini en az azaltacak referans silinir, ardından aday referans setleri arasından doğruluk değerini en çok arttıracak olan referans bulunur ve referans setine eklenir. Algoritma bu şekilde devam eder ve sonunda aday referans setinde, referans setine eklenmesi doğruluk değerinde herhangi bir gelişme olmasına sebep olacak bir ağ kalmayınca sonlanır.

Burada kullanılan doğruluk değeri de şu şekilde hesaplanır; Bölüm 3.1'de sonuç seti, filtre seti ve belirsizlik seti anlatılmaktadır. Aşağıda verilen Çizelge 3.1'e göre sonuç setinde sadece doğru sonuçlar bulunurken, filtre ve belirsizlik setlerinde doğru ve yanlış sonuçlar bulunabilmektedir.

Çizelge 3.1 Veritabanındaki Ağların Setlere Dağılımı

	Sonuç Seti	**Belirsizlik Seti**	**Filtre Seti**
Doğru	N_1	N_2	N_3
Yanlış	0	N_4	N_5

Daha net bir ifadeyle, sonuç seti tam doğru sonuçları bulmaktadır, o sebepten yanış herhangi bir sonuç bu grupta yer almaz. Filtre setinde doğru ve yanlış sonuçların bulunabilmesinin sebebi ise filtre setinin yaklaşık sonuç vermesidir. Bu durumda filtre setinde eşik değerinden yüksek benzerliğe sahip sonuçlar da bulunabilmektedir. Bu durumda doğruluk değeri şu şekilde hesaplanır;

$$\frac{N_1 + N_2}{N_1 + N_2 + N_3} \tag{3.2}$$

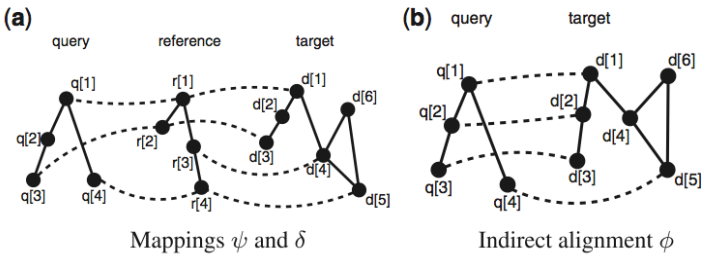
3.3 Sınırların Hesaplanması

Bölüm 3.1'de bir sorgu ağı q' yla, daha önce seçilen referans ağları kullanılarak bir alt sınır, $LB(q, d_i)$ ve bir üst sınır, $UB(q, d_i)$'nin nasıl hesaplandığı genel olarak tarif edilmektedir. Bu bölümde ise bu hesaplamaların detayları anlatılmaktadır. İleride, q ve r_i arasındaki eşleşme ψ_i'yle, r_i ve d arasındaki eşleşme de δ_i' yle anlatılmaktadır.

3.3.1 Alt Sınır Hesabı

Alt sınır hesabı iki aşamada yapılmaktadır. İlk aşamada sorgu ağı q' nun düğümleri, d'nin alt kümesiyle r_i 'nin daha önceden kaydedilen eşleşmeleri kullanılarak eşleştirilir. İkinci aşamada ise eğer mümkünse boşta kalan düğümler eşleştirilir ve son hizalama bulunur. Bu aşamaların ayrıntıları şöyledir;

Birinci Aşama: Bu aşamada (q, r_i) ve (r_i, d) ikilileri arasındaki eşleşmelerden faydalanılarak r_i üzerinden (q, d_i) arasındaki eşleşmeler bulunur. Bu da ϕ_i kullanılarak $\phi_i(\mathrm{q[j]}) = \delta_i(\psi_i(\mathrm{q[j]}))$ şeklinde gösterilir. [5]'den alınan Şekil 3.1(a)'da görüldüğü gibi q[3]'le r[2], r[2]' yle de d[3] eşleşmiştir. Bu eşleşmelerin nasıl yapıldığı Bölüm 3.1'de anlatılmaktadır. Bu iki eşleşmeden faydalanılarak q[3], d[3]'le eşleştirilmiştir.



Şekil 3.1 Alt Sınır Hesabı [5]

Sıkı bir alt sınır bulmak için hesaplamalarda her referans, r_i, birbirinden bağımsız olarak kullanılmaktadır. Sorgu ağı q, her referans ağı r_i'yle eşleştirilmekte ve k tane

farklı LB_i(q, d) puanı elde edilmektedir. Bunlardan da en yüksek puanlı eşleşme alt sınır değeri olarak alınmaktadır. Yani LB(q, d) = $max_i\{LB_i(q,d)\}$.

İkinci Aşama: q ve d arasında birbiriyle eşleşmeyen düğümler olduğunda ikinci aşama kullanılır. Bu durumda yapılabilecek iki farklı seçenek vardır. Bunlardan ilki açıkta kalan, eşleşemeyen düğümleri ekleme ve çıkarma (insertion, deletion) olarak kabul edip bir ceza puanı vermektir. Bu yöntem benzerlik puanını düşürdüğü için çok tercih edilmez. Bunun yerine iki tarafta da boşta kalan düğümler varsa birbirleriyle eşleştirilebilir. Şekil 3.1(a)'da q[2] ve d[2] herhangi bir düğümle eşleşmemiştir. Bu durumda bu iki düğüm birbirleriyle Şekil 3.1(b)'de eşleştirilmiş, iki düğümün benzerlik puanı da toplam benzerliğe ilave edilmiştir. Boşta kalan düğümleri bulmak için q ve d ağlarında kök düğümden başlanarak genişlik öncelikli arama [8] yapılır. Sorgu ağı q' yla d arasındaki son eşleşme Şekil 3.1(b)'de gösterilmektedir.

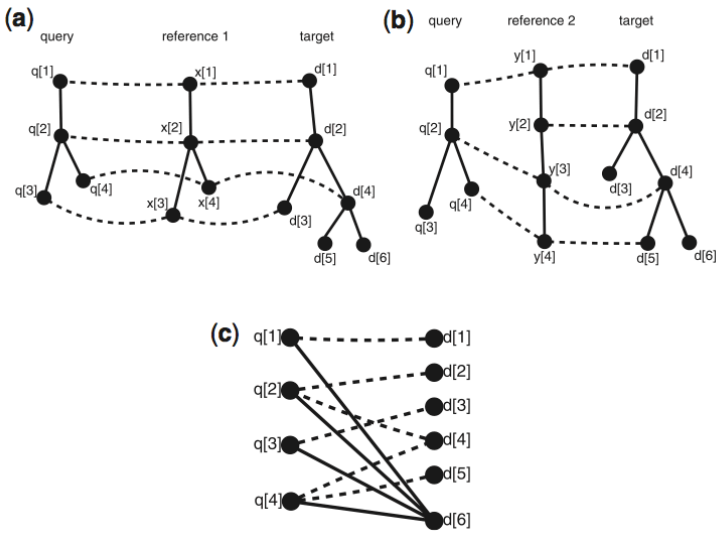
3.3.2 Üst Sınır Hesabı

Üst sınır hesabı bu yöntemin zayıf noktalarındandır çünkü kesin bir sonuç yerine yaklaşık bir sonuç vermektedir. Bölüm 3.2.2'de doğruluk değeri hesaplanırken de görüldüğü gibi, filtre setinde yanlış sonuçların da bulunabilmesi bunu göstermektedir. Üst sınır hesabı da alt sınır hesabı gibi iki aşamada çalışmaktadır. İlk aşamada q ve d'nin birbirleriyle eşleşen düğümleri tüm referanslar üzerinden bulunmakta, ikinci aşamada ise en yüksek puanlı hizalama işlemi yapılmaktadır. Bu aşamalar daha ayrıntılı olarak şu şekildedir;

Birinci Aşama: Öncelikle Bölüm 3.3.1'de olduğu gibi (q, r_i) ve (r_i, d) ikilileri arasındaki eşleşmelerden faydalanılarak r_i üzerinden (q, d_i) arasındaki eşleşmeler bulunur. Alt sınır hesabında bu eşleşmelerden en yüksek puanlı olan seçilmekteydi. Bu sefer ise bu eşleşmelerin hepsi kullanılarak birleşik eşleşme[9] yapılmaktadır. Yani

[8] ing: Breadth-First Search (BFS)
[9] ing: joint mapping

ϕ her d[j]'yi, dolaylı olarak en az bir referans aracılığıyla eşleşen q' yla birleşik olarak eşleştirmektedir. Bazı durumlarda q veya d' de herhangi bir düğümle eşleşmeyen düğümler kalmış olabilir. Bu tip d[j] (q[j]) düğümleri de karşı taraftaki tüm q (d) düğümleriyle eşleştirilir. [5]'den alınan Şekil 3.2'de, anlatılan durum bir örnekle gösterilmektedir. Bu örnekte iki tane referans ağı kullanılmıştır. Şekil 3.2 (a) ve (b)'de referans 1 ve referans 2 aracılığıyla sorgu ağı q' nun veritabanı ağı d'yle eşleşmesi gösterilmektedir. Şekil 3.2(c)'de ise ϕ bağıntısı aracılığıyla en son oluşan q ve d arasındaki birleşik eşleşmeler gözükmektedir. Şekilde d[4] düğümünün q[2] ve q[4]'le eşleştiği görülür. Bunun sebebi d[4]'ün referans 1 aracılığıyla yaptığı eşleşmede q[4]'le, referans 2 aracılığıyla yaptığı eşleşmede ise q[2]' yle eşleşmesidir. Daha fazla referans ağı kullanıldığında ise bu eşleşmelerin sayısı artacaktır. Ayrıca şekilde görüldüğü gibi d[6] düğümü tüm q düğümleriyle eşleşmektedir. Referans 1 ve referans 2 aracılığıyla yapılan eşleşmelerde herhangi bir düğümle eşleşmemiş olması sebebiyle bu yapılmıştır.



Şekil 3.2 Üst Sınır Hesabı [5]

İkinci Aşama: Bu aşamada, birinci aşamada elde edilen eşleşmeler hizalanmaktadır. ϕ eşleşmeleri ağırlıklı iki parçalı çizge[10] olarak kabul edilir. q ve d'nin düğümleri bu çizgenin düğümleri, ϕ eşleşmeleri de kenarlarıdır. Bu durumda üst sınır, UB(q, d)' yi

[10] ing: weighted bipartite graph

hesaplamak için q ve d arasındaki en yüksek ağırlıklı iki parçalı çizge[11] yöntemi kullanılır.

Bu anlatılanlar RINQ yönteminin ayrıntılarıdır. Yukarıda da belirtildiği gibi bu yöntemin açık kaynak kodunun bulunmaması sebebiyle yöntem Java diliyle kodlanmış, yapılan deney ve yöntemlerin karşılaştırılmasında da bu algoritmadan çıkan sonuçlar kullanılmıştır.

[11] ing: maximum weighted bipartite graph

4 QNET HADOOP GERÇEKLEŞTİRİMİ

Bu bölümde QNET' in performansını arttırmak için uygulanan yaklaşımlar anlatılmaktadır. Bölüm 2.2' de belirtildiği gibi QNET algoritması ESBiD ve RINQ yöntemlerinin yapı taşlarındandır. Bu algoritmanın performansındaki artış, yöntemlerin performansını doğrudan etkileyecektir. Açık kaynak kodu bulunmayan QNET algoritması RINQ kapsamında C++ koduyla kodlanmış ve çoklu iş parçacığı [12] mantığıyla çalıştırılmaktadır. Bu kod RINQ yazarları tarafından tarafımıza verilmiştir.

Çalışmanın bu bölümünde öncelikle QNET algoritması JAVA diliyle tekrar kodlanmıştır. Algoritmanın performansını arttırmak amacıyla doğruluğundan herhangi bir ödün verilmeden çalışma süresinde büyük bir gelişme kaydedilmiştir.

QNET' te yapılan ikinci geliştirme ise algoritmanın Hadoop' a uyarlanmasıdır. QNET Java gerçekleştirimi Hadoop çatısında tekrar programlanmış ve dağıtık olarak farklı yapılandırmalarda denenmiştir. QNET Hadoop gerçekleştirimi için 1 master makinayla birlikte 3, 6, 8 ve 9 slave makine kullanılmış, tüm bu farklı yapılandırmaların çalışma zamanları kaydedilmiştir. Kullanılan slave makinaların her birine 2' şer map görevi verilmiş, her makinada 2' şer çekirdek çalışmıştır.

Çizelge 4.1' de 7 ve 8 düğümlü ağların tüm farklı denemelerde ortaya çıkan çalışma zamanları verilmektedir. Çizelge 4.2'de ise 1, 2, 4 ve 8 farklı sorgu ağının aynı anda tüm veritabanıyla QNET kullanılarak Hadoop altyapısında hizalanma süreleri verilmektedir. Ayrıca bu çizelgede Hadoop öbeğinde kullanılan makina sayısının performansa etkisi de gözlemlenebilmektedir. Son olarak Çizelge 4.3'de sorgu ağındaki düğüm sayısının artmasıyla algoritmanın çalışma zamanı arasındaki karşılaştırma vardır. Bu karşılaştırma 10 makinadan (1 master, 9 slave) oluşan Hadoop öbeği kullanılarak yapılmıştır. Tüm bu denemeler için Intel Core i5 2320,

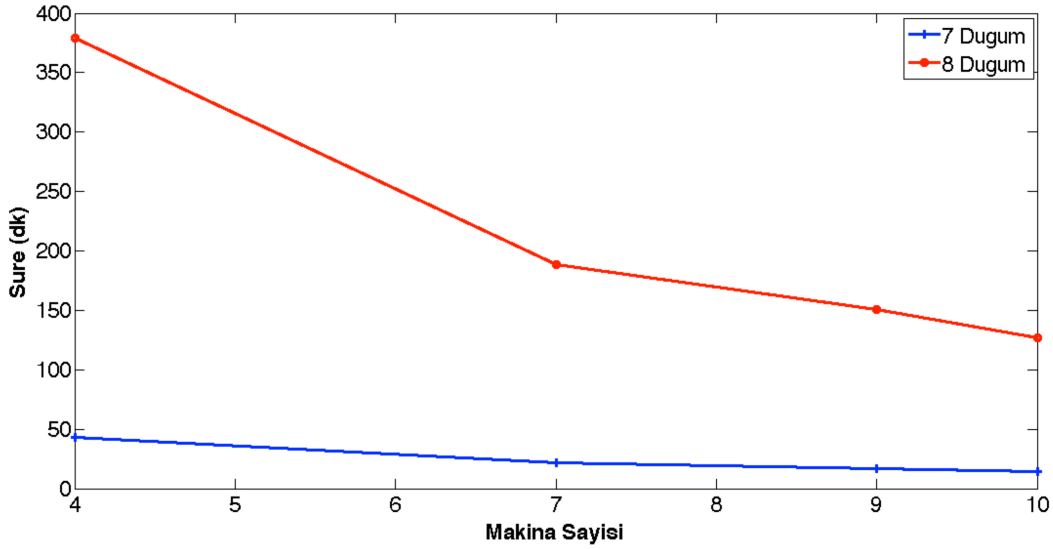
[12] ing: multithreading

3.00 GHz işlemcili, 4 GB RAM'a sahip, Ubuntu işletim sistemli makinalar kullanılmıştır.

Çizelge 4.1 QNET Çalışma Zamanı Karşılaştırması

	C++ 1 Makina	**Java 1 Makina**	**Hadoop 4 Makina (6 Çekirdek)**	**Hadoop 7 Makina (12 Çekirdek)**	**Hadoop 9 Makina (16 Çekirdek)**	**Hadoop 10 Makina (18 Çekirdek)**
7 Düğüm	6:35:50	2:47:50	43:22	21:50	17:09	14:42
8 Düğüm	56:28:4	25:17:55	6:19:01	3:08:54	2:30:32	2:07:02

Çizelge 4.1' de görüldüğü gibi çalışma zamanı açısından QNET' in C++ gerçekleştirimiyle Java gerçekleştirimi arasında 7 düğümlü bir sorgu ağıyla tüm veritabanı karşılaştırıldığında yaklaşık 3 saat 50 dakikalık fark vardır. Bu fark, sorgu ağındaki düğüm sayısı 8'e çıktığında yaklaşık 31 saat 15 dakikaya çıkmaktadır. Hadoop kullanıldığında ise birçok makinanın aynı anda çalışması sebebiyle algoritmanın çalışma süresinde çok büyük bir gelişme gözlemlenmektedir. C++ gerçekleştiriminde 6 saat 35 dakika 50 saniye olan, Java'da 2 saat 47 dakika 50 saniye olan çalışma süresi, 10 makinalı Hadoop çatısında 14 dakika 42 saniyeye düşmektedir. Bunun yanında yine C++'da 56 saat 28 dakika 4 saniye olan, Java'da 25 saat 17 dakika 55 saniye olan 8 düğümlü sorgu ağıyla karşılaştırma süresi ise 2 saat 7 dakika 2 saniyeye düşmektedir. Bu sürelere ek olarak, Hadoop' da kullanılan makina sayısına göre çalışma süresindeki değişimler çizelgede gösterilmektedir.



Şekil 4.1 Hadoop Makina Sayısının Çalışma Süresine Etkisi

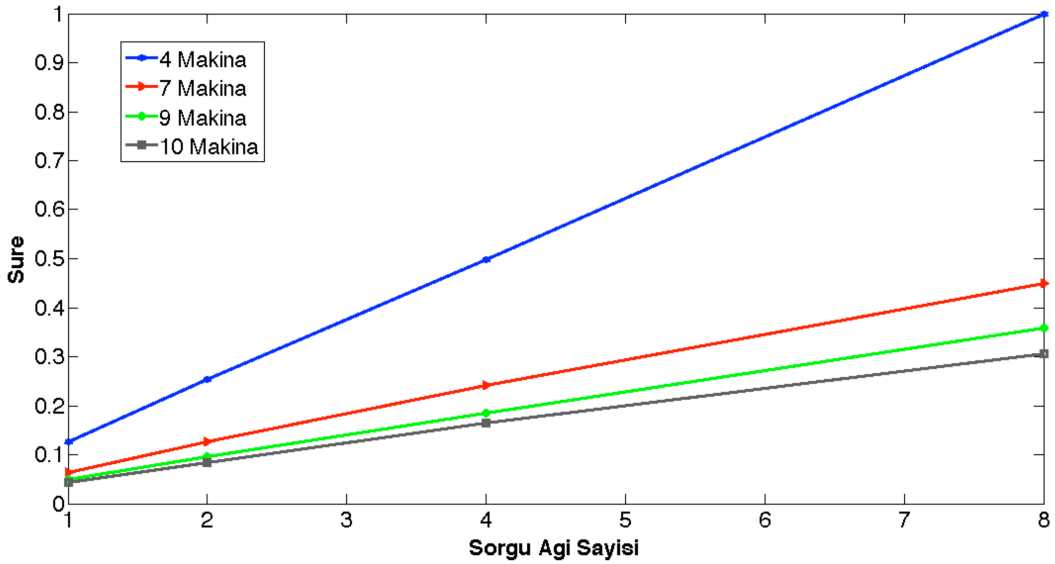
Şekil 4.1' de Hadoop' da kullanılan makina sayısının artmasıyla QNET' in çalışma süresinin nasıl değiştiği görülmektedir. Grafikte 7 ve 8 düğümlü sorgu ağları kullanılmıştır.

Çizelge 4.2' de ise Hadoop kullanıldığında birden fazla sorgu ağının aynı anda tüm veritabanıyla karşılaştırılma süreleri verilmektedir. Bu çizelgede 4, 7, 9 ve 10 makinalı öbekler kullanılmış, her yapılandırmada makinalardan 1 tanesi master, diğerleri slave olarak görev yapmıştır.

Çizelge 4.2 QNET Hadoop: 7 Düğümlü Sorgu Ağı Sayısının Performansa Etkisi

	1 Sorgu Ağı	**2 Sorgu Ağı**	**4 Sorgu Ağı**	**8 Sorgu Ağı**
4 Makina	43:22	1:27:21	2:50:59	5:43:15
7 Makina	21:50	43:19	1:22:53	2:34:30
9 Makina	17:09	33:02	1:03:03	2:02:44
10 Makina	14:42	28:53	56:11	1:45:07

Çizelge 4.2' deki bilgiler ışığında çizilen grafik Şekil 4.2' de görülmektedir. Bu grafikte kullanılan süreler en yüksek zamanlı çalışma süresine göre normalize edilmiş değerlerdir.



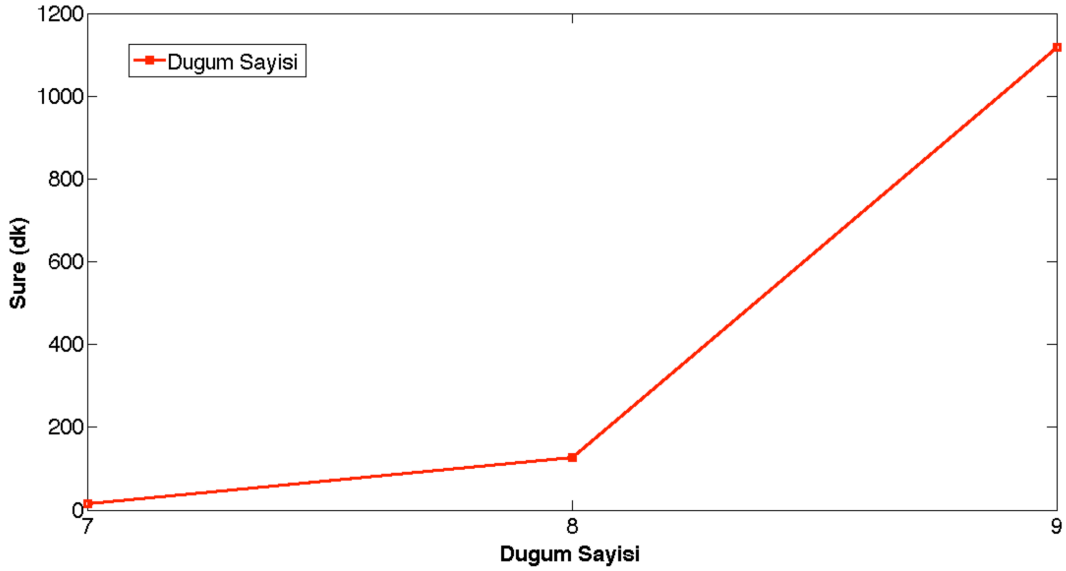
Şekil 4.2 Hadoop Sorgu Ağı Sayısının Çalışma Süresine Etkisi

Çizelge 4.3'te ise sorgu ağında kullanılan düğüm sayısının, 10 düğümlü Hadoop öbeği çerçevesinde hizalanma süreleri verilmektedir.

Çizelge 4.3 QNET Hadoop: Düğüm Sayısındaki Artışın Etkisi

	7 Düğüm	**8 Düğüm**	**9 Düğüm**
Hadoop (10 makina)	14:42	2:07:02	18:38:26

Çizelgede görüldüğü gibi sorgu ağında kullanılan düğüm sayısındaki artış, QNET' in çalışma süresini hızla yükseltmektedir. 8 düğümlü sorgu ağı kullanıldığında yaklaşık 2 saat süren algoritmanın çalışma süresi, düğüm sayısı 9 olduğunda yaklaşık 18,5 saat sürer. Bu artış, Şekil 4.3' de grafiksel olarak görülmektedir.



Şekil 4.3 Düğüm Sayısındaki Artışın Çalışma Zamanına Etkisi

Düğüm sayısındaki bir düğümlük artışın, çalışma zamanı üzerindeki etkisinin bu kadar yüksek olması, 7' den fazla düğüme sahip bir ağ kullanıldığında Hadoop alt yapısının kullanılmasını gerektirmektedir. Özellikle 9 ve üzerinde düğüm kullanıldığında tek makinanın çalışma zamanı çok uzun sürecektir.

QNET' le yapılan tüm hesaplamalarda kullanılan sorgu ağının kenar sayısı sabittir. Bu sayı 7 düğümlü ağlarda 6, 8 düğümlü ağlarda 7 ve 9 düğümlü ağlarda 8'dir. Bunun sebebi QNET' in dönüşe[13] izin vermemesidir.

Paralel hesaplamanın sıralı hesaplamaya göre verimliliğini ölçmek için 4.1'de verilen denklemde gösterilen hızlanma[14] parametresi, S_p, literatürde yer almaktadır.

$$S_p = \frac{T_1}{T_p} \tag{4.1}$$

[13] ing: cycle
[14] ing: speedup

Bu denklemde p, işlemci sayısını, T_1, sıralı algoritmanın çalışma süresini, T_p ise p işlemcili paralel hesaplama algoritmasının çalışma süresini göstermektedir.

Ayrıca verimliliği ölçen başka bir parametre, E_p, 4.2' de verilen denklemle hesaplanır.

$$E_p = \frac{S_p}{p} \tag{4.2}$$

Çizelge 4.4' de Hadoop' da hızlanma ve verimlilik hesabının sonuçları verilmektedir. Bu hesaplar 7 ve 8 düğümlü sorgu ağlarının kullanımı içindir. Hesaplarda hızlanma ve verimlilik QNET Java gerçekleştirimine göre yapılmıştır. Verimlilik hesabında p değeri Hadoop' da çalışan map görevi olarak alınmıştır, bu da her makina için 2'dir ve çekirdek sayısına eşittir. Örneğin 4 makinalı Hadoop öbeğinde 1 master, 3 slave çalışmaktadır ve Hadoop işi 6 map görevine bölünmüş, her slave makinaya 2'şer map görevi verilmiştir. Bu durumda 4 makina için verimlilik hesabı yapılırken hızlanma parametresi 6'ya bölünmüştür.

Çizelge 4.4 Hadoop Hızlanma ve Verimlilik Tablosu

		4 Makina (6 Çekirdek)	**7 Makina (12 Çekirdek)**	**9 Makina (16 Çekirdek)**	**10 Makina (18 Çekirdek)**
7 Düğüm	***Hızlanma***	3,87	7,67	9,79	11,42
	Verimlilik	0,65	0,64	0,61	0,63
8 Düğüm	***Hızlanma***	4,00	8,04	10,08	11,95
	Verimlilik	0,67	0,67	0,63	0,66

QNET' le ilgili yapılan çalışmaların tümü göz önüne alındığında şu sonuçlar ortaya çıkmaktadır;

- 7 düğümlü bir sorgu ağı kullanıldığında QNET JAVA kullanımı 3 saate yakın sürmektedir. Bu süre çok uzun değildir ancak ESBiD yöntemiyle

birlikte kullanılırsa, uygulamanın çalışmasıyla birlikte veritabanında hizalanacak ağ sayısı azaldığı için benzerlik hesabı daha hızlı ve verimli yapılabilecektir.

- 7 düğümlü sorgu ağı Hadoop' la birlikte kullanıldığında ise ESBiD, RINQ gibi yöntemlere gerek kalmadan da tam hizalama verimli bir şekilde yapılabilmektedir. Bunun için Hadoop öbeğinde kullanılan makina sayısı çok önemlidir. 10 makinalı bir Hadoop öbeğinde tüm veritabanıyla hizalama yapmak 14 dakika 42 saniye sürmektedir. Bu süre, makina sayısının artmasıyla daha da kısalacaktır.

- Sorgu ağındaki düğüm sayısının artmasıyla ESBiD yöntemini kullanmak gerekecektir. Tüm veritabanıyla tam hizalama 10 makinalı Hadoop öbeği kullanıldığında 8 düğüm için yaklaşık 2 saat, 9 düğüm kullanıldığında ise 18,5 saat sürmektedir. Bu süreler tolere edilemeyecek kadar uzun sürelerdir. Bu durumda sorgu ağında kullanılan düğüm sayısındaki artışın etkisinin çok fazla olduğu gözlemlenebilir. Hadoop öbeğinde kullanılan makina sayısının arttırılması, bir noktaya kadar bunu dengeleyebilir.

- Hadoop kullanımı, birden çok sorgu ağıyla aynı anda tüm veritabanını hizalamayı da kolaylaştırır. Bunun sonuçları Çizelge 4.2'de verilmektedir.

5 REFERANS SEÇİMİ

Referans tabanlı ağ hizalama yönteminin en temel adımı referans seçimidir. Bölüm 3.2'de RINQ' da bu seçimin nasıl yapıldığı ayrıntılarıyla anlatılmaktadır. Bu çalışmada ise farklı bir yaklaşım oluşturulmuş ve ağların biyolojik özellikleri kullanılmıştır.

Bir kümeden ya da topluluktan referans seçmek, o referansın ya da referansların, topluluğu temsil etmesini gerektirir. Seçilen referanslara bakıldığında referanslardan o topluluk görülebilmelidir. Bu, canlının her hücresinde aynı DNA'nın bulunması ve bu DNA yapısının da insanın özeti olması gibidir. DNA yapısına bakıldığında o DNA'ya sahip insanı görmeden kişi hakkında bilgi sahibi olmak, daha net bir ifadeyle o insanı görmüş gibi olmak gerekmektedir. Yani özetten bütün anlaşılabilmeli, özet bütünü temsil edebilmeli ve özet bütünün sahip oldukları dışında herhangi bir şey söylememelidir. Referans tabanlı ağ hizalama yönteminde de mümkün olduğu kadar bu yaklaşıma uygun referans ağı seçimi yapılmalıdır. Seçilen referans ağların veritabanını ne kadar doğru temsil ettiği, referans seçiminin ne derece doğru yaptığının göstergesidir.

Biyolojik veri tabanları daha önce de bahsedildiği gibi biyolojik ağlardan oluşmaktadır. Bu biyolojik ağların birçok ortak ve farklı özellikleri mevcuttur. Referans seçimi yapılırken bu özellikleri göz önünde bulundurmamak seçimin eksik yapılmasına ve referans setinin veritabanını tam temsil edememesine sebebiyet verecektir. Bu durumda veritabanındaki biyolojik ağların biyolojik özelliklerinden faydalanılarak bir referans seti oluşturmak daha doğru sonuçlar vermelidir.

Bölüm 1' de belirtildiği gibi, uygulamada kullanılmak üzere KEGG veritabanından 297 tane gen düzenleyici ağ seçilerek bir veritabanı oluşturulmuştur. Bu veritabanındaki ağların biyolojik yolakları farklılık göstermektedir. Bunlar gruplandığında 21 tane farklı grup oluşmaktadır. Bu gruplardaki biyolojik ağ sayısı ve ilgili yolakların kısa tanımı aşağıda verilmektedir;

- **WNT Sinyal Yolağı**: Bu yolak proteinlerden oluşur. Hücrenin dışındaki sinyalleri hücre yüzeyindeki reseptörler vasıtasıyla hücre içine taşır. Veritabanında 36 tane bu tip ağ vardır.

- **Kalsiyum Sinyal Yolağı**: Canlılarda gerçekleşen pek çok önemli olayda kalsiyum iyonları belirleyici rol oynamaktadır. Bu olaylar arasında hareket, kalp atışı, beynin bilgiyi isleyip hafızayı oluşturması, yumurtanın döllenme sonucu aktivasyonu, pankreatik hücrelerde salgılama, yaraların iyileşmesi sayılabilir [34]. 11 tane bu tip ağ veritabanında bulunur.

- **Toll-benzeri Reseptör Sinyal Yolağı**: Bu yolak belli patojenlerle ilişkili moleküler kalıpları tanır ve bağışıklık sistemi yanıtlarında önemli rol oynar. İşgalci patojenlere karşı ilk savunma mekanizmasıdır. Ayrıca iltihap, imün hücre düzenleme, üreme ve hayatta kalmada önemli rol oynar [35]. 14 tane bu tip ağ veritabanında mevcuttur.

- **MAPK Sinyal Yolağı**: Hücrenin yüzeyindeki reseptörler tarafından gelen sinyalleri hücre çekirdeğindeki DNA'ya ileten protein zincirleridir [36]. Bu tip ağ sayısı veritabanında 18'dir.

- **mTOR Sinyal Yolağı**: Hücre büyümesi, hücre çoğalması, hücre hareketi, hücre canlılığı, protein sentezi ve transkripsiyon görevlerini düzenler [37]. 10 tane ağ bu tiptedir.

- **VEGF Sinyal Yolağı**: Embriyodaki damar gelişimini ve yeni kan damarı oluşumunu düzenler [38]. Veritabanındaki bu tip ağ sayısı 14'tür.

- **T Reseptör Sinyal Yolağı**: Veritabanında 10 tane ağa sahip bu yolak tipi, bağışıklık sisteminde önemli rol oynar.

- **Adipocytokine Sinyal Yolağı**: Adipocytokine, yağ doku tarafından salgılanan sitokinlerdir. Veritabanında bu yolak 16 tane ağda görülür.

- **İnsülin Sinyal Yolağı**: Bu yolağın görevlerinden bazıları şöyledir: Glikoz depolama ve içeri alımı, protein sentezi, lipid sentezinin düzenlenmesi, mitojenik tepkiler [39]. 10 tane ağ veritabanında bu tiptedir.

- **B Reseptör Sinyal Yolağı**: B hücre reseptörleri, B hücrelerinin yüzeyinde bulunan transmembran reseptör proteinlerdir [40]. Veritabanında bu yolağı kullanan 9 adet ağ bulunur.

- **NOD-benzeri Reseptör Sinyal Yolağı**: NOD-like reseptörler, kalıp tanıma reseptörlerinin bir bölümüdür ve doğuştan gelen bağışıklık sisteminin düzenlenmesinde önemli rol oynarlar [41]. Veritabanında 12 biyolojik ağ bu yolağı kullanır

- **Fc Gamma R-Aracılı Fagositoz**: Bağışıklık sistemiyle ilgilidir. Bu grupta 9 tane ağ bulunur.

- **Jak-STAT Sinyal Yolağı**: Hücre dışındaki kimyasal sinyalleri hücre zarı üzerinden DNA'daki gen promotörlerine ulaştırır. Bu da DNA transkripsiyonuna ve hücre aktivitesine sebebiyet verir [42]. Bu grupta 15 tane ağ vardır.

- **RIG-I-benzeri Reseptör Sinyal Yolağı**: Hücre arası kalıp tanıma reseptörleri cinsindendir. Doğuştan gelen bağışıklık sistemi sayesinde virüslerin tanınmasında rol alır [43]. 5 tane ağ bu grupta yer almaktadır.

- **PPAR Sinyal Yolağı**: Gen ifadesini düzenleyen ve transkripsiyon faktörü olarak görev yapan nükleer reseptör proteinlerdir [44]. Veritabanında bu tipte 16 ağ bulunur.

- **Fosfatidilinozitol Sinyal Yolağı**: Fosfatidilinozitol pı, hücre zarlarında ek grup olarak inozitol taşıyan bir fosfolipit; ökaryotik hücrelerde yüksek enerjili fosfat bağlanmasıyla ikincil haberci olarak görev yapan moleküldür [45]. Veritabanında 45 tane bu tip ağ vardır.

- **ErbB Sinyal Yolağı**: ErbB1, ErbB2,ErbB3 ve ErbB4 adında 4 farklı üyesi olan reseptör tirozin kinazdır. Yetersiz ErbB sinyalleşmesi insanlarda multipl skleroz ve Alzheimer hastalığı gibi nörodejenerasyon hastalığının gelişimiyle ilgilidir [46]. Farelerde ise kalp, beyin, deri ve akciğerde hasarlar oluşturan embriyo ölümlerine sebep olur. Fazla sinyalleşme ise çeşitli katı tümör tiplerinin oluşmasına sebebiyet verir. Bu grupta 18 tane ağ bulunur.

- **MAPK Sinyal Yolağı – maya**: Bu grupta tek bir ağ bulunmaktadır. İşlevi yukarıda anlatılan MAPK Sinyal Yolağının maya için olanıdır.

- **GnRH Sinyal Yolağı**: Folikül uyarıcı hormonun, lüteinleştirici hormonun ve ön hipofiz hormonunun salınımını kontrol eder. Hipotalamus' daki nöronlar tarafından sentezlenir. GnRH aktivitesi çocuklukta çok düşüktür, ancak ergenlikle beraber artar. 3 tane ağ bu tiptedir.

- **Fc Epsilon RI Sinyal Yolağı**: Alerji düzensizlikleri ve parazit bağışıklığında rol oynar. Bu yolağı kullanan veritabanında 10 tane ağ vardır.

- **Kemokin Sinyal Yolağı**: Kemokin, doku ve hücreler tarafından üretilen, fagositik hücreler ve lenfositler gibi bağışıklık sistemi hücrelerinin kemotaksisini ve aktivasyonunu uyaran, ayrıca integrine bağlı akyuvar

yapışmasını tetikleyen maddelerdir [47]. Bu yolağı veritabanında bulunan 11 ağ kullanır.

Yukarıda verilen yolaklara sahip ağlar, her grup farklı bir yolak olmak üzere gruplanmış olur. Referans oluşturulurken RINQ' da, veritabanından rastgele bir ağ seçilip, o ağdan da rastgele bir düğüm seçiliyordu ve bu şekilde diğer düğümler de seçilen düğümlerin komşuları arasından seçiliyordu. Bu yöntemde ise 21 tane farklı grup, ağların sahip olduğu yolaklardan oluşmaktadır. Her gruptan en az bir ağ seçilerek referans oluşturulmaya başlanır. Bu durumda en az 21 tane referans olacak ve her grubu temsil eden mutlaka bir referans, tüm referanslar arasında bulunacaktır. Her gruptan seçilen referans sayısının artması veya her gruptaki ağ sayısına göre referans sayısının belirlenmesi sonuçların doğruluğunu arttıracaktır.

Her gruptan en az bir ağ seçildikten sonra, aynı RINQ yönteminde olduğu gibi, o ağdan rastgele bir düğüm seçilip, o düğümün komşularından rastgele ikinci bir düğüm seçilerek ağ oluşturma işlemi devam eder. Sonunda referans seti oluşturulur ve referans seti üzerinden benzerlik hesabı yapılabilir.

6 İKİNCİ ÜST SINIR BELİRLEME VE SEZGİSEL YAKLAŞIM

RINQ yönteminin temel zayıflıklarından biri üst sınır hesabıdır. Yöntemde alt sınır hesabı tam sonuç vermesine rağmen, Bölüm 3.3.2'de ayrıntılı olarak anlatılan üst sınır hesabı yaklaşık sonuç vermektedir. Bu da sonuçlarda sapmalara ve verimsizliklere sebep olabilmektedir. Böylece üst sınır hesabının tam sonuç vermemesi sebebiyle filtre setinde yanlış sonuçların yanında doğru sonuçlar da olacak, buna ek olarak belirsizlik setinde gereğinden fazla ağ birikecektir.

Daha önce anlatıldığı gibi RINQ yönteminin sonunda oluşturulan setler şu şekilde belirlenmektedir;

- Filtre Seti: $UB(q, d_i) < \varepsilon$
- Sonuç Seti: $\varepsilon \leq LB(q, d_i)$
- Belirsizlik Seti: $LB < \varepsilon \leq UB(q, d_i)$

Belirsizlik setindeki ağ sayısını fazlalığının önüne geçmek amacıyla yönteminin kullandığı üst sınır hesabına ek olarak farklı bir yöntemle ikinci bir üst sınır hesabı yapılmaktadır. Bu yöntem RINQ yönteminde kullanılan üst sınır hesabı yöntemini ortadan kaldırmamakta, ona ek olarak başka üst sınır hesabı daha eklemektedir. Böylelikle iki yöntem birlikte çalışmaktadır. Daha açık bir ifadeyle, RINQ yönteminde kullanılan üst sınır ve alt sınır hesabı aynı şekilde yapılmakta, ağlar filtre seti, sonuç seti ve belirsizlik seti olarak üç gruba ayrılmaktadır. Bu esnada her ağ için hesaplanan ikinci üst sınır değeri kullanılarak sezgisel bir yöntemle belirsizlik setindeki ağlar tek tek tekrar kontrol edilmekte ve bir kısmı belirsizlik setinden çıkartılarak sonuç setine eklenmektedir.

6.1 İkinci Üst Sınır Hesabı

İkinci üst sınır hesabı, sorgu ağı q' yla, veritabanı ağı d_i arasında mümkün olan en yüksek üst sınırı bulur. Bu üst sınır yaklaşık değil kesin bir üst sınırdır. Bu da şu

şekilde hesaplanır; q ve d arasında benzerlik hesaplanırken aşağıdaki formülün kullanıldığı Bölüm 3'de anlatılmaktadır.

$$\begin{aligned} \text{sim(q, d)} = & \sum_{\beta(q[i])\neq 0} sim(q[i], \beta(q[i])) \\ & + \sum_{\beta(q[i])=0} Indel\ Penalty \\ & + \sum_{\substack{(\beta(q[i]),\beta(q[j])\in E_d \\ AND\ (q[i],q[j])\in E_q}} EW_d(\beta(q[i]), \beta(q[j]) \end{aligned} \quad (5.1)$$

Bu formülde kullanılan $\sum_{\beta(q[i])\neq 0} sim(q[i], \beta(q[i]))$ ifadesinin hesabı, daha önce hesaplanan ve saklanan benzerlik dizeyi kullanılarak yapılmaktadır. İkinci üst sınır hesabı yapılırken de benzer şekilde her düğüm için eşleşebileceği en yüksek benzerlikli düğüm, Algoritma 2'deki gibi seçilmektedir. Bunun için de yine daha önce oluşturulan benzerlik dizeyleri kullanılır.

Algoritma 2: *İkinci Üst Sınır Hesabı*

```
Girdi: q, d_i, benzerlik dizeyleri
Çıktı: q - d_i arasındaki benzerlik üst sınır değeri
en_buyuk_benzerlik=0
toplam=0
for Sorgu ağı q'daki her düğüm için
    for Veritabanı ağı d_i'nin her düğümü için
        if sim(q[i],d[j]) > en_buyuk_benzerlik
            en_buyuk_benzerlik= sim(q[i],d[j])
            break
    toplam+= en_buyuk_benzerlik
    en_buyuk_benzerlik=0
```

Bunun sonunda q ve d_i arasında ikinci bir üst sınır değeri bulunur.

6.2 Sezgisel Yaklaşım

İkinci üst sınır hesaplandıktan sonra, oluşturulan sezgisel bir yaklaşım kullanılır ve bu sayede belirsizlik setindeki bazı ağlar sonuç setine geçer. Kullanılan sezgisel yaklaşım iki temel şarta dayanarak belirsizlik setini azaltmaktadır. Bu şartların

temelinde referans seçme yönteminde kullanılan biyolojik yolaklar vardır. Aynı biyolojik yolakları kullanan ağların benzerliğinin daha yüksek olacağı düşünülmektedir ve oluşturulan sezgisel yaklaşımın kaynağında bu vardır. Aşağıdaki iki şarttan herhangi birisi sağlanırsa sorgu ağı q ve veritabanı ağı d benzer kabul edilmekte ve benzerlik setine alınmaktadır.

- q ve d_i'nin kullandığı yolaklar aynı ve hesaplanan ikinci üst sınır değeri, mümkün olan en yüksek değere eşitse. (Bu değer 7 düğümlü ağlar için 700, 8 düğümlüler için 800'dür)
- d_i' nin ikinci üst sınırının yarısı, benzerlik eşik değerinden büyükse.

Şartlardan ilki olan q ve d_i' nin aynı yolakları kullanması, biyolojik verilerin benzerlik hesabında önemli bir yere sahip olduğunu vurgulamaktadır. Biyolojik ağlardaki benzerlikleri hesaba katmadan iki ağın benzerliğini bulmak eksik kalacaktır. Bu durumda aynı biyolojik yolağa sahip ağların benzerliği fazla olacaktır. Buna bağlı olarak hesaplanan ikinci üst sınır değerinin 700'e eşit olup olmadığı kontrol edilmektedir. Bu değer 7 düğümlü q ve d_i arasında bulunabilecek en yüksek üst sınır olarak kabul edilmiş ve aynı biyolojik yolağa sahip, ikinci üst sınır değeri 700 olan ağların benzer olduğu görülmüştür. Bu yaklaşımda, iki düğüm arasındaki BLAST sonucuna göre oluşabilecek en yüksek benzerlik değerinin 100 olduğu varsayımı kullanılmaktadır.

İkinci şart ise d_i' nin ikinci üst sınır değerinin yarısının, benzerlik eşik değerinden büyük olup olmadığının kontrolüdür. Görülmüştür ki ikinci üst sınır değerinin yarısı benzerlik eşik değerinden büyükse iki ağ genellikle benzer olmaktadır.

Bu şartlar göz önünde bulundurularak yapılan denemelerin sonuçları aşağıda verilmektedir. Bu denemeler için 20 farklı sorgu ağı kullanılmış, belli eşik değerlerine göre sonuç, filtre ve belirsizlik setinde oluşan ağ sayısı, belirsizlik setindeki ağlardan kaç tanesinin sezgisel yaklaşımla sonuç setine geçtiği, sonuç setine geçen ağlardan ne kadarının doğru olduğu bilgisi verilmiştir. Çizelge ve

şekillerde kullanılan değerler 20 farklı sorgu ağıyla yapılan denemelerin ortalamasının alınmasıyla elde edilmiştir.

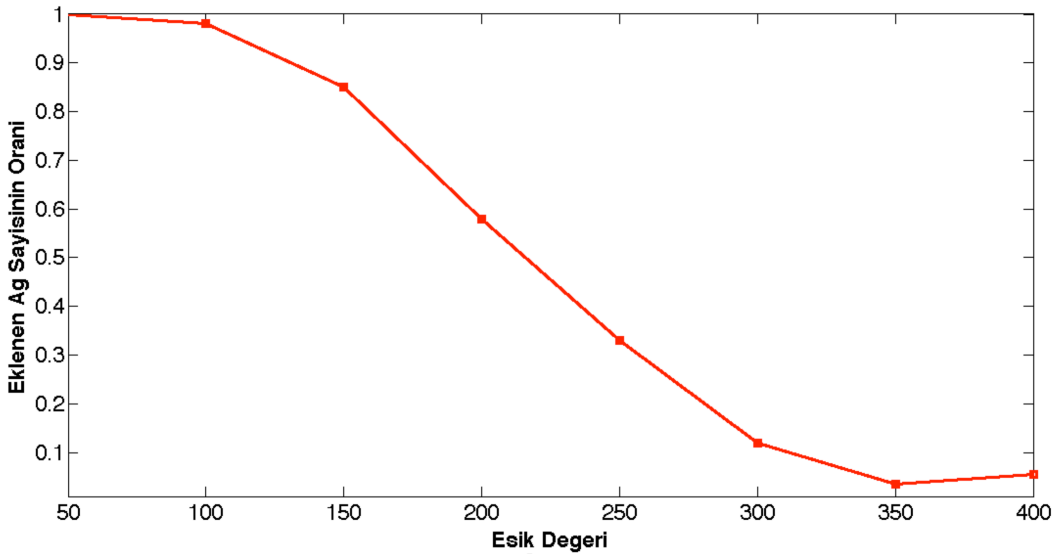
Çizelge 6.1 Sezgisel Yaklaşım Sonuçları

Eşik	Sonuç Seti	Filtre Seti	Belirsizlik Seti	Sonuç Seti'ne Eklenen Ağ Sayısı	Doğru	Yanlış
50	293,3	0	3,7	3,7	3,25	0,45
100	265,8	0	30,95	30,35	27,9	2,45
150	206,1	0,2	90,75	77,25	71,4	5,85
200	138,9	2,9	156,45	90,4	83,3	7,1
250	72,45	13,95	206,85	68,2	64,65	3,55
300	31,8	46,8	218,4	27,2	25,45	1,75
350	11,25	93,4	191,45	6,8	6,8	0
400	4,25	142	146,9	8,15	8,15	0

Çizelge 6.1' de öncelikle bir sorgu ağıyla yapılan hizalama sonucunda, sonuç, filtre ve belirsizlik setinde kaç tane ağ biriktiği gösterilmektedir. "Sonuç Setine Eklenen Ağ Sayısı" sütununda ise belirsizlik setinden sonuç setine kaç tane ağın eklendiği bilgisi vardır. Doğru ve yanlış sütunlarında da bu eklenen ağlardan kaç tanesinin gerçek benzerliğinin, eşik değerinden yüksek olduğu gösteriliyor. Bu doğrulama, uzun QNET metodu kullanılarak yapılmıştır. Ayrıca tablolardaki sonuçlarda +/- 10 puanlık sapmalara izin verilmektedir.

Sonuçlar göstermektedir ki eşik değeri 350 veya 400 olduğunda sonuç setine eklenen ağlarda yanlış yoktur, tüm eklenen ağlar doğrudur ve sonuç setine gerçekten eklenmelidir. Bunun sebebi şudur; sezgisel yaklaşımın iki şartından ikincisi, eşik değeri 350'den büyük alındığında geçersiz olmaktadır, yani "d_i' nin ikinci üst sınırının yarısı, benzerlik eşik değerinden büyükse" şartı 7 düğümlü bir ağ kullanıldığı ve en büyük benzerlik değeri 700 olduğu durumlarda kullanılamamaktadır. Sebebi de d_i'nin ikinci üst sınırının en fazla 700 olabileceği ve bu değerin yarısının da 350 olan eşik değerlerinden yüksek olamayacağıdır. Bu durum göstermektedir ki en yüksek üst sınır değerinin (7 düğüm için 700, 8 düğüm için 800) yarısından büyük eşik değerleri kullanıldığında sezgisel yaklaşımın sadece

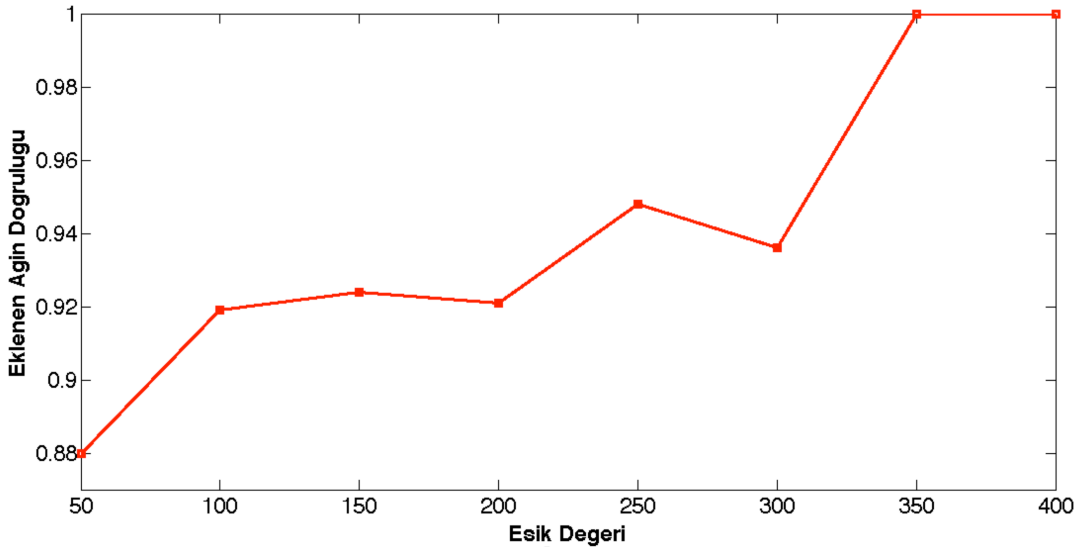
"q ve d_i'nin kullandığı yolaklar aynı ve hesaplanan ikinci üst sınır değeri, mümkün olan en yüksek değere eşitse" olan ilk şartı kullanılabilmektedir. Sezgisel yaklaşımın ikinci şartı her zaman doğru sonuç vermezken, bu şartın her zaman doğru sonucu verdiği gözlemler ve deneyler sonucunda söylenebilir. Bu durumda Çizelge 6.1' de yer alan ve sonuç setine eklenen ağların ne kadarının yanlış olduğunu veren bilgi sadece sezgisel yaklaşımın ikinci şartı için geçerlidir. İlk şartı her durumda doğru sonucu vermektedir.



Şekil 6.1 Eklenen Ağın Belirsizlik Setine Oranı

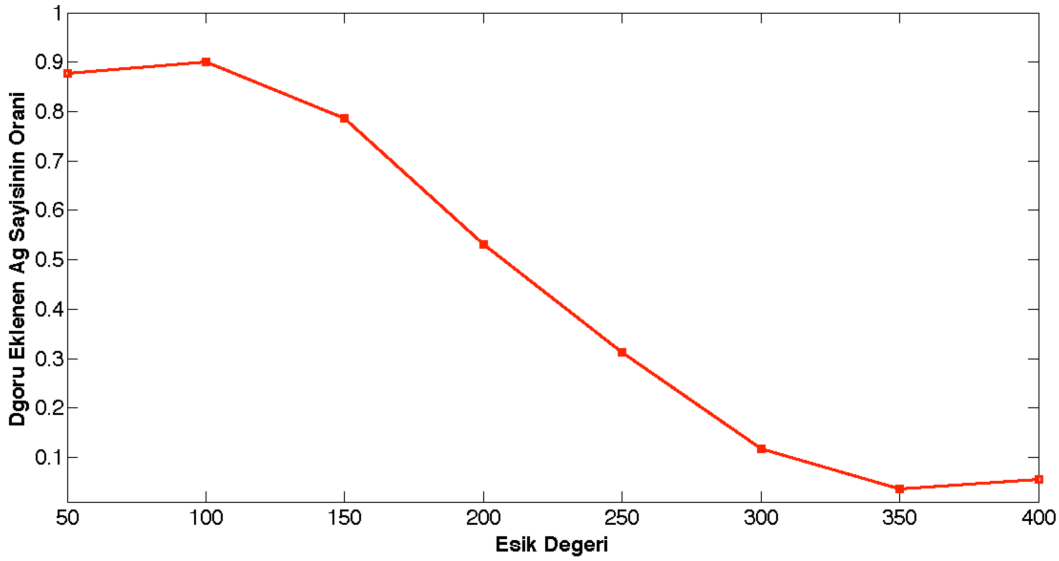
Şekil 6.1' de, sonuç setine eklenen ağ sayısının belirsizlik setindeki ağ sayısına oranı kullanılmış, eşik değerine göre grafiği çizilmiştir. Görüldüğü gibi eşik değerinin artmasıyla, belirsizlik setinden sonuç setine eklenen ağ sayısının oranı azalmaktadır. Bunun sebebi benzerlik değerinin eşik değerine göre düşük kalmasıdır.

Şekil 6.2'de ise sonuç setine eklenen ağların doğruluğu incelenmektedir. Eklenen ağın doğruluk değeri hesaplanırken eklenen doğru ağ sayısı, toplam eklenen ağ sayısına bölünmüştür. Çıkan sonuç göstermektedir ki en yüksek üst sınır değerinin yarısından daha büyük eşik değerleri için eklenen ağın doğruluğu her zaman 1'dir.



Şekil 6.2 Yeni Eklenen Ağın Doğruluğu

Şekil 6.3 ise eklenen ağların sonuç setine katkısının gösterilmesi amacıyla çizilmiştir. Bu grafikte doğru eklenen ağ sayısının belirsizlik setine oranı, eşik değerine oranla gösterilmektedir.



Şekil 6.3 Doğru Eklenen Ağ Sayısının Belirsizlik Setine Oranı

Grafikte görülmektedir ki sezgisel yaklaşım kullanılarak belirsizlik setinden sonuç setine eklenen ağ sayısı, eşik değeri arttıkça azalan bir eğilim göstermektedir. Bu da

gösteriyor ki kullanılan bu yaklaşım, eşik değeri düşükken başarılıdır ancak eşik değeri arttıkça başarısı azalmaktadır. Sebebi ise eşik değerinin artmasıyla filtre setine eklenecek ağ sayısının daha fazla olması, belirsizlik setinden sonuç setine eklenecek ağ sayısının da orantılı olarak azalmasıdır. Kullanılan yaklaşım sadece sonuç setine eklenecek ağları bulduğu, filtre setine bir katkı sağlamadığı için böyle bir grafik ortaya çıkmaktadır. Filtre setine katkı sağlayacak başka bir sezgisel yaklaşımın da kullanılması, eşik değeri arttıkça belirsizlik setinden filtre setine bazı ağların eklenmesine sebep olacaktır.

7 BELİRSİZLİK SETİ HESABI

RINQ yönteminin veritabanındaki ağları sonuç, filtre ve belirsizlik seti olarak üç farklı gruba böldüğünden Bölüm 3.1'de bahsedilmektedir. Sonuç ve filtre setinde biriken ağ sayısının fazla olması istenen bir sonuçtur ancak kullanılan eşik değerine göre, belirsizlik setinde de birçok ağ birikmektedir. Yöntem, burada oluşan ağların hizalanması için herhangi bir öneri getirmemektedir. RINQ' nun sağladığı fayda buraya kadardır. QNET' le hizalanması gereken ağ sayısı düşürülmüş, belirsizlik setindeki ağ sayısına indirgenmiştir. Bu setteki ağlar ise uzun ve maliyetli QNET yöntemiyle hizalanmaktadır. Bu durumda belirsizlik setinde biriken ağ sayısının artması, sorgu ağıyla veritabanının hizalanma süresini arttıracaktır. Bu sette oluşan ağ sayısının azlığı ya da çokluğu da, yöntemin iyi çalışıp çalışmadığının göstergelerinden biri olacaktır. RINQ' nun toplam çalışma zamanı, sorgu ağıyla veritabanı ağları arasında alt sınır ve üst sınır hesaplanma süresiyle, belirsizlik setindeki ağların QNET' le hizalanma sürelerinin toplamıdır.

Bu bölümde, belirsizlik setinde biriken ağların uzun ve maliyetli hizalanmasının önüne geçebilmek için bazı yaklaşımlar önerilmiş ve bu yaklaşımların sonuçlarının karşılaştırılması yapılmıştır. Buradaki amaç, sorgu ağıyla belirsizlik setindeki ağları hızlı ve mümkün olduğunca doğru bir şekilde hizalayarak yöntemin toplam çalışma zamanını düşürmektir.

Belirsizlik setindeki ağların sorgu ağıyla karşılaştırılması için önerilen yaklaşımlardan biri öbeklemedir. Öbekleme, belirsizlik seti hesabında Bölüm 7.1'de anlatıldığı gibi iki farklı şekilde kullanılmaktadır. Öbekleme dışında bu çalışma kapsamında "en yüksek dereceli düğüm" tekniği kullanılmaktadır. Bölüm 7.2'de anlatılan bu teknik 10, 12, 13, 15 ve 17 düğüm kullanılmak üzere 5 farklı şekilde denenmiştir. Yapılan denemelerin sonuçları çizelge ve grafiklerle gösterilmektedir.

7.1 Öbekleme Yöntemi

Bölüm 2.4'de öbeklemenin detayları verilmiş, literatürde bulunan öbekleme yöntemleri özetlenmiştir. Bu çalışmada kullanılan öbekleme yöntemi Voltage Clusterer' dır. JUNG [48] çizge kütüphanesinden alınan bu yöntemin temeli [49]'a dayansa da biraz daha farklıdır.

Bu çalışmada iki çeşit öbekleme yöntemi kullanılmıştır. Çok benzer olan iki yöntemin özellikleri aşağıda detaylandırılmaktadır.

7.1.1 Tek Kademeli Öbekleme

Belirsizlik seti hesabı için kullanılan tek kademeli öbekleme yönteminde öncelikle veritabanındaki ağlar kendi içinde öbeklere ayrılmaktadır. Burada yapılan öbekleme, veritabanındaki her d_i' deki düğüm sayısı göz önüne alarak yapılmıştır. Ortalama 8-10 düğümlük öbekler oluşturulmaya çalışılsa da bazı öbekler daha büyük olmuştur. Genel olarak bu yöntemle veritabanındaki her d_i, 8 - 10 düğümlük, 2, 3 veya 4 parçaya ayrılmıştır. Ancak daha fazla öbeğe ya da düğüm sayısına sahip ağlar da mevcuttur. Oluşturulan her parça sorgu ağıyla karşılaştırılmış ve en yüksek puanlı karşılaştırma, q ve d_i arasındaki benzerlik puanı olarak alınmıştır. Buradaki amaç, büyük ağı daha küçük parçalara bölerek hizalamayı hızlandırmaktır. Sorgu ağıyla karşılaştırılan hedef ağ ne kadar küçük olursa, hizalama da o kadar hızlı olur.

7.1.2 İki Kademeli Öbekleme

Bu yöntem ise Bölüm 7.1.1'de anlatılan tek kademeli öbeklemenin yapılmasıyla başlar. Tek kademeli öbeklemede bazı öbeklerin düğüm sayısının fazla olması sebebiyle ikinci bir öbekleme işlemi yapılır. Düğüm sayısı fazla olan ağlar tekrar öbeklenir ve böylelikle her ağdaki düğüm sayısı ortalama 7-8 olur. Bu durumda toplam ağ sayısı artar ancak ağlardaki düğüm sayısının azalmasıyla, küçük ağlarla hizalamanın maliyetinin düşük olması sebebiyle, toplam çalışma süresi azalır. Yine

aynı şekilde tüm öbekler sorgu ağıyla karşılaştırılır ve en yüksek puanlı eşleşme alınır.

Kullanılan öbekleme yöntemini daha genel bir ifadeyle özetlersek, her d_i' nin öbeklenmesiyle sahip olduğu yaklaşık 4-5 tane öbek vardır. Bunların hepsi sorgu ağıyla hizalanır ve içinden en yüksek puanlı eşleşme d_i' nin hizalanma puanı olarak alınır.

Ayrıca ağdaki düğüm sayısının azalması ve ağ sayısının artmasının oluşturacağı sonuçlar bu bölümde incelenmektedir.

7.2 En Yüksek Dereceli Düğüm Tekniği

En yüksek dereceli düğüm, bu çalışma kapsamında oluşturulan yeni bir tekniktir. Amaç belirsizlik setindeki ağların düğüm sayısını azaltmak, böylelikle sorgu ağıyla daha hızlı hizalama yapabilmektir. Buradaki temel yaklaşım ise şöyledir; her şeyin bir özü vardır ve o öz bütünü temsil eder. Eğer özünün ne olduğu bulunursa bütünle karşılaştırmaya gerek kalmayacaktır. Özle ya da özetle karşılaştırmak bütünle karşılaştırmanın cevabını verecektir.

Yukarıda bahsedilen yaklaşım çerçevesinde denenen teknik ağı, en yüksek dereceli düğümü merkez almak koşuluyla etrafına doğru genişletmektedir. Öncelikle ağda en yüksek dereceli düğüm seçilir ve ardından seçilen düğümün komşuları arasından en yüksek dereceli ikinci düğüm seçilir. Üçüncü düğüm de aynı şekilde bu iki düğümün komşuları arasından en yüksek dereceli düğümdür. Teknik bu şekilde çalışmaya devam eder ve 10, 12, 13, 15, 17 düğümlü ağlar oluşturulur. Burada kullanılan en yüksek dereceli düğüm ifadesi, bir düğümün toplam kenar sayısıdır.

Yukarıda verilen düğüm sayıları, bu çalışmada deneyler için kullanılan düğüm sayılarıdır. Farklı düğüm sayısına sahip ağlar da oluşturulup deneyler genişletilebilir. Seçilen ağdaki düğüm sayısının yüksek olması, o ağla sorgu ağını hizalamanın daha

maliyetli olmasına sebep olur. Ancak ağdaki düğüm sayısının artmasıyla yeni oluşturulan ağ, gerçek ağı daha iyi temsil eder ve böylelikle benzerliği daha yüksek olur. Zaman ve doğruluk açısından karşılaştırma yapılarak en uygun olan düğüm sayısı kullanılmalıdır.

7.3 Belirsizlik Seti Hesabı Sonuçları

Çizelge 7.1' de, anlatılan yedi farklı yöntemin sonuçları ve çalışma sürelerinin karşılaştırılması verilmiştir. Bu denemelerde beş farklı sorgu ağı, 50 farklı veritabanı ağıyla hizalanmış, süre ve puan hesabı, çıkan sonuçların ortalamaları alınarak yapılmıştır.

Çizelge 7.1 Belirsizlik Seti Hesabı Sonuçları

	Tam Hizalama	**TKÖ**	**İKÖ**	**EYDD 10**	**EYDD 12**	**EYDD 13**	**EYDD 15**	**EYDD 17**
Süre (sn)	2129,57	1966,05	1623,25	638,1	781,83	922,67	975,23	1089,51
Puan	14.579,8	11.960,8	12.175,8	11.148,2	11.954,8	12.280,6	12.750,2	13083,6

Çizelge 7.2 Belirsizlik Seti Süre ve Puan Kazancı

	TKÖ	**İKÖ**	**EYDD 10**	**EYDD 12**	**EYDD 13**	**EYDD 15**	**EYDD 17**
Tam Hizalamadan Sapma (Süre)	163,52	506,32	1491,47	1347,74	1206,9	1154,34	1040,06
Ağ Başına Tam Hizalamadan Sapma (Süre)	3,27	10,13	29,83	26,95	24,14	23,09	20,8
Tam Hizalamadan Sapma (Puan)	2619	2404	3431,6	2625	2299,2	1829,6	1496,2
Ağ Başına Tam Hizalamadan Sapma (Puan)	52,38	48,08	68,63	52,5	45,98	36,59	29,9

Çizelge 7.1 ve 7.2' de görülmektedir ki tek kademeli öbekleme yöntemi QNET' le yapılan tam hizalamaya göre süre ve puan olarak çok büyük avantaj

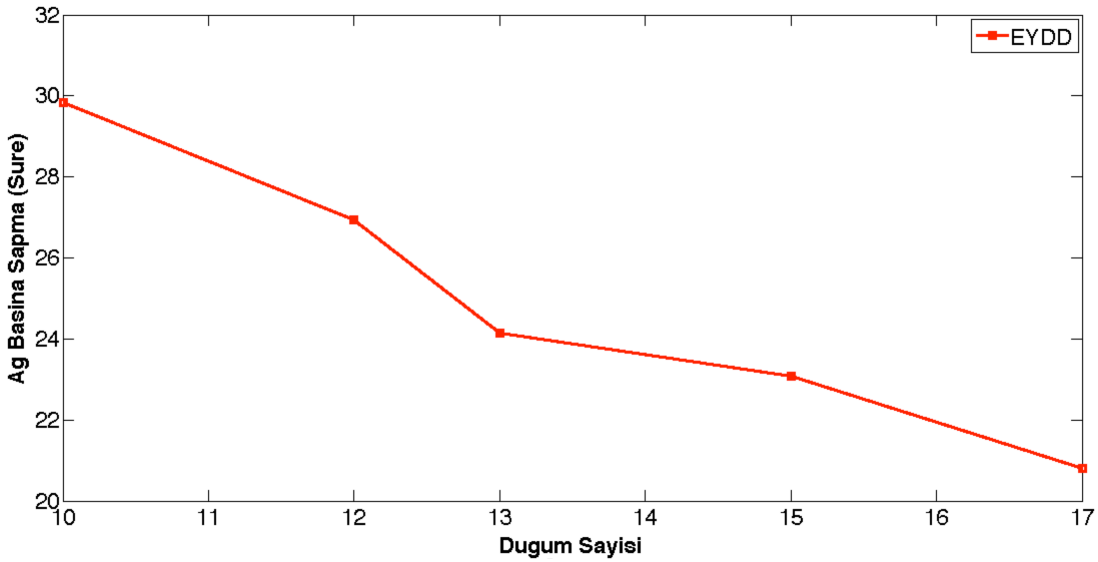
sağlamamaktadır. Süre olarak ağ başına 3,27 saniyelik bir kazanç olmasına rağmen puan olarak ağ başına 52,38' le biraz geri kalmaktadır. İki kademeli öbekleme yöntemi ise süre olarak 10,13' le daha avantajlıdır, puan olarak da ağ başına tam hizalamadan 48,08 puan sapmaktadır. Bu puan ağın parçalara ayrıldığı düşünüldüğünde tam hizalamanın çok gerisinde değildir. Tam hizalamayla ölçülen benzerlik puanı ortalama 300 olarak düşünülürse, iki kademeli öbekleme yöntemiyle bu puanın yaklaşık %85'ine ulaşılabilmektedir. Bu da ağ başına yaklaşık 10,13 saniyelik avantajla, göreceli olarak iyi bir sonuçtur.

Öbekleme için kullanılan iki yöntemin sonuçları göstermektedir ki hizalanacak ağ sayısındaki artışı, düğüm sayısındaki azalma dengelemiş, hatta süreyi daha da aşağıya çekmiştir. Daha önce bahsedildiği gibi iki kademeli öbekleme yönteminde sorgu ağıyla hizalanacak ağ sayısı, tek kademeli yönteme göre daha çoktur. Ancak bu yöntemde ağlardaki düğüm sayısı azaltılmıştır ve yaklaşık 7, 8 civarına çekilmiştir. Bu durum sorgu ağıyla yapılan hizalamaların süresini azaltmış ve ortalama hizalama süresini aşağıya çekmiştir. Sonuç olarak öbeklemedeki amaç şu şekilde özetlenebilir; hizalanacak ağ sayısını mümkün olduğunca az tutarak, oluşturulan öbeklerdeki düğüm sayısı mümkün olduğu kadar azaltılmalı ancak ağların tüm ağı en şekilde iyi temsil etmesi sağlanmalıdır.

En yüksek dereceli düğüm tekniği, sonuçlara bakıldığında öbekleme yöntemlerine göre daha başarılıdır. Puan olarak 13, 15 ve 17 düğümlü en yüksek dereceli düğüm teknikleri, öbekleme yöntemlerinin ikisini de geçmiş, süre olarak da bu yöntemlere göre büyük avantaj sağlamıştır. Hizalamada +/- 10 puanlık sapmalara izin verildiği düşünüldüğünde, süre olarak ağ başına yaklaşık 20,8 saniye avantaj sağlayan 17 düğümlü EYDD tekniğinin tam hizalamadan 29,9 puan uzak kalması çok fazla değildir. Bu da 300 puanlık ortalama bir benzerlik değeri alındığında, bu değerin yaklaşık %91' ine denk gelmektedir.

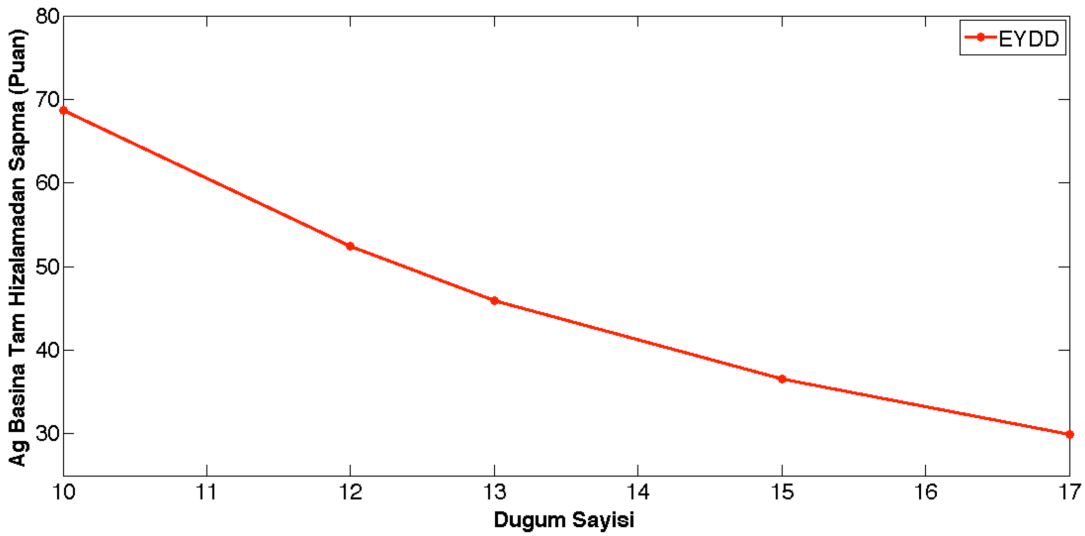
Şekil 7.1'de görüldüğü gibi EYDD tekniğinde kullanılan düğüm sayısının artmasıyla tam hizalamadan süre olarak sapma azalmaktadır, yani çalışma süresi uzamakta, kazanılan avantaj azalmaktadır. Bu istenen bir durum değildir. Tekniğin çalışma

süresi tam hizalamadan ne kadar uzak olursa o kadar avantajlıdır. Çünkü buradaki amaç, tam hizalamaya süre olarak avantaj sağlarken puan olarak da mümkün olduğu kadar yaklaşmaktır.



Şekil 7.1 EYDD Tekniği Süre Kazanımı

Şekil 7.2' de ise teknikte kullanılan düğüm sayısındaki artışın, ağ başına tam hizalamadan sapma puanına etkisi gözlemlenmektedir. Görüldüğü gibi düğüm sayısındaki artışın etkisi olumludur ve düğüm sayısı arttıkça tam hizalamaya puan olarak yaklaşılmaktadır. Bu iki grafik birlikte düşünüldüğünde düğüm sayısının artmasıyla çalışma süresi uzamakta, benzerlik puanı artmaktadır. Bu durumda süre ve puan olarak en verimli düğüm sayısı seçilerek teknik uygulanmalıdır.



Şekil 7.2 EYDD Tekniği Puan Kazanımı

Bu bölümde yapılanlar şu şekilde özetlenebilir; ESBiD veya RINQ yöntemlerinin çalışması sonucunda belirsizlik setinde biriken ağların hizalanması için uzun ve maliyetli tam hizalama yerine daha hızlı bir yaklaşım bulunmak istenmektedir. Bunun için bu ağlar öbekleme yöntemleriyle öbeklere ayrılmış veya en yüksek dereceli düğüm tekniğiyle küçültülmüştür.

Sonuç olarak en yüksek dereceli düğüm tekniği, kullanılan düğüm sayısı 13’ ten fazla olduğu durumlarda öbekleme yönteminden daha başarılıdır. Ayrıca düğüm sayısındaki artış çalışma süresini arttırırken tekniğin doğruluğunu da arttırmaktadır.

8 ESBiD YÖNTEMİ

Bu bölümde RINQ yöntemi temel alınarak tez kapsamında oluşturulan yeni bir referans tabanlı indeksleme yöntemi olan ESBiD özetlenerek anlatılmaktadır. Yöntemin RINQ' ya göre temel farklılıklarından Bölüm 4, 5, 6 ve 7 'de bahsedilmektedir. Bu bölümde ise sadece yöntem dahilinde geliştirilen yeni yaklaşımların genel özeti ve çalışma sırası verilmektedir.

Bölüm 4' te, iki yöntemde de temel olarak kullanılan QNET algoritmasının kodlanmasındaki, programlama dilindeki ve kullanımındaki farklılıktan dolayı ortaya çıkan performans gelişimi anlatılmaktadır. Bölüm 5' te referans seçimine biyolojik verilerin de dahil edilmesi, Bölüm 6' da ise ikinci üst sınır hesabının geliştirilmesiyle birlikte kullanılan sezgisel yaklaşım ve belirsizlik setindeki ağ sayısının azaltılmasından bahsedilmektedir. Son olarak Bölüm 7' de, uygulamanın çalışmasının sonunda belirsizlik setinde biriken ağların daha verimli olarak sorgu ağıyla hizalanması için kullanılan yaklaşımlardan bahsedilmektedir.

ESBiD yönteminin çalışma adımları RINQ' da olduğu gibi çevrimiçi ve çevrimdışı olmak üzere iki bölüme ayrılmalıdır.

8.1 Çevrimdışı Adımlar

i. ***Referans Seti'nin oluşturulması**:* Bu adımda referans seti, Bölüm 5' te anlatıldığı gibi, her biyolojik yolaktan bir ağ alınmak üzere oluşturulmaktadır.

ii. ***Referans Seti'yle veritabanındaki ağların, QNET' le hizalanması ve bu eşleşmelerin kaydedilmesi*:** Bu adım çok uzun sürmektedir. Bir referans ağı en az 7 düğümden oluşur. Her referans ağıyla, veritabanında bulunan 297 tane ağ tek tek QNET kullanılarak hizalanır. Bu hizalama işlemi 7 düğümlü bir referans ağı için Hadoop

kullanılmadığında ortalama 2 saat 45 dakika, 10 makinadan oluşan Hadoop altyapısı kullanıldığında ise yaklaşık 15 dakika sürmektedir. Bu süre sadece bir referans ağı için geçerlidir, toplam süre ise yaklaşık olarak, oluşturulmak istenen referans ağı sayısıyla bu sürenin çarpımıdır.

iii. ***Veritabanındaki ağların öbeklenmesi veya EYDD tekniğine göre düzenlenmesi:*** Veritabanındaki ağlar kullanılacak yönteme göre öbeklemeyle öbeklenir ya da EYDD tekniğiyle, istenilen ağ sayısına göre küçültülür.

8.2 Çevrimiçi Adımlar

i. ***Eşik değerinin belirlenmesi***: Veritabanıyla hizalanacak sorgu ağının hangi eşik değerine göre hizalanacağı belirlenir.

ii. ***Sorgu ağının, QNET kullanılarak tüm referanslarla hizalanması***: Bu hizalamalar ve daha önce yapılan referans – veritabanı hizalamaları sayesinde, Bölüm 3.3.1' de anlatıldığı gibi alt sınır hesabı kullanılarak, sorgu ağı – veritabanı ağı düğümleri arasındaki eşleşmeler bulunur. Ayrıca Bölüm 3.3.2' de anlatıldığı gibi üst sınır hesabı da yapılır. Bu hesap için "Macar Algoritması[15]" [50] kullanılmıştır. Bu algoritma sayesinde en yüksek ağırlıklı iki parçalı çizge problemi çözülür.

iii. ***Bölüm 6'da anlatıldığı gibi ikinci üst sınır hesabının yapılması:*** Bu hesapla beraber sezgisel yaklaşım kullanılarak, belirsizlik setinde yer alacak olan ağlardan bir kısmı sonuç setine aktarılır. Böylelikle belirsizlik setindeki ağ sayısı azaltılmış olur.

[15] ing: Hungarian Algorithm

iv. ***Belirsizlik setindeki ağların hizalanması****:* Veritabanındaki ağlar sonuç seti, filtre seti ve belirsizlik seti olarak üçe ayrıldıktan sonra belirsizlik setinde biriken ağlar, sorgu ağıyla QNET kullanılarak hizalanır. Bu hizalama Bölüm 7'de anlatılan yöntemlerden biriyle, istenirse Hadoop kullanılarak da yapılabilir.

Bu adımların sonunda sonuç seti ve filtre seti olarak veritabanı, en başta hedeflendiği gibi ikiye ayrılmış olur. Sonuç setindeki ağların sorgu ağıyla benzerliği en başta seçilen eşik değerinden yüksektir, yani benzerlik, istenilen benzerlik değerinin üzerindedir. Filtre setindeki ağların benzerliği ise eşik değerinden düşüktür, yani istenilen benzerlik değerinin altındadır.

Yapılan tüm denemelerde kullanılan sorgu ağı, veri tabanından rastgele bir ağın seçilmesiyle oluşturulmuştur. Seçilen ağdan rastgele bir düğüm seçilir, ardından o düğümün komşuları kullanılarak istenilen düğüm sayısına sahip bir sorgu ağı oluşturulur.

9 SONUÇ

Biyolojik ağların benzerliğinin hesaplanması için literatürde var olan birçok yöntem arasından QNET ve RINQ yöntemleri bu çalışma kapsamında kullanılmış, sonuçları, çalışma adımları analiz edilmiş ve yeni bir yöntem olan ESBiD, bu yöntemlerde görülen eksiklikler üzerine yapılan geliştirmelerle oluşturulmuştur.

Çalışmada öncelikle performans açısından iyileştirmeler yapılmış, bu iyileştirmeler yöntemde yapılan değişikliklerle pekiştirilmiştir. Öncelikle QNET yöntemi Java'da kodlanmış ve önemli bir performans artışı sağlanmıştır. İkinci performans adımı olarak yöntem, Hadoop çatısında denenmiş, 10 makinalı(18 çekirdek) bir Hadoop öbeğinde QNET kullanılarak 7 düğümlü bir sorgu ağının tüm veritabanıyla hizalanması 14 dakika 42 saniye sürmüştür. Bu sürenin, QNET C++ gerçekleştiriminde 6 saat 35 dakika 50 saniye olduğu düşünüldüğünde, sağlanan gelişme çok açık gözükmektedir. Bu sayede 11,42 hızlanma, 0,63 verimlilikle sağlanmıştır.

Performans iyileştirmesinin yanında yöntemde de bazı değişiklikler yapılmıştır. Öncelikle referans seçimi yapılırken ağların biyolojik özellikleri de dikkate alınmıştır. Biyologlarla yapılacak ortak çalışmalar neticesinde bu yöntem daha da geliştirilebilecektir.

İkinci olarak sezgisel bir yaklaşım geliştirilmiş ve belirsizlik setinde biriken ağ sayısı azaltılmıştır. Belirsizlik setinde biriken ağlar referans tabanlı indeksleme yönteminin zayıf noktalarındandır. Sebebi ise burada biriken ağların sorgu ağıyla tam hizalama yöntemi kullanılarak hizalanma gerekliliğidir. Bu hizalama ise çok maliyetlidir. Buradaki ağların sayısının azaltılmasıyla, yapılması gereken hizalama sayısı azalacak, böylelikle yöntemin çalışma süresi kısalacaktır. Sonuç olarak uygulanan sezgisel yaklaşımın kullanılmasıyla belirsizlik setinde biriken ağların ortalama % 29,85' i belirsizlik setinden sonuç setine aktarılmıştır. Aktarılan ağların doğruluk oranı ise % 93,22' dir.

Son olarak, belirsizlik setinde biriken ağların sorgu ağıyla QNET kullanılarak hizalanmasına alternatif bazı yaklaşımlar geliştirilmiştir. En yüksek dereceli düğüm tekniği bu kapsamda denenen yeni bir yaklaşımdır. Bu kapsamda 10, 12, 13, 15 ve 17 düğümlü EYDD teknikleri denenmiş, öbekleme yönteminin de kullanımıyla tam hizalama yöntemine göre performansları ölçülmüştür. Yöntemlerden 17 düğümlü EYDD tekniği, %51,14'ü kadar kısa bir sürede, puan olarak tam hizalamaya %89,74 yaklaşmıştır. Düğüm sayısı daha az olan ağlarda EYDD tekniği daha hızlı çalışmakta, puan olarak tam hizalamadan daha çok sapmaktadır. Düğüm sayısının artmasıyla da puan artmakta ancak süre uzamaktadır. Öbekleme yöntemleri de çalışmanın bu kısmında denenmiş ancak EYDD tekniği kadar başarılı olamamıştır. Ağların daha çok öbeklere ayrıldığı ve her öbekteki düğüm sayısının mümkün olduğunca azaltıldığı iki kademeli öbekleme yönteminde süre kazanımı tam hizalamaya göre ağ başına 10,13 saniyedir. Puan olarak da tam hizalamayla ölçülen benzerlik puanı ortalama 300 olarak düşünüldüğünde, bu puanın yaklaşık %85'ine ulaşılabilmektedir. Bu sonuç, EYDD tekniğinin gerisinde kalmasına rağmen göreceli olarak iyi bir sonuçtur.

Tüm denemeler dikkate alındığında tez kapsamında referans tabanlı indeksleme yöntemi üzerine çalışmalar yapılmış ve başarılı sonuçlar alınarak ESBiD yöntemi geliştirilmiştir. Bu yöntemin biyolojik ağların hizalanmasında hız ve performans olarak fayda sağlayacağı ve ileriki çalışmalara referans teşkil ederek, yön vereceği umulmaktadır.

KAYNAKLAR

[1] Oxford Dictionary of English,.: Oxford University Press, 2003.

[2] Jacques Cohen, "Bioinformatics—An Introduction for Computer Scientists," *ACM Computing Surveys (CSUR)*, vol. 36, no. 2, pp. 122-158, June 2004.

[3] Roberto Tamassia, Ed., "Biological Networks," in *Handbook of Graph Drawing and Visualization*.: CRC Press, 2013.

[4] Stefano Boccaletti, Vito Latora, and Yamir Moreno, Eds.,.: World Scientific, 2009, vol. 10, p. 9.

[5] Gunhan Gunsoy and Tamer Kahveci, "RINQ: Reference-based Indexing for Network Queries," *Bioinformatics*, vol. 27, no. 13, pp. i149-i158, 2011.

[6] Uri Alon, "Biological networks: the tinkerer as an engineer," *Science*, vol. 301, pp. 1866-1867, 2003.

[7] Stephen A. Cook, "The Complexity Of Theorem-Proving Procedures," in *Third Annual ACM Symposium on Theory of Computing*, 1971, pp. 151-158.

[8] Yuanyuan Tian, Richard C. McEachin, Carlos Santos, David J. States, and Jignesh M. Patel, "SAGA: a subgraph matching tool for biological graphs," *Bioinformatics.*, vol. 23, no. 2, pp. 232-239, 2007.

[9] Rosalba Giugna and Dennis Shasha, "GraphGrep: A Fast and Universal Method for Querying Graphs," in *16th International Pattern Recognition Conference* , vol. 2, 2002, pp. 112-115.

[10] Xifeng Yan, Philips S. Yu, and Jiawei Han, "Graph Indexing: A Frequent Structure-based Approach," in *2004 ACM SIGMOD International Conference*, 2004, pp. 335-346.

[11] Xifeng Yan, Philips S. Yu, and Jiawei Han, "Substructure Similarity Search in Graph Databases," in *In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005, pp. 766-777.

[12] Yusuf Kavurucu, "Network Motifs and Indexing Techniques in Biological Networks," *Journal of Naval Science and Engineering*, vol. 8, no. 2, pp. 87-102, 2012.

[13] Huahai He and Ambuj K. Singh, "Closure-Tree: An Index Structure for Graph Queries," in *Proceedings of the 22nd International Conference on Data Engineering*, 2006, pp. 38-38.

[14] Banu Dost et al., "QNet: A Tool for Querying Protein Interaction Networks," *Journal of Computational Biology*, vol. 15, no. 7, pp. 913-925, 2008.

[15] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, 2008.

[16] Minoru Kanehisa and Susumu Goto, "KEGG: Kyoto Encyclopedia of Genes and Genomes," *Nucleic Acids Research*, vol. 28, no. 1, pp. 27-30, 2000.

[17] Saul B. Needleman and Christian D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology* , vol. 48, no. 3, pp. 443-53, 1970.

[18] Temple F. Smith and Michael S. Waterman, "Identification of Common Molecular Subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195-197, 1981.

[19] Margaret O. Dayhoff and Robert M. Schwartz, "A model of Evolutionary Change in Proteins," in *Atlas of protein sequence and structure*., 1978, pp. 345-352.

[20] Steven Henikoff and Jorja G. Henikoff, "Amino acid substitution matrices from protein blocks," *Proceedings of the National Academy of Sciences*, vol. 89, no. 2, pp. 10915-10919, 1992.

[21] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman, "Basic Local Alignment Search Tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403-410, 1990.

[22] (2013, July) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/Hadoop

[23] Tom White, *Hadoop: The Definitive Guide*, 2nd ed.: O'Reilly.

[24] Hakan İlter. (2012, Temmuz) DevVeri.com. [Online]. http://devveri.com/hadoop/hadoop-mapreduce-ornek-uygulama

[25] Matteo Matteucci. (2013, June) A Tutorial on Clustering Algorithms. [Online]. http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/

[26] Kayhan Dursun, "Çok Amaçlı Genetik Algoritma İle Kategorik verilerin Sınıflandırılması," Bilgisayar Mühendisliği, Tobb Ekonomi ve Teknoloji Üniversitesi Fen Bilimleri Enstitüsü, Ankara, Thesis 2012.

[27] Jerzy Stefanowski, Data Mining - Clustering, 2008.

[28] Martin Ester, Hans Peter Kriegel, Jörg Sander, and Xiaowei Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *2nd International Conference on Knowledge Discovery and Data Mining*, vol. 96, 1996, pp. 226-231.

[29] Michael Ankerst, Markus M. Breunig, Hans Peter Kriegel, and Jörg Sander, "OPTICS: Ordering Points To Identify the Clustering Structure," in *ACM SIGMOD international conference on Management of data*, 1999, pp. 49-60.

[30] Wei Wang, Jiong Yang, and Richard Muntz, "STING : A Statistical Information Grid Approach to Spatial Data Mining," in *VLDB*, vol. 97, Feburary 1997, pp. 186-195.

[31] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang, "Wavecluster: A multi-resolution clustering approach for very large spatial databases," in *VLDB*, vol. 98, 1998, pp. 428-439.

[32] Jian Pei, Data Mining -- Grid-Based Clustering.

[33] Teuvo Kohonen, *Self Organising Maps*.: Springer, 2001, vol. 30.

[34] Erol Özgür, "Memeli kültür hücrelerinde hücre içi kalsiyum sinyalinde görev alan elemanların entegre bir sistem olarak incelenmesi," Sağlık Bilimleri Enstitüsü, Ankara Üniversitesi, Ankara, Yüksek Lisans Tezi 2008.

[35] (2009, July) Cell Signalling Technology. [Online]. http://www.cellsignal.com/reference/pathway/Toll_Like.html

[36] (2013, March) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/MAPK/ERK_pathway

[37] Christopher S. Beevers, Fengjun Li, Lei Lui, and Huang Shile, "Curcumin

inhibits the mammalian target of rapamycin-mediated signaling pathways in cancer cells," *International Journal of Cancer*, vol. 119, no. 4, pp. 757-64, August 2006.

[38] Tocris Bioscience. [Online]. http://www.tocris.com/pathways/vegfPathway.php#.UchFOhai3ww

[39] (2013, January) WikiBooks. [Online]. http://en.wikibooks.org/wiki/Structural_Biochemistry/Cell_Signaling_Pathways/Insulin_Signaling

[40] (2013, May) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/B-cell_receptor

[41] (2013, June) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/NOD-like_receptor

[42] (2013, Feburary) Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/JAK-STAT_signaling_pathway

[43] Encyclopedia of Molecular Pharmacology, *The other two families of PRRs, the NOD-like receptors (NLRs) and the RIG-like helicases (RLHs) are soluble receptors present in the cytosol and act as sensors to detect a variety of viral and bacterial products*, 2nd ed., Stefan Offermanns and Walter Rosenthal, Eds.: Springer, September 2012.

[44] Penn State University. PPAR Resource Page.

[45] (2013, June) Nedir Ne Demek? [Online]. http://www.nedirnedemek.com/phosphatidylinositol-nedir-phosphatidylinositol-ne-demek

[46] Erez M. Bubil and Yosef Yarden, "The EGF receptor family: spearheading a merger of signaling and therapeutics," *Current Opinion in Cell Biology*, vol. 19, no. 2, pp. 124-134, April 2007.

[47] Türk Dil Kurumu, *Veteriner Hekimliği Terimleri Sözlüğü*., 2009.

[48] Joshua O'Madadhain, Daniel Fisher, Padhraic Smyth, Scott White, and Yan-Biao Boey, "Analysis and Visualization of Network Data using JUNG," *Journal of Statistical Software*, vol. 10, no. 2, pp. 1-35, 2005.

[49] Wu Fang and Bernardo A. Huberman, "Finding communities in linear time: a physics approach," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 38, no. 2, pp. 331-338, 2004.

[50] Harold W. Kuhn, "The Hungarian Method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83-97, 1955.

[51] Ron Y. Pinter, Oleg Roghlenko, Esti Yeger Lotem, and Michal Ziv Ukelson, "Alignment of metabolic pathways," *Bioinformatics*, vol. 21, no. 16, pp. 3401-3408, 2005.

[52] Tomer Shlomi, Daniel Segal, Eytan Ruppin, and Roded Sharan, "QPath: a method for querying pathways in a protein-protein interaction network," *BCM Bioinformatics*, vol. 7, 2006.

[53] Brian P. Kelly et al., "PathBLAST: a tool for alignment of protein interaction networks," *Nucleic acids research*, vol. 32, no. suppl 2, pp. W83-W88, 2004.

[54] Ferhat Ay, Tamer Kahveci, and Valerie de Crecy Lagard, "A fast and accurate algorithm for comparative analysis of metabolic pathways," *J. Bioinformatics and Computational Biology*, vol. 7, no. 3, pp. 389-428, 2009.

[55] Ferro Alfredo et al., "NetMatch: a Cytoscape plugin for searching biological networks," *Bioinformatics*, vol. 23, no. 7, pp. 910-912.

[56] Eric Banks, Elena Nabieva, Ryan Peterson, and Mona Singh, "NetGrep: fast network schema searches in interactomes," *Genome Biology*, vol. 9, no. 9, 2008.

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, Adı	: SÖYLEV, Arda
Uyruğu	: T.C.
Doğum Tarihi ve Yeri	: 31.07.1984 İstanbul
Medeni Hali	: Bekar
Telefon	: 0 (552) 302 11 77
E-Posta	: asoylev@gmail.com

Eğitim

Derece	**Eğitim Birimi**	**Mezuniyet Tarihi**
Lisans	FMV Işık Üniversitesi Bilgisayar Mühendisliği	2007

İş Deneyimi

Yıl	**Yer**	**Görev**
2012-2013	KTO Karatay Üniversitesi	Araştırma Görevlisi
2007-2008	Nortel Networks Netaş	ARGE Departmanı Telekom Mühendisi

Yabancı Dil

İngilizce

Yayınlar

1. Soylev, Arda; Bicakci, Kemal; Tavli, Bulent, "Uncovering the Impact of Minimum-Energy Routing on Lifetime of Wireless Sensor Networks," *Distributed Computing in Sensor Systems (DCOSS), 2013 IEEE International Conference on*, vol., no., pp.436, 441, 20-23 May 2013