ÖZEL ÖRGÜ ARA BAĞLANTI AĞLARINDA TEMEL ÇİZGE ALGORİTMALARININ TASARIMI VE ANALİZİ

2024
DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ

Ayşe Nur ALTINTAŞ TANKÜL

Tez Danışmanları
Doç. Dr. Burhan SELÇUK
Doç. Dr. Muhammed Kamil TURAN

ÖZEL ÖRGÜ ARA BAĞLANTI AĞLARINDA TEMEL ÇİZGE ALGORİTMALARININ TASARIMI VE ANALİZİ

Ayşe Nur ALTINTAŞ TANKÜL

Tez Danışmanları
Doç. Dr. Burhan SELÇUK
Doç. Dr. Muhammed Kamil TURAN

T.C.
Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalında
Doktora Tezi
Olarak Hazırlanmıştır

KARABÜK
Ocak 2024

Ayşe Nur ALTINTAŞ TANKÜL tarafından hazırlanan "ÖZEL ÖRGÜ ARA BAĞLANTI AĞLARINDA TEMEL ÇİZGE ALGORİTMALARININ TASARIMI VE ANALİZİ" başlıklı bu tezin Doktora Tezi olarak uygun olduğunu onaylarım.

Doç. Dr. Burhan SELÇUK .....
Tez Danışmanı, Bilgisayar Mühendisliği Anabilim Dalı
Doç. Dr. Muhammed Kamil TURAN .....
Tez İkinci Danışmanı, Biyoinformatik Mühendisliği Anabilim Dalı

Bu çalışma, jürimiz tarafından Oy Birliği ile Bilgisayar Mühendisliği Anabilim Dalında Doktora tezi olarak kabul edilmiştir. 25/01/2024

Ünvanı, Adı SOYADI (Kurumu)	İmzası
Başkan : Prof. Dr.Ali KARCI (İÜ)	Online
Üye : Doç. Dr. Okan ERKAYMAZ (MSÜ)	Online
Üye : Doç. Dr.Burhan SELÇUK (KBÜ)	.....
Üye : Doç. Dr.Muhammed Kamil TURAN (KBÜ)	.....
Üye : Doç. Dr.Hakan KUTUCU (KBÜ)	.....
Üye : Dr. Öğr. Üyesi Ferhat ATASOY (KBÜ)	.....
Üye : Dr. Öğr. Üyesi Oğuzhan MENEMENCİOĞLU (KBÜ)	.....

KBÜ Lisansüstü Eğitim Enstitüsü Yönetim Kurulu, bu tez ile, Doktora derecesini onamıştır.

Doç. Dr. Zeynep ÖZCAN .....
Lisansüstü Eğitim Enstitüsü Müdürü

"Bu tezdeki tüm bilgilerin akademik kurallara ve etik ilkelere uygun olarak elde edildiğini ve sunulduğunu; ayrıca bu kuralların ve ilkelerin gerektirdiği şekilde, bu çalışmadan kaynaklanmayan bütün atıfları yaptığımı beyan ederim."

Ayşe Nur ALTINTAŞ TANKÜL

ÖZET

Doktora Tezi

ÖZEL ÖRGÜ ARA BAĞLANTI AĞLARINDA TEMEL ÇİZGE ALGORİTMALARININ TASARIMI VE ANALİZİ

Ayşe Nur ALTINTAŞ TANKÜL

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanları:
Doç. Dr. Burhan SELÇUK
Doç. Dr. Muhammed Kamil TURAN
Ocak 2024, 113 sayfa

Bu tezde, 2 boyutlu örgü çizgelerinin alt çizgeleri olan Bağlantılı Kare Ağ Çizgeleri (CSNG) (2022, 2023) ve Fraktal Kübik Ağ Çizgeleri (FCNG) (2015) üzerinde temel çizge problemleri çalışılmıştır ve bu problemler için çalışma zamanı ve çizge maliyeti açısından daha iyi sonuçlar elde edilmesi hedeflenmektedir. CSNG ve FCNG çizgeleri, iletişim ağı veya ara bağlantı ağları olarak adlandırılan ağların bir çeşididir. Ara bağlantı ağları düğümler arasında verimli bir şekilde veri aktarımı yapmak üzere tasarlanmışlardır ve büyük ölçekli bilgi işlem sistemlerinin mimarisinde kritik bir bileşen olarak rol oynamaktadırlar. Bu çizgeler özyinelemeli olarak tanımlanmış, ara bağlantı ağlarında sıklıkla tercih edilen hiperküpün türevleridir. Çizgelerin tanımlanmasında her bir düğümün etiketleri arasında bir bitlik değişimi esas alan gri kod kullanılmıştır.

CSNG üzerine yapılan çalışmanın odak noktası hiperküp yardımıyla CSNG için temel çizge problemlerine çözüm bulan algoritmalar geliştirmektir. CSNG çizgesinde kapalı Hamilton yolunu bulan verimli bir algoritma önerilmiştir. Bunula birlikte çizgedeki düğümleri 2 boyutlu düzlemde haritalanmasını ve tek noktaya yönlendirmesini gerçekleştiren algoritmalar önerilmiştir. Fakat bu algoritmalar 2 boyutlu örgü çizgelerindeki tek noktaya yönlendirme algoritması ile aynı çalışma zamanına sahip olduğundan, çalışma zamanını iyileştirmek için etiketleme ve tek noktaya yönlendirme algoritmalarında paralelleştirme yapılarak işlem birimi sayısına göre daha iyi sonuçlar elde edilmiştir. Bunun yanı sıra, CSNG için hiperküp yayın algoritmalarının bir benzerleri önerilmiştir.

Fraktal yapılar birçok temel alanda kullanılan birbirini tekrar eden geometrik yapılardır. FCNG aynı CSNG gibi bir hiperküp varyant olarak tanımlanmış bir fraktal çizgedir. FCNG için daha önce ortaya konulmamış olan yeni topolojik özellikler elde edilmiştir. FCNG üzerinde yönlendirme ve en kısa yol problemleri için yeni bir strateji tanıtılmış ve bu stratejiyi kullanan özyinelemeli algoritmalar önerilmiştir. Ağ düğümlerini 2 boyutlu düzlemde haritalamak için bir algoritma ve en kısa yolu oluşturmak için kullanılan algoritmalar önerilmiş ve çalışma süreleri hesaplanmıştır. Yönlendirme ve en kısa yol problemleri için çalışma süreleri 2 boyutlu örgü ağlarda verilen algoritmalar ile benzer sonuçlar vermiş olsa da bu problemler için FCNG çizgesinin kullanımı ağ oluşum maliyetini düşürmüştür.

Anahtar Sözcükler : Ara bağlantı ağları, Çizgesel göstergeler, Hamilton çizgeleri, Yönlendirme, Yayınlama, Etiketleme..

Bilim Kodu **:** 92402

ABSTRACT

Ph. D. Thesis

ROUTING AND PATH ALGORITHMS IN INTERCONNECTION NETWORK GRAPHS AND ANALYSIS OF ALGORITHMS

Ayşe Nur ALTINTAŞ TANKÜL

Karabük University
Institute of Graduate Programs
Department of Computer Engineering

Thesis Advisors:
Assoc. Prof. Dr. Burhan SELÇUK
Assoc. Prof. Dr. Muhammed Kamil TURAN
January 2024, 113 pages

In this thesis, basic graph problems on the subgraphs of 2D mesh graphs, Connected Square Network Graphs (CSNG) (2022, 2023) and Fractal Cubic Network Graphs (FCNG) (2015), are studied with the aim of obtaining better results in terms of runtime and graph cost for these problems. CSNG and FCNG graphs are a variant of interconnection networks. Interconnection networks are designed to efficiently transfer data between nodes and are a critical component in the architecture of large-scale computing systems. These graphs are recursively defined derivatives of the hypercube, which is often preferred in interconnection networks. The graphs are defined using gray code based on a one-bit exchange between the labels of each node.

The focus of the work on CSNG is to develop algorithms that solve basic graph problems for CSNG using hypercubes. An efficient algorithm that finds the closed Hamiltonian path in a CSNG graph is proposed. In addition to this, algorithms have been proposed for mapping the nodes in the graph in the 2D plane and unicast routing. Since these algorithms have the same runtime as the unicasting algorithm in 2D mesh graphs, in order to improve the runtime, the labeling and unicasting algorithms are parallelized to achieve better results with respect to the number of processing units. In addition, similar hypercube broadcast algorithms are proposed for CSNG.

Fractal structures are repeating geometric structures used in many fundamental fields. FCNG is a fractal graph defined as a hypercube variant, just like CSNG. New topological properties have been obtained for FCNG that have not been introduced before. A new strategy for routing and shortest path problems on FCNG is introduced and recursive algorithms using this strategy are proposed. An algorithm for mapping network nodes in 2D plane and algorithms for shortest path generation are proposed and their running times are calculated. Although the running times for the routing and shortest path problems are similar to the algorithms for 2D mesh networks, the use of the FCNG graph for these problems reduces the network construction cost.

Key Words **:** Interconnection networks, Graphical indices, Hamiltonian graphs, Routing, Broadcasting, Labeling.

Science Code **:** 92402

TEŞEKKÜR

Değerli Danışmanlarım Doç. Dr. Burhan Selçuk ve Doç. Dr. Kamil Turan, Doktora tez çalışmamın planlanmasında, araştırılmasında, yürütülmesinde ve oluşumunda gösterdiğiniz özverili rehberlik ve destek için içtenlikle teşekkür ederim. Bilgi birikiminiz, benim akademik yolculuğumda ışık tutan bir rehber oldu. Her sorumu sabırla yanıtladığınız ve deneyimlerinizi paylaştığınız için minnettarım. Eşim Mehmet Tankül ve kızım Bilge Çağın Tankül'e, bu süreçteki sevgi, anlayış ve destekleri için teşekkür etmek isterim. Sizlerin varlığı, bu zorlu süreçte beni motive eden en büyük güç oldu. Ayrıca, sevgili annem Elif Altıntaş ve babam Adnan Altıntaş'a, hayatım boyunca sağladıkları destek ve sevgi için ve doktoranın son süreçlerinde yanımda oldukları için minnettarlığımı ifade etmek isterim. Ailenizin sıcaklığı ve desteği, bu yolculukta beni güçlendiren bir kaynak oldu. Bu süreçte emeği geçen herkese tüm kalbimle teşekkür ederim.

İÇİNDEKİLER

Sayfa

Sayfa

Sayfa

ŞEKİLLER DİZİNİ

Sayfa

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

ALGORİTMALAR DİZİNİ

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

SİMGELER

G : Çizge

V : Düğüm kümesi

E : Kenar kümesi

E’ : İç/dış bağlantı kenar kümesi

l : Bağlantı sayısı

p : Düğüm sayısı

deg : Düğüm derecesi

b : İkiye bölme bant genişliği

r : Çizgenin çapı

max : En büyük

min : En küçük

ec : Dış merkezlik

|| : İki dizeyi birleştirme

$\oplus$: Bit bazlı XOR işlemi

P : Yol

H : Hiperküp

S : Kaynak

D : Hedef

SCJ : Kaynak fraktal bağlantı düğümü

DCJ : Hedef fraktal bağlantı düğümü

MSG : İleti

V_S : Kaynak düğümün etiketi

V_D : Hedef düğümün etiketi

SNJ : Kaynağın bulunduğu FCNG1(k-1) boyutlu fraktalın hedefin olduğu fraktala olan bağlantı düğümlerinden en yakını

DNJ : Hedefin bulunduğu FCNG1(k-1) boyutlu fraktalın kaynağın olduğu fraktala olan bağlantı düğümlerinden en yakını

k : FCNG çizgesinin derecesi

M : Harita, etiket dizisi

KISALTMALAR

CSNG : Connected Square Netwok Graph (Bağlantılı Kare Ağ Çizgesi)

G/Ç : Giriş/Çıkış

OCN : On Chip Networks (Çip Üzerinde Ağlar)

SAN : System/Storage Area Networks (Sistem/Depolama Alanı Ağları)

LAN : Local Area Networks (Yerel Alan Ağları)

WAN : Wide Area Networks (Geniş Alan Ağları)

ATM : Automated Teller Machine (Otomatik Vezne Makinesi)

MPC : Massive Parallel Computers (Devasa Paralel Bilgisayarlar)

PC : Personel Computer (Kişisel Bilgisayar)

VoD : Video on Demand (İsteğe Bağlı Video)

HEFC : Hierarchical Extended Fibonacci Cubes (Hiyerarşik Genişletilmiş Fibonacci Küpleri)

FCNG : Fractal Cubic Netwok Graph (Fraktal Kübik Ağ Çizgesi)

HCN : Hierarchical Cubic Netwok (Hiyerarşik Kübik Ağı)

CBS : Coğrafi Bilgi Sistemi

SPP : Sortest Path Problem (En Kısa Yol Problemi)

TSP : Traveller Salesman Problem (Gezgin Satıcı Problemi)

VRP : Vehicle Routing Problem (Araç Rotalama Problemi)

BRP : Bus Routing Problem (Otobüs Rotalama Problemi)

NoC : Network-on-Chip (Çip Üzerinde Ağ)

MSB : Most Significant Bit (En Önemli Bit)

GPU : Graphics Processing Unit (Girafik İşlemci Birimi)

CPU : Central Processing Unit (Merkezi İşlem Birimi)

PAAD : Partially Adaptive and Deterministic (Kısmen Uyarlanabilir ve Deterministik)

WF : West First (Önce Batı)

DYAD : Dynamic Adaptive and Determistic (Dinamik Uyarlanabilir ve Deterministik)

TriBA : Triplet Based (Triplet tabanlı)

SDNoC : Software Defined NoC (Yazılım Tanımlı NoC)

OBL : Output Buffer Length (Çıktı Arabelleği Uzunluğu)

DBSS : Destination-Based Selection Strategy (Hedef Tabanlı Seçim Stratejisi)

FAR : Fully Adaptive Routing (Tamamen Uyarlanabilir Yönlendirme)

BSP : Bulk Synchronous Processing (Toplu Eşzamanlı İşleme)

BÖLÜM 1

GİRİŞ

Dijital sistemler modern toplumun bir parçası haline gelmiştir. Dijital bilgisayarlar, fiziksel sistemleri simüle etmekten büyük veri tabanlarını yönetmeye ve belge hazırlamaya kadar çeşitli görevler için kullanılmaktadır. Dijital iletişim sistemleri telefon görüşmelerini, video sinyallerini ve internet verilerini iletmektedirler. Ses ve video eğlencesi giderek artan bir şekilde dijital biçimde sunulmakta ve işlenmektedir. Bunun yanında, otomobillerden ev aletlerine kadar neredeyse tüm elektronik ürünler dijital olarak kontrol edilmektedir. Dijital bir sistem üç temel yapı taşından oluşmaktadır: mantık, bellek ve iletişim. Mantık, aritmetik işlemler gibi mantıksal işlemleri gerçekleştirerek veya kararlar alarak verileri dönüştürmekte ve birleştirmektedir. Bellek, verileri daha sonra almak için depolamakta ve zamanı geldiğinde iletmektedir. İletişim ise, yapılan işlemler sırasında sistem elemanları arasında verileri bir konumdan diğerine taşma işlemidir. Yapılacak tezin konusu dijital sistemlerin iletişim bileşenini ele almaktadır ve bir dijital sistemin alt sistemleri arasında veri taşımak için kullanılan iletişim ağları üzerine çalışılmıştır. Günümüzde dijital sistemlerin büyük çoğunluğunun çalışma başarısı, mantık ya da belleklerinin iyi olmasından ziyade iletişim ya da ara bağlantılarının yapısı ve performansı tarafından kısıtlanmaktadır. Yüksek seviyeli bir sistemin saat döngüsünün büyük bir kısmı kapı gecikmesine değil kablo gecikmesine ayrılmıştır ve gücün çoğu kablolar üzerinden gerçekleşen iletim için gerekmektedir. Daha küçük, daha hızlı ve daha uygun fiyatlı depolama ve işlemcilerin üretilmesine olanak sağlayan teknolojideki ilerlemelere rağmen ışık hızı sabit kalmaktadır. Bileşenler arasındaki veri alışverişi frekansı ve hızı, çağdaş CPU'ların saat hızlarının oldukça gerisinde kalmakta ve sistem bileşenleri arasındaki ara bağlantıları kontrol eden pim ve kablo yoğunlukları, bileşenlerin kendisinden daha yavaş ölçeklenmektedir. Bu nedenlerden dolayı, ara bağlantı ağları gelecekte dijital sistemlerin geliştirilmesi için

çok önemli olacaktır. Ara bağlantı ağları, tasarımcılar gereksinimi karşılayamayacak kadar az olan bağlantı bant genişliğini kullanmanın daha etkili yollarını aradıkça, günümüzün dijital sistemleri için iletişim sorunlarına bir çözüm olarak ortaya çıkmaktadır [1].

Bu tezde bilgisayar ağlarında sıklıkla kullanılan ara bağlantı ağlarından olan 2 boyutlu örgü çizgeleri ve hiperküp çizgeleri için alternatif iki farklı çizge ele alınmıştır. Bu çizgeler; Bağlantılı Kare Ağ Çizgeleri (CSNG) (2022, 2023) ve Fraktal Kübik Ağ Çizgeleri'dir (FCNG) (2015). Ele alınan bu çizgeler 2 boyutlu örgü çizgelerinin bir alt çizgesi olmakla beraber aynı zamanda hiperküp özellikleri taşımaktadır. 2 boyutlu örgü çizgelerinde temel çizge problemleri için geliştirilen klasik algoritmalardan farklı olarak hiperküp özelliklerinden faydalanılarak algoritmaların performansı arttırılmaya çalışılmış ve çizge maliyetinin düşürülmesi hedeflenmiştir.

Bu tezde öncelikle iletişim ağlarından (interconnection networks), bu ağlar ile ilgili terminolojilerden, ağların özellikleri ve ağ topolojisinden bahsedilip, yaygın olarak bilinen ve kullanılan ağ topolojilerinden halka (ring) ağları, klik (clique) ağları, örgü (mesh) ağları, torus ağları ve hiperküp ağları incelenmiş ve ağlar üzerinde kaynaktan hedefe mesaj iletimi olan yönlendirme (routing), ağda tüm düğümler üzerinden bir sefer geçen Hamilton yolu, mesajın bütün ağa iletimi olan yayma (broadcast), dinamik programlama ve böl ve fethet yöntemleri açıklanmıştır. Daha sonra iletişim ağ çizgelerinden olan Bağlantılı Kare Ağ Çizgeleri (Connected Square Netwok Graph CSNG) [2] ile ilgili bilgiler verilmiştir. Bu ağ ile ilgili terminoloji, ağın oluşturulması ve analitik özellikleri incelenmiştir ve ağ üzerinde kaynaktan hedefe mesaj iletimi olan yönlendirme algoritması, Hamilton yolu ve yayma algoritmaları önerilmiştir ve bu algoritmaların çalışma zamanları hesaplanmıştır. CSNG üzerinde kullanılan haritalama ve tek yöne yayın algoritmalarının paralelleştirilmesi yapılarak daha iyi bir çalışma süresi elde edilmesi amaçlanmıştır. Son olarak bir hiperküp türü olan Fraktal Kübik Ağ Çizgelerinin (Fractal Cubic Network Graph FCNG) [3] oluşturulması, ağın topolojik özellikleri hakkında bilgi verilmiş, ağ üzerinde yönlendirme algoritmaları, FCNG2(k) üzerinde en kısa yol algoritması, haritalama algoritması ve en kısa yol algoritmasını hesaplarken kullanılan bir alt algoritma

sunulmuştur. Her iki çizge için önerilen algoritmaların çalışma zamanları hesaplanmış ve algoritmaların çizgeler üzerinde nasıl çalıştığı örnekler ile açıklanmıştır.

BÖLÜM 2

İLETİŞİM AĞ ÇİZGELERİ

2.1. GİRİŞ

İletişim ağları, çeşitli uygulama alanlarının (yüksek performanslı bilgi işlem, depolama G/ç, küme/çalışma grubu/kurum sistemleri, ağlar arası iletişim vb.) operasyonel taleplerini karşılamak için bilgisayar sistemleri içinde ve arasında farklı düzeylerde kullanılmak üzere tasarlanmıştır. Birbirine bağlanacak olan işlem birimlerinin sayısına ve yakınlıklarına göre, iletişim ağları dört temel alanda gruplandırılabilir [4]:

2.1.1. Çip Üzerinde Ağlar (On-Chip Networks OCN'ler)

Çip üzerinde ağ (network-on-chip NoC) olarak da adlandırılan bu ağ türü, mikro mimari işlevsel birimlerini, kayıt dosyalarını, önbellekleri, bilgi işlem birimleri, işlemci ve çipler veya çoklu çip modülleri içindeki çekirdekleri birbirine bağlamak için kullanılmaktadır. Şu anda OCN'ler, maksimum ara bağlantı mesafesi milimetre düzeyinde olan bu türden birkaç bin adede kadar aygıtın bağlantısını desteklemektedir. Yüksek performanslı yongalarda kullanılan OCN'lerin çoğu, artan teknoloji ölçeklendirmesi ve geçiş entegrasyonunun neden olduğu yonga geçişli kablo gecikme sorunlarını azaltmak için özel olarak tasarlanmıştır. IBM'in CoreConnect'i, ARM'nin AMBA'sı ve Sonic'in Smart Interconnect'i gibi bazı tescilli tasarımlar daha geniş kullanım kazanmaktadır. OCN örneği olarak Cell Broadband Engine işlemci çipinde kullanılan Element Interconnect Bus verilebilir. Bu ağ, çip üzerindeki 12 öğe için ~2400 Gb/sn'de (3,2 GHz işlemci saati için) en yüksek değerine ulaşmaktadır [5].

2.1.2. Sistem/Depolama Alanı Ağları (System/Storage Area Networks SAN'lar)

Sistem/depolama alanı ağları, çok işlemcili ve çok bilgisayarlı sistemlerde işlem birimleri arasındaki işlemci-bellek ara bağlantılarının yanı sıra sunucu ve veri merkezi ortamlarındaki depolama ve G/Ç bileşenlerini birbirine bağlamak için kullanılır. IBM Blue Gene/L süper bilgisayarı gibi bazı süper bilgisayar SAN'ları binlerce cihazın birbirine bağlanmasını destekler, ancak bu kapasitenin tamamı genellikle kullanılmamakta ve yapılan uygulamalarda yüzler düzeyinde cihaz birbirine bağlanmaktadır. Maksimum ara bağlantı mesafesi çok büyük olmayan bir alanı kaplamaktadır ve genellikle birkaç on metre civarındadır. Az sayıda bazı SAN'ların ise birkaç yüz metreyi kapsayan mesafeleri bulunmaktadır. Örneğin, 2000 yılının sonlarında tanıtılan popüler bir SAN standardı olan InfiniBand, 300 m'lik bir mesafede 120 Gbps'ye kadar sistem ve depolama G/Ç ara bağlantılarını desteklemektedir.

2.1.3. Yerel Alan Ağları (Local Area Networks LAN'lar)

Bir kümedeki PC'leri birbirine bağlayarak oluşturulan ağlar LAN olarak adlandırılmaktadır. Bu tür ağlar, bir makine dairesi boyunca, bir bina veya kampüs ortamı boyunca dağıtılan bilgisayar sistemlerini birbirine bağlamak için kullanılmaktadır. Başlangıçta, LAN'lar yalnızca yüz cihaza kadar bağlı olan ağlar olarak tanımlanırken, köprüleme ile LAN'larda artık binden fazla cihaz birbirine bağlanabilmektedir. Maksimum ara bağlantı mesafesi genellikle 10 kilometreye kadar olan bir alanı kapsar, ancak bazılarının on kilometreden daha fazla mesafe aralıkları da vardır. En popüler ve dayanıklı LAN olan Ethernet, 40 km mesafe boyunca maksimum performansı destekleyen 10 Gb/sn standart bir sürüme sahiptir.

2.1.4. Geniş Alan Ağları (Wide Area Networks WAN'lar)

Uzun mesafeli ağlar olarak da adlandırılan WAN'lar, dünya genelinde dağıtılan ve ağ desteği gerektiren bilgisayar sistemlerini birbirine bağlamaktadır. WAN'lar, milyonlarca bilgisayarın binlerce kilometrelik mesafe ölçekleri üzerinden birbiriyle

olan iletişimlerini sağlamaktadır. Banka işlemlerinde kullanılan ATM'ler, dünyanın her yerinde görülebilecek bir WAN örneğidir.

Devasa paralel bilgisayarlar (massive parallel computers MPC'ler), yeni malzemelerin ve enerji kaynaklarının geliştirilmesi, yeni ilaçların geliştirilmesi, sağlık hizmetlerinin iyileştirilmesi, afet önleme ve azaltma stratejileri, hava tahmini ve maddenin ve evrenin kökenleri gibi çok çeşitli bilimsel araştırma kullanılmaktadır. MPC'ler uygulamalardaki büyük ölçekli sorunları çözmek için on binlerden milyonlarcaya kadar olabilecek hesaplama düğümü bir araya getirilerek oluşturulmaktadır. İletişim ağları, MPC sistemlerinin performansında kilit bir unsurdur. MPC'ler cazip bir bilgisayar mimarisi olarak bilinmektedir.

Bazı ağ çözümleri ticari standartlar haline gelmiş, bazıları ise tescilli kalmıştır. Tercih edilen çözümler, tasarım gereksinimlerine bağlı olarak bir iletişim ağı etki alanından diğerine önemli ölçüde farklılık gösterse de ağ sorunlarını ele almak için kullanılan sorunlar ve kavramlar, etki alanlarında dikkate değer ölçüde benzer olmaktadır. Hedef etki alanı ne olursa olsun, ağlar sistem performansı ve maliyet verimliliği açısından darboğaz oluşturmayacak şekilde tasarlanmalıdır. Bu nedenle, bilgisayar mimarlarının hedefi, mümkün olan en kısa sürede maksimum miktarda mevcut bilgiyi aktarabilen, mümkün olan en düşük maliyetli iletişim ağlarını tasarlamaktır.

2.2. TERMİNOLOJİ

Çizge: Nesneler arasındaki karmaşık, doğrusal olmayan ilişkileri temsil etmek için kullanılan soyut bir veri türüdür.

Topoloji: Bir ağda düğümler arasındaki fiziksel veya mantıksal bağlantıları tanımlayan yapıdır ve yönlendirmeyi, güvenilirliği, verimi, gecikmeyi, oluşturma kolaylığını etkilemektedir.

Yönlendirme: Bir mesajın kaynaktan hedefe gitmesi için en uygun yolu bulma sürecidir.

Bağlantılar: Ağ üzerinde sinyal taşıyan iletim ortamı ile bağlantı gerçekleşir.

Düğüm: Bir ağ içinde veri gönderebilen, alabilen veya iletebilen herhangi bir işlem birimini ifade eder.

Kenar: Düğümler arasındaki mantıksal bağlantıdır.

Mesaj: Ağ içindeki düğümler veya bileşenler (örn. çekirdekler, hafıza birimleri) arasında iletilen veri veya bilgi birimini ifade eder.

2.3. AĞ ÖZELLİKLERİ

Statik bir ağ, köşelerin düğümler olduğu ve kenarların ağ bağlantıları olduğu yönsüz bir çizge olarak temsil edilebilir. Genel olarak mesajlar bir ağ bağlantısı üzerinden aynı anda her iki yönde gönderilebildiğinden bu ağ yönsüz bir çizgedir. Çizgenin iki düğümü arasındaki mesafesi, iki düğüm arasındaki en kısa yolun uzunluğudur. Uzunluk, kaynaktan hedefe giden yoldaki kenarların sayısıdır. İletişim ağı, sistemdeki herhangi iki düğüm arasında bir yol olmasını sağlar [6].

Çizgenin bazı özelliklerinin çalışılması, ağın özelliklerini analiz etmek için önemlidir.
Bu özellikler arasında en çok aşağıdakiler kullanılmaktadır:

$G = (V, E)$ çizgesinde V düğümleri, E ise kenarları temsil etmektedir. Çizge üzerinde bir yol $V_1, \ldots, V_C$ düğümlerinin bir dizisi şeklinde tanımlanır ve tüm $1 \leq i \leq C - 1$ düğümleri için $(V_i, V_{i+1}) \in E$ kenarları ifade eder. Bir çizgedeki yol

uzunluğu, kenar sayısıdır: $l = C - 1$. Bir çizgedeki iki düğüm arasındaki mesafe, bu iki düğüm arasındaki en kısa yolun uzunluğudur.

Boyut

G çizgesindeki düğümlerin sayısı $p = |V|$ çizgenin boyutudur.

Bağlantı sayısı

G çizgedeki kenarların sayısı $l = |E|$, çizgede bulunan bağlantısa sayısana denir.

Mesafe

- *Yönlendirme Mesafesi:* Yönlendirme Mesafesi, çizgedeki rota boyunca bulunan bağlantıların sayısıdır.
- *Ortalama Mesafe:* Ortalama mesafe, tüm geçerli rotalardaki ortalama bağlantı sayısıdır.

Derece ve çap

Bir ağ iletişim çizgesi $G = (V, E)$'nin ana yapısal özelliklerini aşağıdaki gibi tanımlanır:

- *Düğüm derecesi (degree):* deg bir düğümün gelen bağlantılarının ve giden bağlantılarının sayısıdır. Yönlendirilmiş çizgelerde, bir s tepe noktası için, $deg(s) = deg_{gelen}(s) + deg_{giden}(s)$ olarak tanımlanır. Tüm düğümler aynı dereceye sahip olduğunda, *G* çizgesi düzenli bir çizge olur ve düzenli çizgenin derecesini $deg(G)$ ile gösterilir.
- *Çap (Diameter):* *G* çizgesin herhangi iki düğümü arasındaki en büyük mesafe çizgenin çapıdır ve $max\{ec(y)|y \in V(G)\}$ olarak tanımlanır.
- *Yarıçap (radius):* *G* çizgesin herhangi iki düğümü arasındaki en küçük mesafeyi temsil eder ve $min\{ec(y)|y \in V(G)\}$ olarak tanımlanır.
- *Dış merkezlik (eccentricity):* Çizgedeki bir y köşesinin dış merkezliği, y ile çizgede bulunan başka bir köşe arasındaki en büyük mesafedir ve $ec(y)$ ile ifade edilir.

Bağlantı ve ikiye bölme

Uygulamada, çözülecek problemin veri seti boyutlarına göre bir bilgisayar kümesinin düğüm sayısını çok artırmak gerekebilir. Bu, ölçeklenen jenerik topolojiler yani genişletilebilir topolojiler oluşturabilmeyi gerektirir. Aşağıdaki kavramları tanımlayarak alt topolojilerden özyinelemeli topolojiler karakterize edilebilir:

- *Bağlanabilirlik (connectivity):* Bağlı bir çizgeden birbirine bağlı olmayan iki çizge elde etmek için kaldırılması gereken minimum bağlantı (kenar) sayısı olarak tanımlanır.
- *İkiye bölme (bisection) bant genişliği, b*: İkiye bölme bant genişliği, iki eşit yarıya bölünmüş olan (ikiye bölme) ağın yarımları arasında veri aktarırken elde edilebilecek maksimum bant genişliğidir. Ağın performansını ölçmek için önemli bir kriterdir. Daha yüksek bir ikiye bölme bant genişliği, ağın sistemin farklı bölümleri arasında eşzamanlı veri aktarımı için daha fazla kapasiteye sahip olduğunu gösterir.

Hamilton yolu, Hamilton döngüsü (Hamiltonian path, Hamiltonian circuit):
Hamilton yolu bilgisayar biliminde, veri modellemesi için kullanılan bir rotalama yöntemidir. Çizge teorisinde tanımı oldukça basittir. Bir yolun Hamilton yolu olarak nitelendirilebilmesi için, bu yol her düğümü bir kez ziyaret etmesi ve daha önce gidilmiş olan bir düğüme tekrar gitmemesi gerekir. Eğer Hamilton yolunda, başlangıç düğümü ile bitiş düğümü arka arkaya ise yani yol başladığı yerde bitiyorsa veya tam bir döngüyü tamamlıyorsa Hamilton döngüsü olarak adlandırılır.

İyi bir ağ topolojisi için yönergeler:
İletişim ağı için iyi bir topoloji oldukça önemlidir. Topoloji, bir dizi özelliği karşılayan bir çizge ailesidir. Aşağıda verilen maddeler iyi bir topoloji için gerekli olan yapılandırmalardır:

- Düşük bir donanım maliyeti elde etmek için düzenli ağ derecesi en aza indirilmelidir.

- İletişim ağı tarafından kullanılan kısa yolları elde etmek için ağ çapı en aza indirilmelidir.
- Daha büyük hacimli verileri işlemek (ölçeklenebilirlik) için *p*'yi (düğüm/işlemci sayısı) arttırarak ağ boyutu maksimize edilmelidir.

Topolojileri karşılaştırmak için bu özellikleri olumlu ve olumsuz olacak şekilde aşağıdaki şekilde listelenebilir.

Bir topolojinin olumlu yönleri:

- Düzgün veya simetrik olması
- Topolojinin özelliklerini korurken topolojiyi büyütme veya küçültme özelliği
- Paralel bir sistemin performansını artırmak için ölçeklenebilirlik özelliği
- Diğer sanal topolojileri kolayca simüle etme özelliği
- Topolojide mesaj yönlendirme kolaylığı

Bir topolojinin olumsuz yönleri:

- Yönlendirme mesajları için yüksek maliyet veya yüksek karmaşıklık (büyük derece) değerine sahip olması
- Donanım arızasına dayanıklı olmaması (düşük dereceli veya zayıf bağlantı)
- İletişim ilkelleri için verimli olmaması (düşük derece ve büyük çap)
- Yüksek performanslı bilgi işlem için verimli olmaması yani küçük *p* değerine sahip olması (topolojinin yalnızca küçük boyutlar için kullanılabilir olması)

2.4. AĞ TOPOLOJİSİ

Ağ topolojisi, bir iletişim ağında bulunan düğümler, bağlantılar gibi öğelerinin düzenlenmesine denmektedir [1]. Bilgisayar bilimlerinde ağ topolojisi, bir ağındaki

elemanların konumlandırılma ve bağlanma [7] yapısıdır, mantıksal veya fiziksel olarak tasvir edilebilmektedir. Bir çizge teorisi uygulaması olan ağ topolojisinde iletişim kuran cihazlar düğümler olarak ve cihazlar arasındaki bağlantılar düğümler arasındaki bağlantılar veya hatlar olarak modellenir [6]. Fiziksel topoloji, bir ağın çeşitli bileşenlerinin yerleştirilmesidir. Buna örnek olarak cihaz konumu ve kablo kurulumu verilebilir. Bir ağ içindeki veri akışı ise ağın mantıksal topolojisi ile gösterilir. İki farklı ağ aynı mantıksal topolojiye sahip olabilir ancak fiziksel bağlantılar, düğüm mesafeleri, sinyal türleri veya iletim hızları açısından farklılık gösterebilir.

Genel olarak çok bilinen ve kullanılan birkaç statik ağ topolojisi vardır. Bunlar halka, klik, örgü, torus, hiperküp ağ topolojileridir.

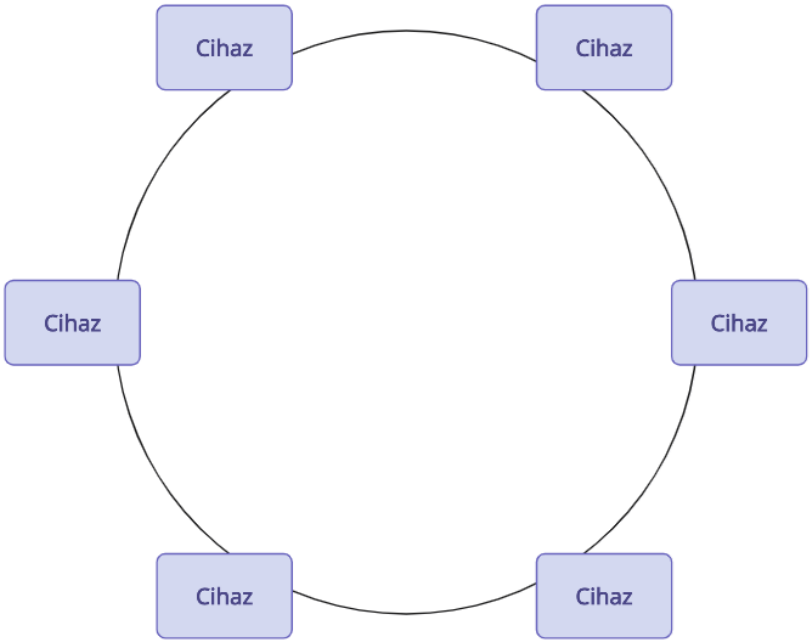
2.4.1. Halka (Ring) Ağları

Şekil 2.1, halka topolojisinin bir örneğini göstermektedir. n düğüm içeren bir ağ için halka topolojisinin ana özellikleri şunlardır:

$$Çap = \left\lfloor \frac{n}{2} \right\rfloor \tag{2.1}$$

$$İkiye\ bölme\ bant\ genişliği = 2 \tag{2.2}$$

Halka topolojisinin uygulanması yüksek maliyetli değildir, ancak büyük ölçekli sistemler için kullanımı tercih edilmemektedir. Bunun nedeni halka ağlarında ölçeklenebilirlik zorluklarının bulunması, tek nokta arızası riski, bant genişliği ve gecikme sorunlarının olmasıdır.



Şekil 2.1. Halka topolojisi ([6] referans alınarak çizildi)

Halka topolojisi üzerine yapılmış birçok çalışma bulunmaktadır. Son zamanlarda halka ağı üzerine çeşitli araştırmalar yapılmıştır. Atta ve Sen'in çalışmasında VoD yani isteğe bağlı video hizmetlerinde içerik yerleştirme için omurga ağında halka topolojisi kullanılarak hub'ların konumlarının bulunması, her segmentin tam olarak bir hub'a atanması, her kullanıcının taleplerine göre tek bir hub'a tahsis edilmesi ve talepleri kullanıcılardan hub'lara yönlendirmek için en uygun yolların bulunması problemleri ilk defa tek bir optimizasyon problemi olarak ele alınmış ve önerilen metotların etkili olduğu test sonuçları ile gösterilmiştir [8]. Eydi ve arkadaşlarının çalışmasında düzensiz halka ve merkez ağ yapısı tasarımı için halka topolojisi kullanılarak iletişimle ilişkili maliyetleri azaltmak amaçlanmıştır ve ağ çözümlemesi için daha iyi sonuçlar elde edilen bir algoritma önerilmiştir [9]. Gerçek dünyada yaygın olan düzensiz halka ve hub ağ yapısında aynı halka üzerindeki düğümler arasındaki ortalama mesafeyi analitik olarak tahmin etmek için Fu ve arkadaşları bir model ve algoritma geliştirmişler ve kabul edilebilir sonuçlar elde etmişlerdir [10]. Farklı problemlerde halka yapıda yapay sinir ağları [11][12] kullanılarak çözüm elde edilmiş ve diğer bir çalışmada farklı sınıfları ve öncelik yapısı bulunan kuyruklara sahip bir halka-topoloji stokastik ağı ele alınmıştır [13]. Kullanılan yaklaşım,

modelin kararlı olması ve ergodik davranış göstermesi durumunda kararlılık koşulu ile halkanın farklı bölümlerinin doluluk oranı arasındaki ilişkiyi açıkça kullanmıştır.

2.4.2. Klik (Clique) Ağları

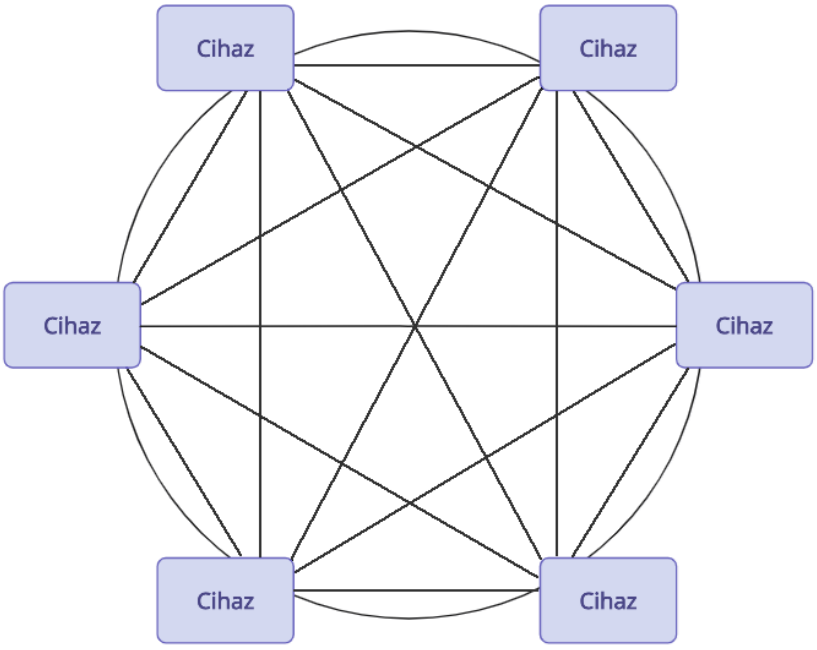
Şekil 2.2, klik olarak da adlandırılan tamamen bağlı bir ağı göstermektedir. Ağda bulunan düğüm sayısı n olduğu varsayıldığı zaman ağın ana özellikleri şunlardır:

$$Derece = n - 1 \quad (2.3)$$

$$Çap = 1 \quad (2.4)$$

$$İkiye\ bölme\ bant\ genişliği = \left\lceil \frac{n}{2} \right\rceil * \left\lceil \frac{n}{2} \right\rceil \quad (2.5)$$

Klik ağı, performans açısından en uygun ağdır. Ancak, bağlantı sayısı düğüm sayısıyla birlikte karesel olarak arttığı için bu ağın uygulanması çok maliyetlidir. Büyük ölçekli sistemler için pratikte kullanılamaz.



Şekil 2.2. Klik topolojisi ([6] referans alınarak çizildi)

Klik ağları analiz, veri madenciliği, sınıflandırma, ulaşım ağı modeli, tıp araştırmalarında ve daha birçok farklı alanda kullanılmıştır. [14] yeni bir topoloji

modeli olarak ideal n derinlikli klik ağ topolojisi geliştirmiş ve bu ağa sahip otobüs ulaşım ağı modeli oluşturmuştur. Kliklere dayalı diğer yeni bir ağ yaklaşımını [15] ortaya koymuştur. Bu yaklaşım, yalnızca klikler ağının özelliklerini yakalamak için bir dizi yeni endeksten değil, aynı zamanda karmaşık klikler ağlarını karakterize etmek için bir yöntemden de oluşur. Üçlü biçimsel kavram analizine dayalı dinamik sosyal ağlarda k-Klik madenciliği [16], yapısal beyin ağı sınıflandırmasında klik alt çizgeleri öğrenme [17], kliklerin anlamsal ağlarına dayalı sözcük tekrarı yoluyla metinsel bağdaşıklık endeksleri [18] yapılan çalışmalardan bazılarıdır. Ayrıca rastgele klik ağlarının yapısal özellikleri de incelenmiştir [19].

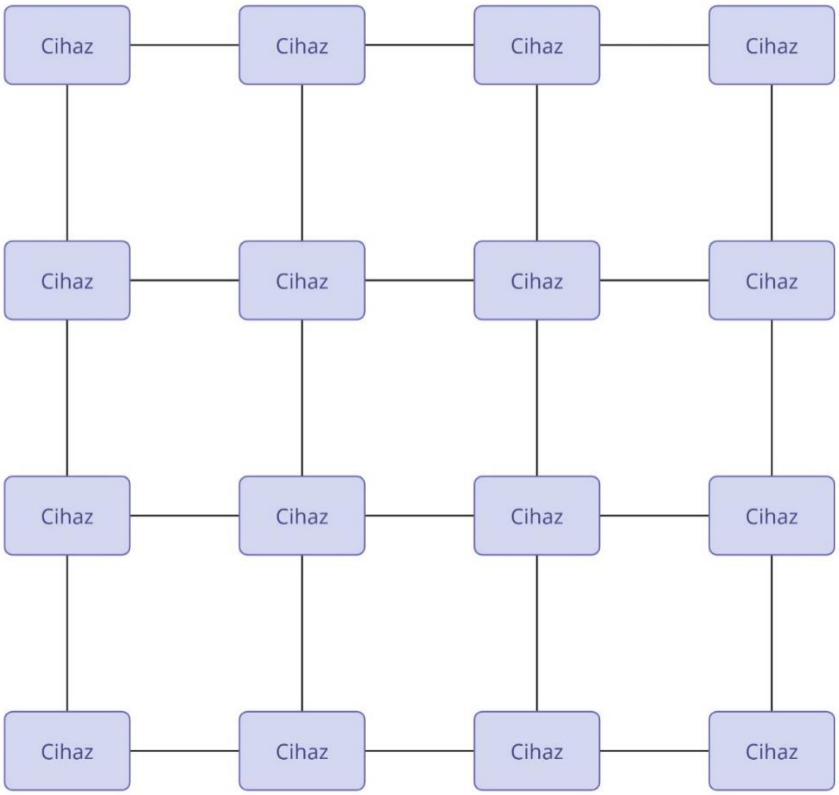
2.4.3. Örgü (Mesh) Ağları

Şekil 2.3, 2 boyutlu bir örgü ağ örneğini göstermektedir. Uygulamada daha fazla boyutlu (çoğunlukla 3 boyutlu) ağlar kullanılabilir. 2 boyutlu *n* düğüme sahip bir ağın ana özellikleri aşağıda verilmiştir:

$$Derece = 4 \tag{2.6}$$

$$Çap = 2 * \sqrt{n} - 1 \tag{2.7}$$

$$İkiye\ bölme\ bant\ genişliği = \sqrt{n} \tag{2.8}$$



Şekil 2.3. 2B Örgü topolojisi ([6] referans alınarak çizildi)

Şekil 2.3'ün, çizgedeki her düğümün aynı zamanda bir uç nokta (işlemciler-daireler) olduğu statik bir ağın (doğrudan ağ) açık bir temsilini sağlar. 2-B ağ, 2-B uzay problemlerine karşılık gelen ve görüntü işleme gibi iletişimin çoğunlukla yerel olduğu hesaplamaları çalıştırmak için çok uygundur.

2B ağ topolojisi paralel ve dağıtılmış sistemlerin mimarisinde önemli bir rol oynar. Bu topoloji genellikle kablosuz ağlarda [20], Çip Üzerinde Ağ (NoC) mimarisinde [21][22][23][24][25], süper bilgisayarlarda [26] ve paralel işleme ve yüksek performanslı hesaplamanın gerekli olduğu benzer uygulamalarda kullanılır. Farklı problem türleri için 2 boyutlu ağ topolojisi üzerine çeşitli çalışmalar bulunmaktadır. Ma vd. [27] çok çekirdekli sinir ağı çipinde mantıksal çekirdekleri fiziksel devre düğümlerine eşleyen ve yönlendirme kilitlenmesini önleyen bir optimizasyon tekniği sunmuş, önerilen algoritmanın çekirdekler arası iletişim için yönlendirme süresinden ve güç tüketiminden oldukça verimli bir şekilde tasarruf sağladığını ve yönlendirme çeşitliliğinin önemli ölçüde iyileştirildiğini, etkin nokta yollarının optimizasyondan sonra zikzak ve komşu eşleme algoritmalarına kıyasla büyük ölçüde azaldığını

göstermiştir. Amin vd. [28] tarafından önerilen iHPSA, NoC haritalaması için Geliştirilmiş Parçacık Sürüsü Optimizasyonu ve Benzetilmiş Tavlamayı birleştirir ve görevleri, iletişim bant genişliğine göre kümelere ayırmak için bir makine öğrenimi yöntemi olan K-ortalamalar kümeleme algoritmasını kullanır. Deneysel sonuçlar, iHPSA'nın iletişim maliyetini, gücü, enerjiyi ve gecikmeyi azaltan verimli bir uygulama yaklaşımı sunarak doğadan ilham alan diğer algoritmalardan daha iyi performans gösterdiğini ortaya koymuştur. Bani-Mohammad vd. [26] kullanılan iletişim modellerinin (Yakın Komşu, Halka, Böl ve Fethet Binom Ağacı, Hızlı Fourier Dönüşümü ve Rastgele) 2 boyutlu ağ bağlantılı çoklu bilgisayarlar için önerilen bitişik olmayan tahsis stratejilerinin performansı üzerindeki etkisini araştırmıştır. Bunlar Paging(0), Multiple Buddy Strategy, Adaptive Non-contiguous Allocation stratejisi, ve Greedy Available Busy List stratejisidir. Elde edilen sonuçlarda, Greedy Available Busy List yönteminin genel iletişim modelleri için iş geri dönüş süresi açısından en iyi genel performansa sahip olduğunu göstermiştir. En kısa yol problemi 2 boyutlu NoC örgü ağı üzerinde çalışılan diğer problemdir, Inam vd. [21] ve Onaizah vd. [22] tarafından yapılan yayınlar bu çalışmaların örneklerindendir. Inam vd. daha iyi genişletilebilirlik ve küçük düğüm derecesi sağlayan yeni bir ağ topolojisi sunarak bu ağ topolojisi üzerinde en kısa yol problemi için bir algoritma önermiştir. Bu algoritma, 2B örgü ağlarında uygulanan en kısa yol algoritması ile benzer yol uzunluğu elde etmiştir. Onaziah vd. ise TriBA-NoC olarak adlandırılan 40 düğümlü TriBA (Triplet tabanlı) NoC mimarisini önermiş, çok çekirdekli 40 karo DDR-3 donanımında uygulamıştır. Çalışmada TriBA-NoC sistemleri için yeni yönlendirici mimarisi ile TR-132 en kısa yönlendirme yolu algoritması uygulanmış, 40-çekirdekli bir TriBA-NoC modelinin çeşitli trafik modelleri için ortalama paket gecikme süresinin sonuçları gösterilmiştir. Önerilen 40 çekirdekli TriBA-NoC modelinin sonuçları TriBA modeli ile karşılaştırıldığında daha iyi performanslar elde edildiği gözlenmiştir. NoC mimarisi üzerinde yapılan yönlendirme algoritmaları çalışmalarından bazıları, Manzoor vd. [29], Ji vd. [24], Gogoi vd. [25] tarafından yapılmıştır. Manzoor vd. PAAD (Partially Adaptive and Deterministic) adında yeni bir kilitlenme olmayan, tıkanıklığa duyarlı ve yönlendirme için diyagonal koordinatları kullanan bir yönlendirme algoritması sunmuşlardır. Çalışmada ele alınan ana sorun, yönlendirme kilitlenmelerinin nasıl önleneceği ve bununla birlikte tıkanıklıkla nasıl başa çıkılacağıdır. XY, WF (West

First), ve DYAD (Dynamic Adaptive and Determistic Routing) algoritmaları ile karşılaştırıldığında, PAAD deneysel sonuçlarının olumlu olduğu ve ölçeklenebilir performansa sahip büyük ölçekli NoC uygulamalarında algoritmanın potansiyelini ortaya koyduğu gözlenmiştir. Ji vd. ağ tıkanıklığını ele almak ve NoC performansını artırmak için Yazılım Tanımlı NoC (SDNoC) tabanlı Tamamen Uyarlanabilir Yönlendirme (FAR) algoritması önermektedir. Değerlendirme sonuçları, önerilen FAR'ın ortalama paket gecikmesi ve doygunluk veriminde tipik yönlendirme şemalarına göre sırasıyla %8,2 ve %6,1'e varan bir performans artışı sağladığını göstermektedir. OBL (Output Buffer Length) ve DBSS (Destination-Based Selection Strategy) ile karşılaştırıldığında yerleşim alanı ve güç tüketimi en az %6,2 ve %3,9 daha azdır. Gogogi vd. NoC üzerinde hataya duyarlı bir yönlendirme yaklaşımı ile bunu destekleyecek bir yönlendirici mimarisi önermiştir. Diğer hataya dayanıklı yaklaşımlar olan "Minimal path Connection retaining Fault-tolerant" ve "Self-Healing" ile karşılaştırıldığında sıfır hata ile sırasıyla gecikmede 1,89 kat ve 0,92 kat ek yüke karşı verimde ortalama 13,27 kat ve 4,81 kat iyileştirme elde eder. Kablosuz ağlar için 2 boyutlu ağ mimarisi üzerinde Chai vd. [20] hibrit kablosuz ağ yapısına iyi uyum sağlayan hem proaktif hem de reaktif yönlendirme protokollerini birleştiren hibrit yönlendirme protokolü geliştirmiştir. Ji vd. [23] çalışmasında, 20 farklı algoritma için meta-sezgisel algoritmalar kullanarak NoC üzerinde çeşitli NP zorlukta olan problemleri optimize etmeye yönelik kapsamlı bir araştırma yapmışlar ve performanslarını karşılaştırmışlardır. Bu çalışmada elde edilen sonuçlardan biri, Hibrit meta sezgisel algoritmaların çeşitli problemler için daha iyi çözümler bulabildiği, ancak genellikle daha yüksek hesaplama yükü ve depolama gereksinimlerine sahip olduklarıdır.

2.4.4. Torus Ağları

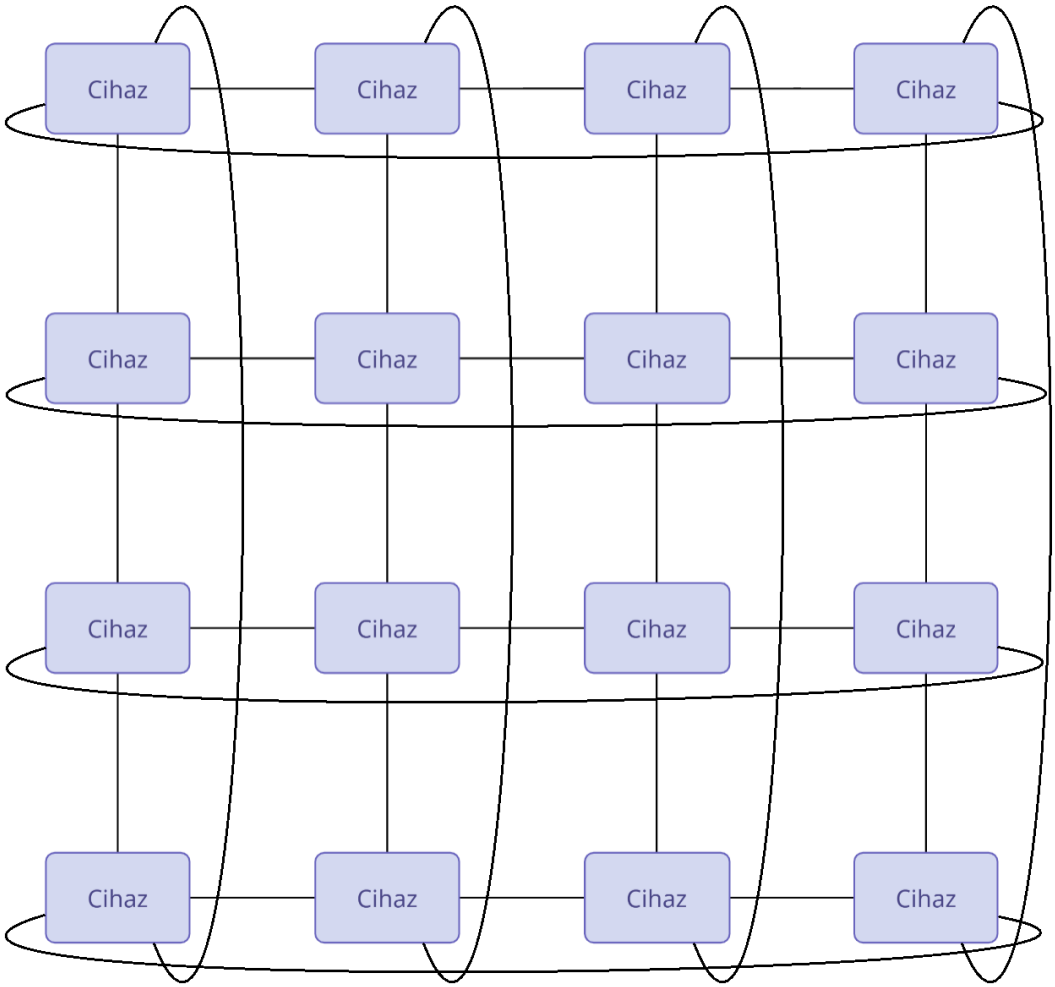
2 boyutlu torus ağı, her boyuttaki ilk ve son düğümlerinin ek bağlantılar ile birbirine bağlaması dışında 2 boyutlu örgü ağa benzer. Şekil 2.4, 2B torus ağının bir örneğini göstermektedir. 2B torus ağının ana özellikleri şunlardır:

$$Derece \ = \ 4 \tag{2.9}$$

$$Çap \ = \ 2 * \left\lfloor \frac{\sqrt{n}}{2} \right\rfloor \tag{2.10}$$

$$İkiye\ bölme\ bant\ genişliği \ = \ \ 2 * \sqrt{n} \tag{2.11}$$

Örgü ağı ile karşılaştırıldığında, torus topolojisi, ağ bağlantıları açısından neredeyse aynı maliyet karşılığında daha küçük bir çap ve daha büyük bir ikiye bölme bant genişliği sunar. 2B torus ağı, iyi ölçeklenebilirlik özelliğine sahiptir ve bu nedenle, çok büyük ölçekli paralel sistemlerde kullanılmıştır [30][31].



Şekil 2.4. 2B Torus topolojisi ([6] referans alınarak çizildi)

Yeni nesil devasa paralel bilgisayar sistemi için yeni bir torus ağı türevi olan midimew bağlantılı torus ağı [30] ortaya konulmuştur. 3 boyutlu torus ağı için ölçeklenebilir sıra haritalama algoritması [31], torus topolojisi için çizge tabanlı yönlendirme algoritması [32], sanal kanal yük dengeli yönlendirme algoritması [33], yarı çapraz torus ağları için topolojik özellikler ve yönlendirme algoritması [34], 2B torus ağlarında ağaç tabanlı çok noktaya yayın algoritması [35], Hamilton döngü modeliyle solucan deliği yönlendirmeli 2 boyutlu torus ağlarında çok noktaya yayın iletişimi [36] çalışmalarında torus ağları üzerinde yeni algoritmalar sunulmuştur.

2.4.5. Hiperküp Ağları

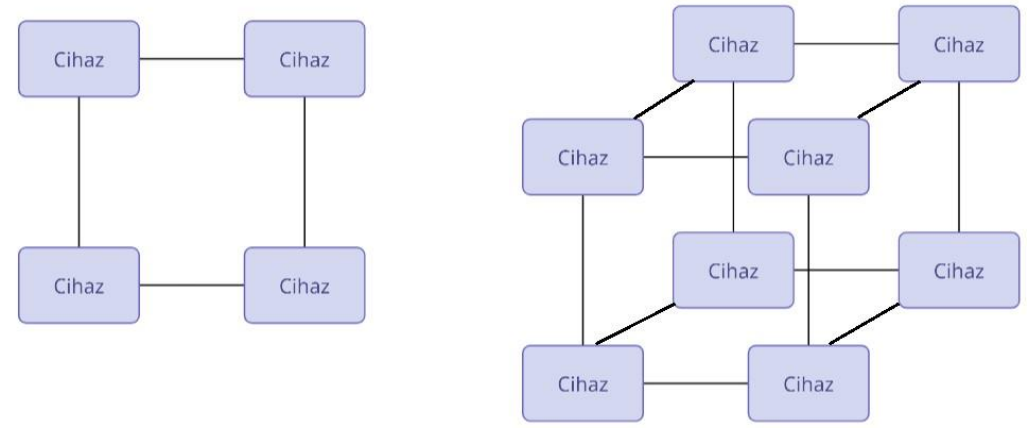
Hiperküp iletişim ağı, 2'nin kuvveti olarak ifade edilebilen n düğümün bağlanmasıyla oluşturulur. B boyutlu bir hiperküp ağında bulunan düğüm sayısı $n = 2^B$ kadardır. Bir hiperküp temelde her boyutta iki düğüm bulunan çok boyutlu bir örgü ağıdır.

$$Derece = B \tag{2.12}$$

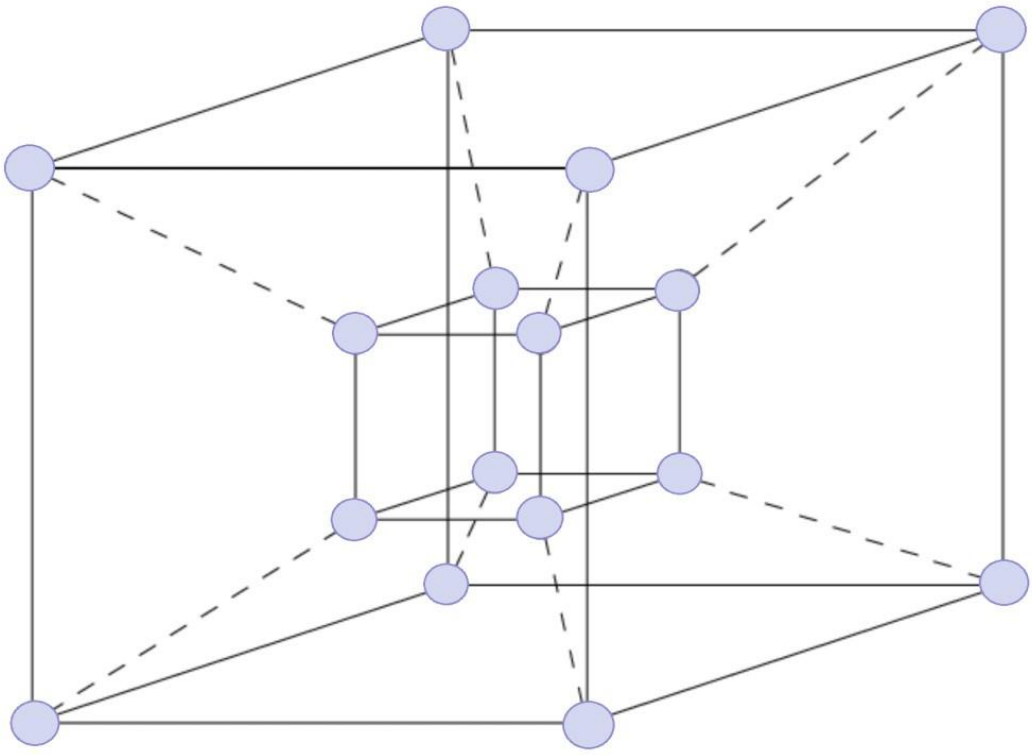
$$Çap = B \tag{2.13}$$

$$İkiye\ bölme\ bant\ genişliği = \frac{n}{2} \tag{2.14}$$

Burada m, ağdaki düğümleri etiketlemek için gereken bit sayısıdır ve B'ye eşittir. Dolayısıyla, ağda 4 düğüm varsa, 2 boyutlu hiperküp ağındaki tüm düğümleri temsil etmek için 2 bit gerekir. Ağ, ikili temsillerinde sadece bir bit farklılık gösteren düğümlerin bağlanmasıyla oluşturulur. Buna genellikle ikili (binary) etiketleme denir. Bir 3B hiperküp ağ ağı, 8 düğümü ve 12 kenarı olan bir küp olacaktır. İki 3B ağı kopyalayarak ve etikete bir bit en önemli bit (most significant bit MSB) olacak eklenerek 4B hiperküp ağı oluşturulabilir. Yeni eklenen bit, 3B hiperküpün ilki için "0" ve diğer 3B hiperküp için "1" olmalıdır. İlgili bir bit değiştirilmiş MSB'lerin köşeleri, daha yüksek hiperküp ağı oluşturmak için bağlanır. Bu yöntem, $(m-1)$-bit ile temsil edilen herhangi bir m-bit hiperküp oluşturmak için kullanılabilir.



(a) (b)



(c)

Şekil 2.5. a) 2 boyutlu hiperküp, b) 3 boyutlu hiperküp, c) 4 boyutlu hiperküp ([6] referans alınarak çizildi)

Hiperküp çizgeleri çeşitleri bilgisayar mimarilerinde sıklıkla kullanılmaktadır. Yapılan çalışmalarda Hayes ve ark. 1990'larda süper bilgisayarları uygulamak için hiperküp mimarisini kullanmıştır [37] ve daha önceki makinelerin aksine, çok

kullanıcılı UNIX benzeri bir programlama ortamı sağlamak için hiperküpün doğal homojenliğinden yararlanmış, bunun yanı sıra son derece yüksek G/Ç veri iletim hızlarını desteklemiştir. Sudoküpler (psedocub) hiperküplerle benzer özelliklere sahip olduğundan Nieminen ve ark. sudoküpleri hiperküp kullanarak tanımlamıştır [38]. Chae ve ark. genel kübik çizgeler üzerinde çalışmıştır [39], Harary ve ark. hiperküp çizgelerinin kapsamlı analizini sunmuşlardır, önce temel özellikler gözden geçirilmiş, alt grafikleri ve döngüleri ile tanımlanan n-küpün özellikleri incelenmiş ve son olarak çeşitli gömme ve paketleme problemleri tartışılmıştır [7]. Mao ve Nicol k-ary n-küp [40] adlı çizgeler üzerinde çalışmış, k-ary n-küplerin kombinatoryal özelliklerini incelemiştir. Çalışmada özellikle, maksimum kenar sayısına sahip belirli sayıda düğümün alt grafiğini karakterize etme problemi incelenmiştir. Bu teorik sonuçlar daha sonra dal-sınır bölümleme algoritmalarında bir alt sınırlama fonksiyonunu hesaplamak ve bazı düzensiz bölümlerin optimalliğini belirlemek için uygulanmıştır.

Birçok araştırmacı hiperküp ve türevlerini inceleyerek topolojik özelliklerini elde etmiştir. Örneğin Katlanmış Hiperküp, El-Amawy ve Latifi [41] tarafından incelenmiş ve temel özellikleri standart n-küpün özellikleri ile karşılaştırmıştır. Homojenliği ve simetrisi nedeniyle katlanmış hiperküpün, n-küpün düğüm ve kenar simetrisi gibi birçok çekici özelliğine sahip olduğu, buna ek olarak, katlanmış hiperküpün birçok iletişim açısından n-küpe göre daha üstün olduğu ve FHC'nin bire-bir ve yayınlama gibi birçok yönlendirme algoritmasında n-küpten daha iyi performansa sahip olduğu gösterilmiştir. Cahng ve Chen [42] yeni bir sınıf olarak artımlı olarak genişletilebilir katlanmış hiperküpü önermiş ve basit bir yönlendirme algoritması vererek özelliklerini açıklamışlardır. Bu ağ yapısı tamamlanmamış hiperküp, süperküp gibi çizgelerin yarısı kadar olmasına rağmen optimal hata toleransına sahip, neredeyse düzenli olduğu sonucuna varılmıştır. Başka bir hiperküp çeşidi olan Çapraz Küplerin topolojik yapısı Dong ve arkadaşları tarafından incelenmiştir. [43] ve bunun içine bir 3 boyutlu ağ ailesi yerleştirilmiştir, Efe [44] Çapraz Küpler kullanarak paralel mimariler için alternatif bir yol sunmuştur. Ağın çapı hiperküpün çapının sadece yarısı kadardır ve köşeler arasındaki ortalama mesafe daha küçüktür ve dilatasyon 2 gömme yoluyla bir hiperküpü simüle edebilir. Önerilen ağın temel özellikleri tartışıldıktan sonra, optimal yönlendirme ve yayın

algoritmaları geliştirilmiştir. Ağın bir SIMD mimarisi olarak yetenekleri, yarı grup hesaplamaları, matris çarpımı ve sıralama örnekleri verilerek gösterilmiştir. Bu algoritmaların her biri, hiperküp uygulamalarına kıyasla iletişim adımlarının sadece yaklaşık yarısını gerektirmektedir. Lai ve ark. istenilen bir torusu çapraz küpün içine yerleştirmek için dinamik bir programlama algoritması oluşturmayı önermiştir [45]. lgoritmanın zaman karmaşıklığı, istenilen torusun boyutuna göre doğrusaldır. Sonuç olarak, bir ayrık tori ailesi aynı çapraz küp üzerinde verimli ve paralel olarak simüle edilebilir.

Bunun yanı sıra, hiyerarşik ağlarla ilgili birçok çalışma bulunmaktadır. Abd-El-Barr ve Al-Somani [46] farklı türdeki hiyerarşik ara bağlantı ağlarını ve bunların topolojik özelliklerini inceleyip karşılaştırmıştır. Bu çalışmanın sonuçları, tüm hiyerarşik ağlar arasında Root-Folded Heawood ve Flooded Heawood'un, ağ çapı ve derecesinin çarpımı olarak tanımlanan en iyi ağ maliyetini sağladığını göstermektedir. Çalışma ayrıca HFCube(n, n)'ün en iyi paketleme yoğunluğunu, yani büyük NoC uygulaması için gereken en küçük çip alanını sağladığını göstermektedir. Ghose ve Desai [47] devasa dağıtılmış bellek için yeni bir ara bağlantı sistemi olan hiyerarşik kübik ağı (HCN) tanıtmıştır. HCN, düğüm başına yaklaşık yarısı kadar bağlantı kullanmasına rağmen, benzer bir hiperküpün çapının yaklaşık dörtte üçüne sahiptir. HCN tabanlı bir sistem ve hiperküp tabanlı bir sistem üzerinde çalışan çeşitli veri paralel uygulamalarının ihtiyaç duyduğu yönlendirme adımlarının sayısının yaklaşık olarak aynı olduğu tespit edilmiştir. Düzgün ve yerelleştirilmiş trafik modellerinin simülasyonu, HCN'nin hiperküpten daha iyi olduğunu ortaya koymaktadır. Hiyerarşik Genişletilmiş Fibonacci Küpleri (Hierarchical Extended Fibonacci Cubes) HEFC, Karci [48] tarafından önerilen yeni bir ara bağlantı ağı yapısıdır ve HEFC'lerdeki yönlendirme HCN'deki yönlendirme kadar kolaydır ve HEFC'nin ölçeklenebilirliği HCN ve hiperküpün ölçeklenebilirliğinden daha iyidir. Karci ayrıca hiyerarşik olarak tanımlanabilen yeni çizgeler ve Hiyerarşik Kübik çizgelerin alt çizgelerini önermek için Fibonacci serisini kullanmıştır ve bu çizgeler rekürsif olarak tanımlanabilen diğer çizgelere göre boyut olarak daha küçük yapıdadır [49]. Karcı ve Selçuk yeni hiperküp çeşitlerini tanıtmıştır. Bunlar; Fraktal yapıları kullanan Fraktal Kübik Ağ Çizgesi (FCNG) [3] ve hiperküp kullanan Bağlı Kübik Ağ Çizgesi'dir [50]. Selçuk tarafından tanıtılan Bağlantılı Kare Ağ Çizgeleri iki farklı tanım

kullanılarak elde edilebilir ve hiperküpün bir çeşidi olan bu çizge Hamilton çizgesidir [2].

2.5. YÖNLENDİRME

İyi bir yol seçimi, ağın paylaşılan kaynaklarına yönelik talebi dengelerken, genellikle ziyaret edilen düğüm veya bağlantı sayısı olarak ölçülen uzunluklarını en aza indirir. Bir yolun uzunluğu, ağ üzerinden bir mesajın gecikmesini doğrudan etkiler. Bir topoloji seçildikten sonra, bir mesajın hedefe ulaşmak için ağ üzerinde gidebileceği birçok olası yol bulunabilmektedir. Yönlendirme, ağ topolojisindeki düğümler arasındaki bu olası yollardan herhangi birininin seçilmesi işlemidir.

Bir rota veya yol, $P = \{c_1, c_2, \dots, c_k\}$ kanallarının sıralı bir kümesidir; burada c_i kanalının çıkış düğümü c_{i+1} kanalının giriş düğümüne eşittir. Kaynak c_1 kanalının girişidir ve hedef c_k kanalının çıkışıdır. Bazı ağlarda, kaynak düğümden hedef düğüme yalnızca tek bir yol varken, torus veya hiperküp gibi ağlarda kaynaktan hedefe birçok olası yol vardır. Çok sayıda yol olduğunda, iyi bir yönlendirme algoritması, optimum yolu seçer. Örnek olarak yol haritası ele alındığında, topoloji yol haritasını, yolları ve kavşakları temsil ederken, yönlendirme yöntemi her kavşakta bir noktadan diğer bir noktaya doğru giden arabanın hangi yöne döneceğine karar vererek arabayı yönlendirir.

Amaca ve karmaşıklığa göre iki ana tip yönlendirme problemi vardır [51]. Yol bulma problemlerinin birinci kategorisinde en kısa yol ana problemdir. Diğer yönlendirme problemleriyle karşılaştırıldığında bu problemler oldukça basittir. Tüm düğümleri ziyaret ederek oluşturulan tur inşa problemleri ikinci grubu oluşturmaktadır. Belirli bir ağ içerisinde bir turun tamamının oluşturulması birinci kategorideki problemlerden daha karmaşıktır. Örnek olarak, bir ağ şeklinde temsil edilen gerçek dünya problemlerini çözmek için gerçek mesafe ve coğrafi bilgi sistemi (CBS) kullanılmaktadır.

Yol bulma problemleri genelde en kısa yol problemleridir (Shorthest Path Problem SPP) ve iki düğüm arasındaki minimum toplam maliyeti bulur. Maliyet, zaman, mesafe, gider gibi değişkenler olabilir. Tek kaynaklı en kısa yol problemi, kaynak ve hedef köşeler arasında bir yol bulmak içindir. Bu problemler için farklı algoritmalar bulunur. Ağdaki tüm köşe çiftleri arasındaki yolları bulmak, tüm çiftler en kısa yol problemi (All Pair Shortest Path Problem) olarak adlandırılır. Tur inşa problemleri, belirli kriterleri karşılayan turlar veya yollar oluşturmakla ilgili çeşitli kombinatoryal optimizasyon problemleridir ve üç sınıfa ayrılabilir: Gezgin Satıcı Problemi (Traveller Salesman Problem TSP) ilk ve en yaygın olarak bilinenidir, Araç Rotalama Problemi (Vehicle Routing Problem VRP), birçok satıcının yer aldığı genelleştirilmiş TSP versiyonudur ve son olarak Otobüs Rotalama Problemi (Bus Routing Problem BRP) rota dengeleme özelliğine sahip başka bir TSP çeşididir, zaman penceresi kısıtlamalarına sahip olabilir ve veri yolları aynı veya farklı kapasitelere sahip olabilir.

SPP birçok farklı türde ara bağlantı ağında uygulanmıştır. [52]'de en kısa yol yönlendirmesi hiyerarşik hiperküp ağlarda optimum tümden tüme kişiselleştirilmiş değişim gömme için uygulanmıştır. [53]'te düzensiz halka ve göbek ağı için yeni modelle ortalama en kısa mesafe bulunmuştur. [22]'de yonga levha üzerindeki büyük ölçekli ağ ağı için bire bir en kısa yol problemi incelenirken, torus ağı için bire bir en kısa yol problemi [54]'te incelenmiştir

2.6. GEZGİN SATICI PROBLEMİ (HAMİLTON YOLU)

Gezgin Satıcı Problemi, çözümü için geniş bir arama alanına sahip olduğu ve dolayısıyla çalışma süresinin polinom olmadığı anlamına gelen NP-zor optimizasyon problemlerinden biridir [33]. Gezgin Satıcı Probleminde (TSP) amaç, satıcının bulunduğu şehirden başlayarak her şehre bir kez uğrayıp başladığı şehre geri dönerek yapacağı en kısa turu bulmaktır. Bu problemde herhangi iki şehir arasında bir yol olduğu ve bu yolun uzunluğunu bilindiği varsayılmaktadır. TSP, çizge teorisi dilinde, şehirlerin tepe noktasıyla ve yolların da köşeler arasındaki kenarlarla temsil edildiği (düz) bir çizge üzerinde en kısa Hamilton döngüsünü bulmaktır. Problemin anlaşılması kolay ama çözülmesi zordur.

Hamilton yolu, bir çizgedeki her düğümden yalnızca bir kez geçen yoldur ve adını 19. yüzyıl matematikçisi William Hamiltonian'dan almıştır. Bir Hamilton yolunun başlangıç düğümü ile bitiş düğümü arasında sadece bir kenarsa, yol tam bir döngüyü tamamlayarak oluşturulabiliyorsa bu yollara Hamilton döngüsü adı verilir. Hamilton döngüsünü içeren bir çizge aynı zamanda Hamilton yolunu da içerir, ancak bunun tersi her zaman doğru değildir. Hamilton döngüsüne sahip çizgelerde Hamilton çizgeleri denir.

TSP birçok farklı ara bağlantı topolojileri üzerinde çalışılmıştır. TSP için halka ve hiperküp üzerinde paralel olarak uygulanan Karınca Kolonisi Optimizasyon Algoritmasının özelliği Min Max Karınca Sistemi Algoritmasını farklı başlangıç değerleri ile paralel işlemciler üzerinde çalıştırarak en iyi değeri bulmaktır [55]. Hiperküp üzerinden TSP için Kimyasal Reaksiyon Optimizasyonu [56] paralel olarak uygulanmıştır ve Genetik Algoritmaya göre çalışma zamanı ve maliyet olarak daha iyi sonuçlar elde etmiştir. Hiperküp üzerine çalışılan diğer bir algoritma da Paralel Gri Kurt algoritmasıdır [57] ve yapılan çalışmaya göre Paralel Genetik Algoritma ve Paralel Kimyasal Reaksiyon Optimizasyon Algoritmasına göre çalışma zamanı, hızlanma ve paralel verimlilik olarak daha iyidir. [58], NISQ devre derleme probleminin torus ağındaki TSP ile aynı olduğunu göstermektedir. TSP ayrıca paraziti azaltmak amacıyla kanal tahsis şeması için örgü ağlar için de çalışılmıştır [59]. Bunlar TSP üzerine farklı topolojilerde yapılan çalışmalardan sadece birkaçıdır.

2.7. YAYIN

Bilgisayar ağlarında yayınlama, ana bilgisayarlar arasındaki iletişim için kullanılan ağ üzerinden veri gönderilmesi anlamına gelir. Ağ oluşturmada kullanılan yayın türleri aşağıdaki gibidir [60]:

- Tek noktaya yayın iletimi (unicast), verileri bir göndericiden bir alıcıya göndermek için kullanılan bire bir iletimdir.
- Yayın iletimi (broadcast), mesajın bir göndericiden tüm potansiyel alıcılara aynı anda bir ağ içerisinde gönderildiği bir iletişim yöntemidir.

- Çok noktaya yayın iletimi (multicast), tek bir göndericinin aynı anda belirli bir gruba, birden fazla alıcıya veri iletmesine olanak tanıyan bir iletişim yöntemidir.

Yayınlama işlemi, düğüm çiftleri arasında sıralı veri iletimi ile gerçekleşir. Yayın iletişimine genellikle görüntü işlemede, algoritmaların veya bilimsel hesaplamaların paralel uygulanmasında, büyük veri dizilerinin sistem düğümleri arasında yayılmasında ve çeşitli veri işleme faaliyetlerinin yürütülmesinde ihtiyaç duyulur. Mesajın ağ üzerinden yayınlanmasının iki yolu vardır [61]. Birincisi, birden hepsine (one to all) yayın yani mesajın kaynaktan ağ üzerindeki tüm düğümlere iletilmesidir. Yayınlanması gereken m boyutlu veriler başlangıçta yalnızca kaynak düğümde mevcuttur. Prosedürün sonunda, her düğümde bir tane olmak üzere başlangıç verilerinin n (ağdaki düğüm sayısı) kopyası bulunur. Hepsinden hepsine (all to all), aynı zamanda toplam değişim (total exchange) veya dedikodu (gossip) olarak da adlandırılan diğer yayın türüdür. Bu yayında her düğüm ağdaki diğer düğümlere bir mesaj iletir. Bu yayın birden hepsine yayın türünün genelleştirilmiş hali gibidir, tüm düğümler aynı anda birden hepsine yayın başlatır. Bir düğüm tüm düğümlere aynı mesajı gönderir, ancak farklı düğümler farklı mesajlar yayınlayabilir. Torus ve ızgara ağları[62], örgü ağlar [63], yıldız ağlar [64] ve hiperküp ağların türleri [3][64][65][66] gibi seyrek ağlar için yayın algoritmaları mevcuttur. En çok bilinen ve kullanılan yayın algoritmaları Recursive Doubling, Network Partitioning ve Extending Dotained Node algoritmalarıdır [67].

2.7. BÖL VE FETHET

Algoritma tasarlamanın farklı yöntemleri vardır. Yaklaşımlardan biri, adından da anlaşılacağı gibi sorunu alt problemlere bölerek orijinal problemin çözülmesini kolaylaştıran bir yöntem olan böl ve fethet yöntemidir. Bu yöntem, bir alt problemlerin elde edilen çözümlerini birleştirerek asıl problemin çözümüne ulaşır [68]. Böl ve fethet yaklaşımı program modülerliğini artırır ve genellikle basit ve etkili algoritmaların oluşturulmasına yardımcı olur. Bundan dolayı, algoritma tasarımcıları için etkili bir araçtır. Böl ve fethet algoritmalarının yapısı yinelemelidir. Bölünmüş alt problemler genellikle asıl problemin boyutundan daha küçüktür, bu

nedenle asıl problem, alt problemleri çözmek için yinelemeli olarak kendisini çağırır. Böl ve fethet yönteminin her yinelemeli düzeyde üç adımı vardır;

- Problemi benzer alt problemlere böler,
- Çözmek için özyinelemeli yöntemi kullanarak bu alt problemleri çözer,
- Alt problemlerin çözümlerini, asıl problemin çözümünü elde etmek için birleştirir.

Böl ve fethet paralel algoritmalarının paralel olarak tasarımı özellikle büyük ve karmaşık problemlerin etkili bir şekilde çözülmesinde önemli bir rol oynar. Paralel böl ve fethet, özellikle büyük veri setleri veya karmaşık problemlerin paralel işleme açık alt problemlere ayrılabildiği durumlar için uygundur. Ancak, bu stratejinin etkin bir şekilde uygulanabilmesi için paralel işleme uygun bir altyapının (paralel bilgisayarlar, çok çekirdekli işlemciler vb.) olması gerekmektedir.

Böl ve fethet stratejisinde ilk adımda oluşturulan alt problemler genellikle bağımsız olduğundan paralel olarak çözülebilirler. Alt problemler yinelemeli olarak çözülür, bu da bir sonraki bölme adımında daha fazla alt problemin paralel olarak çözülmesine olanak sağlar. Ancak şunu da belirtmek gerekir ki, tamamen paralel bir algoritma elde etmek için böl ve fethet yönteminin bölme ve birleştirme adımlarının paralelleştirilmesi gerekir. Paralel algoritmalarda orijinal problemi mümkün olduğu kadar çok alt probleme bölerek paralel olarak çözmek oldukça yaygındır. Örnek olarak paralel böl ve fethet yöntemini kullanan birleştirme sıralaması (merge sort) algoritması, n öğeden oluşan bir diziyi ortandan ikiye böler, her biri yarıyı paralel olarak sıralar ve sıralanmış alt problemleri en son birleştirir.

Böl ve fethet algoritması farklı çizge yapılarında uygulanabilir. Hiperküp ara bağlantı çizgelerinde, üç köşegen (tridiagonal) sistemleri çözmek için bir böl ve fethet yöntemi uygulanmıştır [69] ve The Parallel Cyclic Reduction ile karşılaştırıldığında daha iyi bir performansa sahip olduğu görülmüştür. [70]'te hiperküp üzerinde böl ve fethet algoritmalarının alt adımları için yükü dengeli bir şekilde dağıtan genel bir alt küp tahsis (subcube allocation) uygulaması verilmiştir ve bu uygulamanın çeşitli uyarlamaları kelebek ağlarında ve genel hiperkübik ağlarda çalışabilmektedir. [71]'de

hiperküp yapısına sahip paralel bilgisayarların verimli bir şekilde çalıştırılması için rastgele paralel programların görevlerini kümelendiren bir buluşsal algoritma tanımlanmıştır ve algoritmanın en kötü çalışma süresi $O(dn^3)$ olup, burada n paralel programdaki görev sayısı ve d hiperküpün boyutudur. [72]'te 3 boyutlu örgü ve torus ara bağlantı ağına binom ağacı yerleştirilmiş ve bu yapı için böl ve fethet yönlendirme algoritması önerilmiştir. Tüm gömme parametreleri aynı olduğundan, bir binom ağacının 3B ağ veya torus üzerine gömülmesinin sonuçlarında hiçbir fark yoktur, bu nedenle zaman sonuçları da aynıdır. 2B örgü ağlarda haritalama için basit ve optimal bir yaklaşım [73]'de verilmiştir. Hem Refleeting hem de Growing haritalamalarından daha basit olan yaklaşım, solucan deliği ağları için en uygunudur ve depola ve ilet ağları için Growing haritalamasından daha iyi sonuç vermiştir. Çip üzerinde 2B örgü ağı için kısmen uyarlanabilir ve deterministik yönlendirme algoritması [29]'de verilmiştir ve PAAD, simüle edilmiş beş uygulama için XY, WF ve DYAD'den daha iyi performans göstermiştir. Hibrit GPU-CPU ağları için diğer haritalama algoritması olan HyPar [74]'de önerilmiştir. Hibrit CPU-GPU sistemlerinde grafik uygulamalarını işlemek için bir böl ve fethet modeli, verilen grafiği cihazlar arasında böler ve her iki cihazda da eşzamanlı bağımsız hesaplamalar gerçekleştirir. Çizgelerin çok cihazlı yürütülmesi için yaygın Toplu Senkron İşleme (BSP) yaklaşımı ile karşılaştırıldığında, HyPar yöntemi ortalama %74-%92 performans artışı sağlamıştır.

2.7. DİNAMİK PROGRAMLAMA

Dinamik programlama, problemi benzer alt problemlere bölen ve bunları yinelemeli olarak çözen, böl ve fethet yöntemiyle hemen hemen aynı olan bir algoritma tasarım yöntemidir [68]. Dinamik programlama genellikle en uygun çözüme ulaşmak için bir dizi seçimin yapılması gereken optimizasyon problemlerinin çoğuna uygulanmaktadır. Dinamik programlamada, orijinal problemin bölümlenmiş alt problemlerinin tekrarlandığı ve bağımsız olmadığı durumlarda kullanılır. Problemin çözümünde seçim yaparken sıklıkla aynı alt problemler ortaya çıkmaktadır, aynı olan alt problemlerden yalnızca bir tanesi çözülür ve ardından problemin çözümü bir tabloya kaydedilir. Alt problemlerin çözümünden tasarruf sağladığı için aynı işlemleri yeniden hesaplama ihtiyacını ortadan kaldırarak kod maliyetini düşürür.

Dinamik programlamanın yapısı özyinelemeli olup, yöntem ana problemin alt problemlerini özyinelemeli olarak çağırır ve alt problemi sadece problemle ilk kez karşılaşıldığında çözer. Dinamik programlamanın geliştirme sürecinde dört adım vardır [68];

- İdeal çözümün yapısı tanımlanır
- İdeal çözüm özyinelemeli olarak belirlenir
- İdeal çözüm aşağıdan yukarıya yaklaşımla hesaplanır
- Hesaplanan verileri kullanılarak mümkün olan en iyi çözüm bulunur

İlk üç adım dinamik programlamanın temel adımlarıdır. Eğer sadece optimal çözümün değerine ihtiyaç varsa sadece ilk üç adımı kullanmak yeterli olacaktır, son adım atlanabilir. Dinamik programlama bilgisayar bilimi, matematik, ekonomi ve biyo-enformatik gibi alanlarda kullanılmaktadır. Ara bağlantı ağlarında yerleşim optimizasyon problemi [75], örgü ağda bağlantı planlama [75] ve yönlendirme [76], çapraz küpte (crossed cube) çok boyutlu torus simülasyonu [45], doğrusal tamamlayıcı iletişim için optimal işlemci haritalaması [77], matris zincir çarpımı [78] ve hiperküpte sırt çantası problemi [79] dinamik programlama yöntemi kullanılarak çözülmüştür.

BÖLÜM 3

BAĞLANTILI KARE AĞ ÇİZGELERİ (CONNECTED SQUARE NETWORK GRAPHS CSNG)

Selçuk [2] tarafından daha önce ortaya konulmuş olan Bağlantılı Kare Ağ Çizgeleri (CSNG) bu bölümde yeniden ele alınmıştır. CSNG, 2 boyutlu bir örgü yapısıdır ve aynı zamanda bir hiperküp çeşidi olmasıdır. Bu nedenle bu tezin odak noktası, CSNG için çeşitli problemlere hiperküp yardımıyla çözüm bulan algoritmalar geliştirmektir. Öncelikle dinamik programlama yardımıyla yeni bir Hamiltonian yol algoritması geliştirilmiştir. Ayrıca çizgedeki etiketlerin haritalanmasını ve tek noktaya yayın yönlendirmesini gerçekleştiren iki farklı algoritma ortaya konmuştur. Bunun yanında haritalama ve tek noktaya yayın yönlendirme için paralelleştirmenin nasıl yapılacağı incelenmiştir. Son olarak, hiperküp için yayın algoritmalarının tanımı kullanılarak CSNG için benzer yapıda olan yayın algoritmaları tanımlanmıştır. Verilen algoritmaların çalışma mantığı şekiller üzerinde gösterilmiş ve örnekler ile açıklanmıştır.

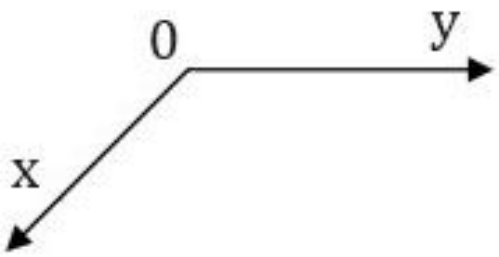
3.1. ÇİZGENİN OLUŞTURULMASI VE ANALİTİK ÖZELLİKLER

CSNG hiperküp benzeri yapısı olan bir çizgedir. *u* ve *v*, G çizgesindeki köşe noktalarını ifade eder. G çizgesi, düğüm seti V(G) ve kenar seti E(G) olan CSNG olarak kabul edilir, *u* düğümünün derecesi deg(*u*) ile ifade edilir ve $u \in V(G)$. "||" sembolü iki dizinin bitiştirilmesini temsil eder. Hamming mesafesi aşağıda verilen denkleme göre hesaplanır:

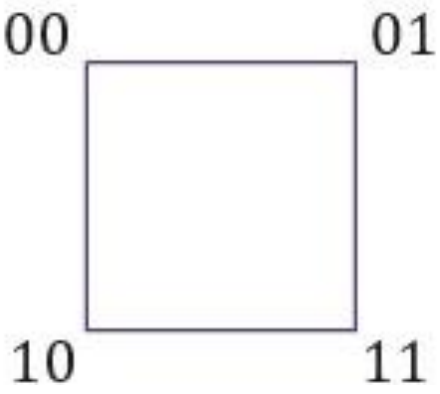
$$\sum_{i=0}^{n-1} (a_i \oplus b_i) \tag{3.1}$$

Hamming mesafesi hesaplanırken kullanılan "⊕" sembolü bit-bazlı XOR işlemidir ve iki düğüm arasındaki mesafeyi bitler arasındaki değişiklik sayısı olarak hesaplamada kullanılır. Çizgenin arka arkaya gelen düğümlerinin arasında bir bitlik fark vardır. Bu özellik çizgenin hiperküp özelliği taşıdığının da göstergesidir.

Şekil 3.1 (b)'de gösterildiği gibi S(2) 2B uzayda bir kare ile gösterilir. 2B koordinat sistemi Şekil 3.1 (a)'da verildiği gibidir.



(a)



(b)

Şekil 3.1. a) 2 boyutlu koordinat uzayı [2], b) S(2).

CSNG çizgesi oluşturulurken, oluşturulacak olan çizginin bir alt boyuttaki çizgesi, çizgenin oluşturulacağı yöne göre, x eksenine veya y eksenine göre yansıtılmış biçimleri kullanılmaktadır. CSNG(k, m) çizgesinin x ekseninde yansıtılmış biçimi CSNG′(k, m) çizgesi, y ekseninde yansıtılmış biçimi ise CSNG′′(k, m) çizgesi ile ifade edilecektir.

Tanım 3.1. CSNG(0, 0) = S(2) olsun. CSNG(*k, m*) iki adımda tanımlanabilir [2].

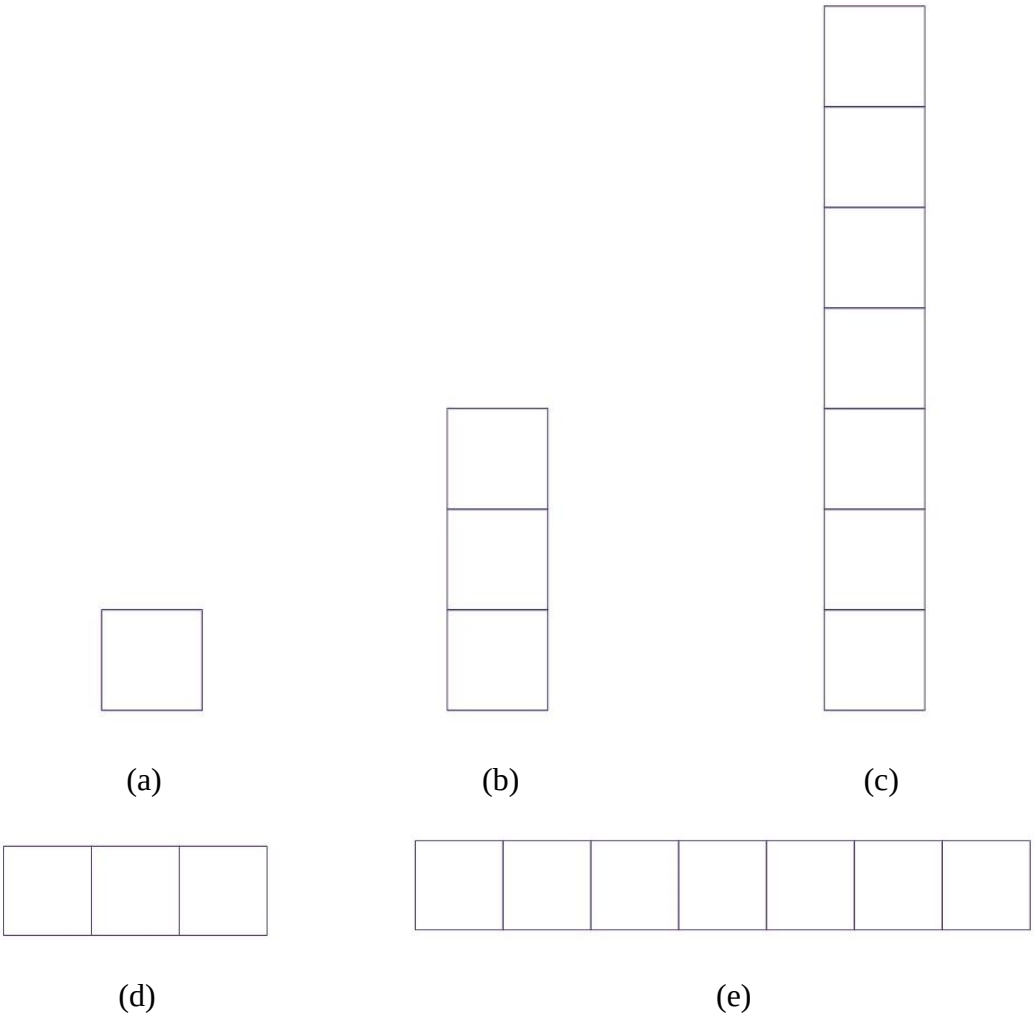
Durum I. CSNG çizgesinin tek yönde oluşturulması

- $\sum_{i=0}^{k} 2^i$ ortak iki düğüme (bir kenar) sahip karelerin *y* ekseni boyunca sağ ve/veya sol kenarlarından birbirine bağlandığını varsayalım. Verilen bu çizge CSNG(*k,* 0) olarak adlandırılacaktır. Örneğin Şekil 3.2'de bulunan (d) ve (e) ağ yapıları sırasıyla CSNG(1, 0) ve CSNG(2, 0)'dir.
- Ortak iki düğümü (bir kenarı) olan $\sum_{i=0}^{m} 2^i$ karelerinin *x* ekseni boyunca alt ve/veya üst kenarlarından birbirine bağlı olduğunu varsayalım. Bu çizgeye CSNG(0, *m*) adı verilir. Örneğin Şekil 3.2 (b) ve (c)'de verilen ağların yapısı CSNG(0, 1) ve CSNG(0, 2)'dir.

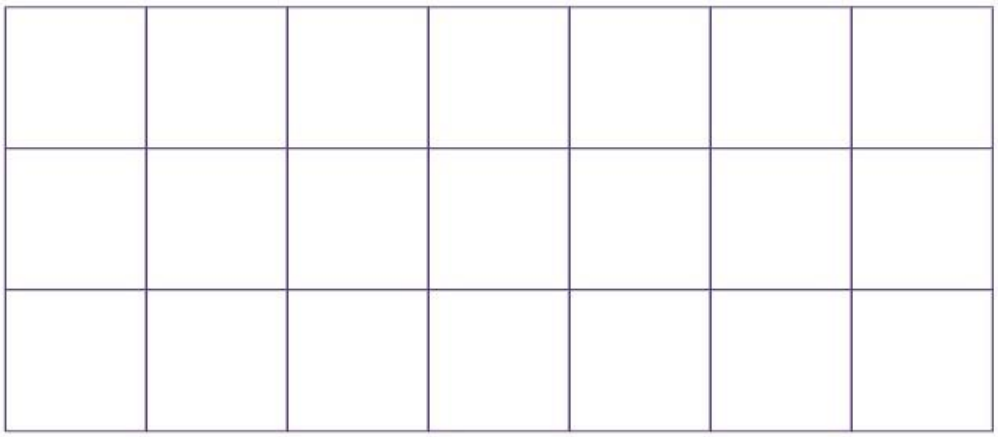
Durum II. İki yönde oluşturma

- Ortak bir kenara (sağ ve sol kenarlar) sahip $\sum_{i=0}^{k} 2^i$ CSNG(*k,* 0) çizgesinin *y* ekseni doğrultusunda bağlandığını varsayalım. Bu çizgeye CSNG(*k, m*) adı verilir.
- Ortak bir kenara (alt ve üst kenarlar) sahip $\sum_{i=0}^{k} 2^i$ CSNG(0, *m*) çizgesinin *x* eksenindeki doğrultusunda bağlandığını varsayalım. Bu çizgeye CSNG(*k, m*) adı verilir.

Örneğin, Şekil 2'de verilen ağ yapısı CSNG(1, 2)'dir. Bu çizge, *k* değeri 1'e ve *m* değeri 2'ye eşit olan bir CSNG(*k, m*) çizgesidir.



Şekil 3.2. Sırasıyla a.CSNG(0, 0) [2], b.CSNG(0, 1), c.CSNG(0, 2) d.CSNG(1, 0) [2], e.CSNG(2, 0) [2].



Şekil 3.3. CSNG(1,2) [2].

Tanım 2.1.2. İki tane CSNG(*k*, *m*-1) veya CSNG(*k*-1, *m*) boyutundaki çizgeler, boyutu iki katına çıkacak şekilde yeni bir ağ oluşturmak için birleştirilebilir. Bu yeni ağ $k \geq 0$, $m \geq 0$ olduğu durumda CSNG(*k*, *m*) = G(V, E) olarak belirtilir. Oluşacak ağ için iki durum vardır [2]:

Durum I.

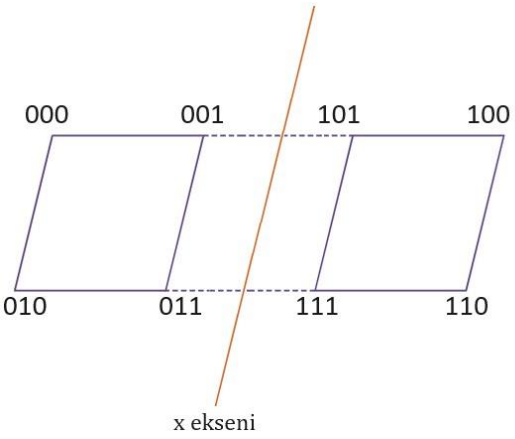
- Çizge oluşturulurken boyutu x ekseninde iki katına çıkacaksa, o zaman $0 \,||\, \mathrm{CSNG}(k-1, m)$ ve $1 \,||\, \mathrm{CSNG}''(k-1, m)$ çizgelerindeki düğüm ve kenarlar CSNG(*k*, *m*), yani $G(V_x, E_x)$ çizgesine dahil edilir.
- Eğer $\forall v_i \in V, p = 0, \ldots, k+m-1, 2^p \leq Label(v_i) \leq 2^p + 1, |k-m| \leq 1$ ise, o zaman $\forall(0 \,||\, v_i, 1 \,||\, v_i) \in E_x$.

Şekil 3.4 (b)'de CSNG(0, 0) ve CSNG(0, 0) çizgesinin *y* ekseninde yansıtılmış hali olan CSNG″(0, 0) çizgelerinin birleştirilmesiyle CSNG(1, 0) çizgesi elde edilmiştir. Aynı şekilde Şekil 3.6'da CSNG(0, 2) ve CSNG(0, 2) çizgesinin *y* ekseninde yansıtılmış hali olan CSNG″(0, 2) çizgelerinin birleştirilmesiyle CSNG(1, 2) çizgesi elde edilmiştir.

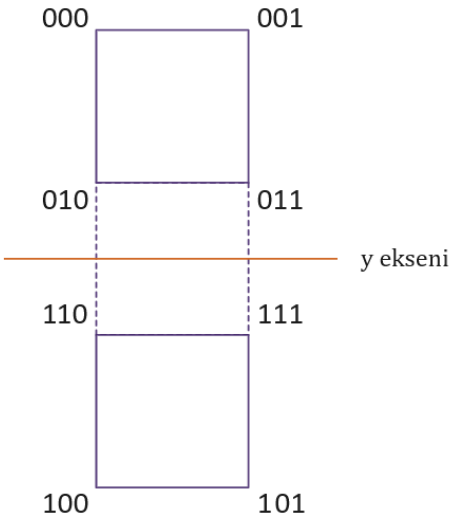
Durum II.

- Çizge oluşturulurken boyutu y ekseninde iki katına çıkacaksa, o zaman $0 \,||$ CSNG(*k*, *m*-1) ve $1||$ CSNG′(*k*, *m*-1) çizgelerindeki düğüm ve kenarlar CSNG(*k*, *m*)= $G(V_y, E_y)$ çizgesine dahil edilir.
- Eğer $\forall v_i \in V, Label(v_i)$ çift, $Label(v_i) < 2^{k+m}$, $|k-m| \leq 1$ ise, o zaman $\forall(0 \,||\, v_i, 1 \,||\, v_i) \in E_y$.

Şekil 3.4 (a)'da CSNG(0, 0) ve CSNG(0, 0)'ın *x* ekseninde yansıtılmış hali olan CSNG′(0, 0) çizgelerinin birleştirilmesiyle CSNG(0, 1) çizgesi elde edilmiştir. Aynı şekilde Şekil 3.6'da CSNG(0, 1) ve CSNG(0, 1)'in *y* ekseninde yansıtılmış hali olan CSNG″(0, 1) çizgelerinin birleştirilmesiyle CSNG(0,2) çizgesi elde edilmiştir.

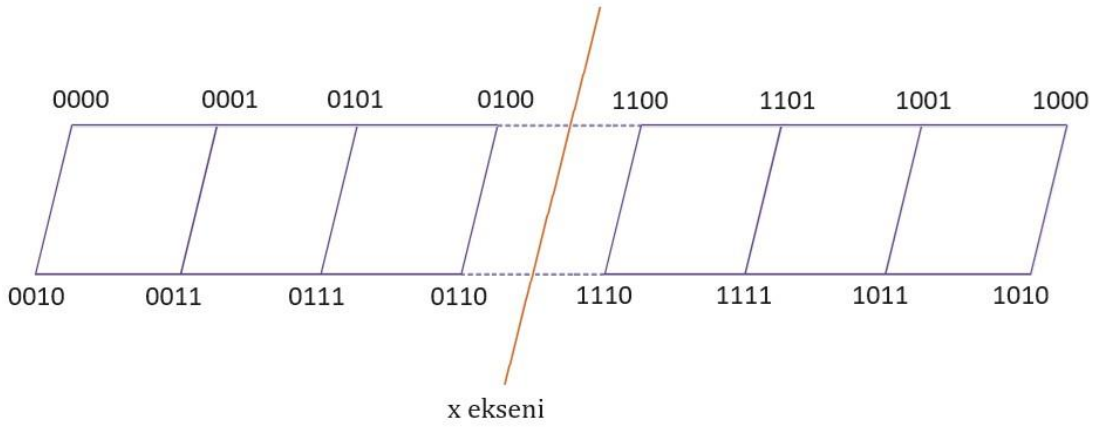


(a)

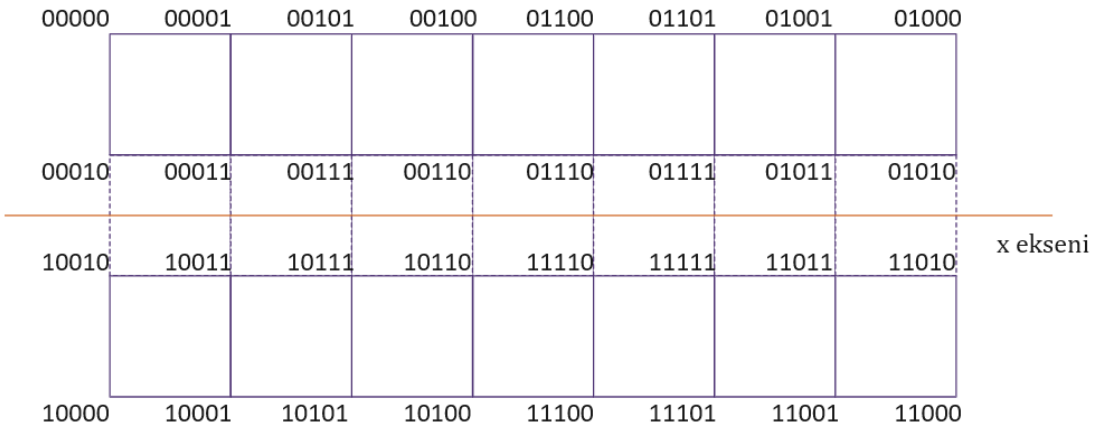


(b)

Şekil 3.4. Sırasıyla a) CSNG(0,1) Çizgesinin oluşturulması: Yansıtma düzlemi x düzlemidir. b) CSNG(1,0) Çizgesinin oluşturulması: Yansıtma düzlemi y düzlemidir [2].



Şekil 3.5. CSNG(0,2) Çizgesinin oluşturulması: Yansıtma düzlemi x düzlemidir [2].



Şekil 3.6. CSNG(1,2) Çizgesinin oluşturulması: Yansıtma düzlemi y düzlemidir [2].

3.2. HAMİLTON YOLU ALGORİTMASI

Bu bölümde, CSNG için yeni bir Hamilton yolu algoritması tanıtılacaktır. Dinamik programlama yöntemi, böl ve fethet yöntemi gibi özyinelemeli yapılar algoritmada kullanılmıştır. Dinamik programlamada asıl problem alt problemlere ayırılıp, tekrar eden alt problemler sadece bir kez hesaplanır. Bu hesaplama sonucunda elde edilen değer tabloda saklandığı ve tekrar eden problemler yeniden hesaplanmadığı için dinamik programlama, böl ve fethet yönteminden daha iyi performans sunabilmektedir. Algoritma 3.1'de dinamik programlama yöntemi kullanılmıştır. Bu algoritmada gri kod yardımıyla komşu düğümlerin etiketleri arasında 1 bitlik değişim sağlanmaktadır. Böylece çizge üzerinde bir düğümden diğerine giderken etiketlemede sadece 1 bit değiştirmek yeterlidir. İki etiket arasındaki Hamming mesafesi XOR işlemi ile bulunur ve fark 1'dir.

Örnek 3.1

İlk olarak, 2B uzayda bulunan bir kare için aşağıdaki gibi bir Hamilton yolu olduğu varsayılır;

$$00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \qquad (3.2)$$

İlk düğümün CSNG(3, 0) (veya CSNG(2, 1), CSNG(1, 2), CSNG(0, 3)) üzerinde *00010* etiketine sahip bir düğüm olduğu varsayıldığında, dış yinelemeli yapı için etiketin üst üç biti 000 ve iç yapı için etiketin alt iki biti 10 başlangıç düğümü olarak seçilir. Eşitlik 3.2'de ele alınan Hamilton yolu, başlangıç düğümü 10 olacak şekilde saat yönünde kaydırılarak en iç yapı için Hamilton yolu elde edilir. CSNG(0, 0) yani kare için Hamilton yolu H(0) ve H(0)'ın tersi I(0) olarak adlandırılacaktır.

Temel durumda olan çizge için Hamilton yolu H(0) ve yolun tersi I(0) aşağıdaki şekilde sırasıyla Eşitlik 3.3 ve Eşitlik 3.4'te verilmiştir.

$$H(0) = 10 \rightarrow 00 \rightarrow 01 \rightarrow 11 \tag{3.3}$$

$$I(0) = 11 \rightarrow 01 \rightarrow 00 \rightarrow 10 \tag{3.4}$$

1. özyineleme ile H(1) ve tersi I(1) aşağıdaki gibi elde edilebilir.

$$\begin{aligned} H(1) &= 0||H(0) \rightarrow 1||I(0) \\ &= 010 \rightarrow 000 \rightarrow 001 \rightarrow 011 \rightarrow 111 \rightarrow 101 \rightarrow 100 \rightarrow 110 \end{aligned} \tag{3.5}$$

$$\begin{aligned} I(1) &= 1||H(0) \rightarrow 0||I(0) \\ &= 110 \rightarrow 100 \rightarrow 101 \rightarrow 111 \rightarrow 011 \rightarrow 001 \rightarrow 000 \rightarrow 010 \end{aligned} \tag{3.6}$$

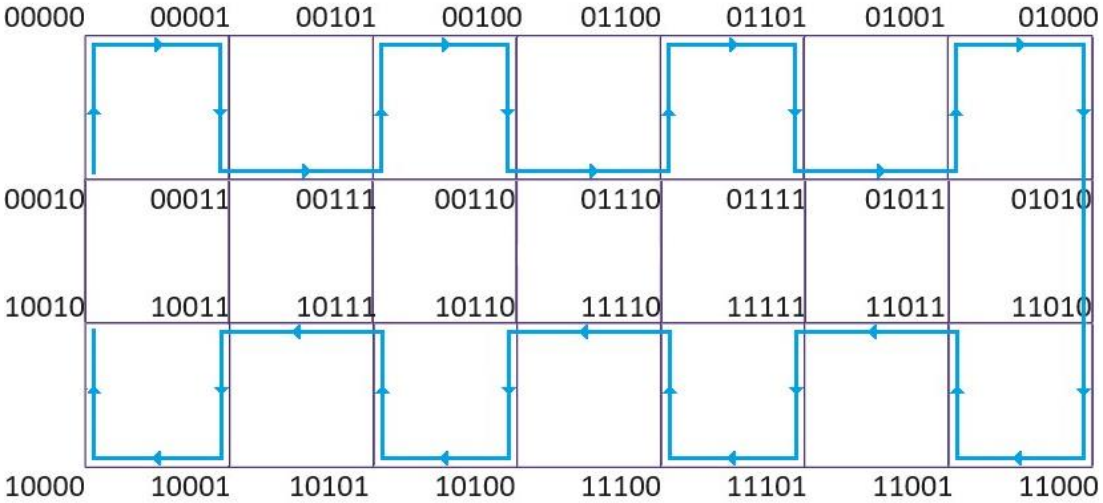
2. özyineleme ile H(2) ve tersi I(2)aşağıdaki gibi elde edilebilir.

$$\begin{aligned} H(2) &= 0||H(1) \rightarrow 1||I(1) \\ &= 0010 \rightarrow 0000 \rightarrow 0001 \rightarrow 0011 \rightarrow 0111 \rightarrow 0101 \rightarrow \\ &\quad 0100 \rightarrow 0110 \rightarrow 1110 \rightarrow 1100 \rightarrow 1101 \rightarrow 1111 \rightarrow \\ &\quad 1011 \rightarrow 1001 \rightarrow 1000 \rightarrow 1010 \end{aligned} \tag{3.7}$$

$$\begin{aligned} I(2) &= 1||H(1) \rightarrow 0||I(1) \\ &= 1010 \rightarrow 1000 \rightarrow 1001 \rightarrow 1011 \rightarrow 1111 \rightarrow 1101 \rightarrow \\ &\quad 1100 \rightarrow 1110 \rightarrow 0110 \rightarrow 0100 \rightarrow 0101 \rightarrow 0111 \rightarrow \\ &\quad 0011 \rightarrow 0001 \rightarrow 0000 \rightarrow 0010 \end{aligned} \tag{3.8}$$

3. özyineleme ile H(3) aşağıdaki gibi elde edilebilir.

$$
\begin{aligned}
H(3) \;=\; & 0||H(2) \;\rightarrow\; 1||I(2) \\
= \; & 00010 \rightarrow 00000 \rightarrow 00001 \rightarrow 00011 \rightarrow 00111 \rightarrow 00101 \rightarrow \\
& 00100 \rightarrow 00110 \rightarrow 01110 \rightarrow 01100 \rightarrow 01101 \rightarrow 01111 \rightarrow \\
& 01011 \rightarrow 01001 \rightarrow 01000 \rightarrow 01010 \rightarrow 11010 \rightarrow 11000 \rightarrow \\
& 11001 \rightarrow 11011 \rightarrow 11111 \rightarrow 11101 \rightarrow 11100 \rightarrow 11110 \rightarrow \\
& 10110 \rightarrow 10100 \rightarrow 10101 \rightarrow 10111 \rightarrow 10011 \rightarrow 10001 \rightarrow \\
& 10000 \rightarrow 10010
\end{aligned}
\tag{3.9}
$$



Şekil 3.7. Örnek 3.1'de bulunan Hamilton yolunun CSNG(1,2) üzerinde gösterimi.

Açıklama 3.1

Algoritma 3.1 dinamik programlama kullanarak CSNG(*k, m*) için Hamilton yolunu hesaplamaktadır. Eğer bu algoritma böl ve fethet yaklaşımı ile tasarlanmış olsaydı, algoritmada iç içe geçmiş iki özyinelemeli yapı ile karşılaşılması gerekecekti. Böl ve fethet yöntemi yerine dinamik programlama kullanılarak Hamilton yolu daha basit bir strateji ile bulunur.

Adım 1 herhangi bir döngü, özyineleme veya diğer fonksiyonları içermez, sadece sabit sayıda işlem içerir, bu nedenle zaman karmaşıklığı O(1)'dir. Adım 2 döngü içerir ve döngü değişkeni *j*, 1'den başlayıp *k* + *m*'ye kadar her iterasyonda bir artar. Döngü içinde "||" işlemi bulunur, yani her iterasyon esasen gizli bir for döngüsü içerir. Her *j* için "||" işleminde kullanılacak dizilerin boyutlarının toplamı Algoritma

3.1'in karmaşıklığını verir. $j = 1$ için, dizinin boyutu: 2^2 , $j = 2$ için, dizinin boyutu: $2^3, \ldots, j = k + m$ için, dizinin boyutu: 2^{k+m+1}. Toplam çalışma zamanı; 2^2 + 2^3 + ... + 2^{k+m+1} = $2^{k+m+2} - 4$ şeklinde geometrik seri açılımı kullanılarak hesaplanır. Böylece, Algoritma 3.1'in zaman karmaşıklığı O(2^{k+m+2}) olarak hesaplanır. 2^{k+m+2} aynı zamanda çizgedeki toplam düğüm sayısına eşittir.

CSNG(k, m) 2^{k+m+2} düğüme sahiptir. CSNG(0, 0) kareye eşdeğerdir ve 2^2 adet düğüme sahiptir. C(2, 0), C(0, 2) ve C(1, 1) aynı sayıda düğüme sahip olmalarına rağmen izomorfik çizgeler değillerdir. Bununla birlikte, CSNG(k, m) çizgesinin inşası nedeniyle izomorfik olmayan bu çizgeler için bir Hamilton yolu, Algoritma 3.1 kullanılarak benzer bir şekilde elde edilir.

S(2)'de 8 Hamilton yolu ve 4 farklı başlangıç düğümü bulunmaktadır. Bu yollar aşağıdaki gibi belirtilmiştir;

00 → 01 → 11 → 10 (3.10)

00 → 10 → 11 → 01 (3.11)

01 → 11 → 10 → 00 (3.12)

01 → 00 → 10 → 11 (3.13)

11 → 10 → 00 → 01 (3.14)

11 → 01 → 00 → 10 (3.15)

10 → 00 → 01 → 11 (3.16)

10 → 11 → 01 → 00 (3.17)

Algoritma 3.1. Bu algoritma dinamik programlama sürecini kullanarak CSNG(*k, m*) ile alternatif bir Hamilton yolu hesaplar.

```
Data: k, m, N: başlangıç düğümü ve case: 1 veya 2
Result: H: CSNG(k,m)çizgesinin Hamilton yolu
Function HamiltonianPath(k, m, N, case)
      // Adım 1: Kare için alternatif Hamilton yolu
      if k + m == 0 then
            if case == 1 then
                  H(0) = N⊕00 → N⊕01 → N⊕11 → N⊕10
                  I(0) = N⊕10 → N⊕11 → N⊕01 → N⊕00
            end
            if case == 2 then
                  H(0) = N⊕00 → N⊕10 → N⊕11 → N⊕01
                  I(0) = N⊕01 → N⊕11 → N⊕10 → N⊕00
            end
      end
      // Adım 2: CSNG(k, m) için Hamilton yolu
      for j = 1 to k+ m do
            if N(j+2) == 0 then
                  H(j) = 0 || H(j-1) → 1 || I(j-1)
                  I(j) = 1 || H(j-1) → 0 || I (j-1)
            end
            if N (j+2) == 1 then
                  H(j) = 1 || H(j-1) → 0 || I(j-1)
                  I(j) = 0 || H(j-1) → 1 || I (j-1)
            end
      end
      return H
end function
```

Bu yollar için genel bir formül, Algoritma 3.1'deki 1. adım kullanılarak aşağıda verilmiştir;

Durum 1

$$\text{H(0)} : N \oplus 00 \rightarrow N \oplus 01 \rightarrow N \oplus 11 \rightarrow N \oplus 10$$

Durum 2

$$\text{H(0)} : N \oplus 00 \rightarrow N \oplus 10 \rightarrow N \oplus 11 \rightarrow N \oplus 01$$

Hiperküp stratejisine benzer şekilde yeni çizgeler türetmek için, bulunan yolların ters sıralamasına da ihtiyaç vardır. Bunun için Algoritma 3.1'deki 1. adımda ters yollar aşağıdaki gibi belirtilir;

Durum 1

$$\text{I(0)} : N \oplus 10 \rightarrow N \oplus 11 \rightarrow N \oplus 01 \rightarrow N \oplus 00$$

Durum 2

$$\text{I(0)} : N \oplus 01 \rightarrow N \oplus 11 \rightarrow N \oplus 10 \rightarrow N \oplus 00$$

Algoritma 3.1'in 2. adımında, bir önceki alt problemde elde edilen yol ve bu yolun tersi Tanım 3.1 kullanılarak birleştirilir.

CSNG(k, m) çizgesinin ağ yapısı 2B örgü ağ yapısı ile benzerlik göstermektedir. Literatürde yapılan çalışmalarda 2B örgü ağlarda ($M(p,r)$) [80][81][82] pr büyüklüğünde Hamilton yolu için algoritmalar genellikle O(pr) zaman karmaşıklığına sahiptir. Burada, $p = 2^{k+1}$ ve $r = 2^{m+1}$ 'dir. Zaman karmaşıklıkları karşılaştırıldığında CCNG(k, m)'de önerilen Hamilton algoritması ile M(p,r) için Hamilton algoritmaları ile aynı karmaşıklığa sahip olduğu görülmektedir.

3.3. TEK YÖNE YAYIN YÖNLENDİRME

Bu bölümde, CSNG için yeni tek noktaya yayın yönlendirme algoritmaları tanıtılmaktadır. CSNG üzerinde haritalama için verilen algoritma her düğüm için etiketlerin oluşturulmasını sağlayan özyinelemeli bir algoritmadır. Yol bulmak için

verilen özyinelemeli algoritma etiketleme algoritmasını kullanarak elde edilen 2 boyutlu dizi üzerinde en kısa yolu bulmaktadır. Bu algoritmaların daha iyi anlaşılması için örnekler ve şekiller detaylı bir şekilde verilmiştir ve algoritmaların çalışma zamanları hesaplanmıştır.

3.3.1. Haritalama Algoritması

CSNG iletişim ağında her düğüm için etiketlerin belirlenmesi ve 2 boyutlu düzlemde bu etiketlerin konumlandırılması için haritalama algoritması geliştirildi. Haritalama algoritmasında kullanılan mantık çizgenin oluşturulma mantığı ile aynıdır. Verilen *k* ve *m* değerlerine göre, öncelikle *x* düzleminde boyut her özyinelemede ikiye katlanarak CSNG(*k, 0*) elde edilir. Çizgenin boyutu arttırılırken bir alt boyutun çizgesi ve bu çizgenin *y* düzlemine göre aynalanmış hali etiketlerinin başlarına sırasıyla 0 ve 1 biti eklenerek birleştirilir. Daha sonra elde edilen CSNG(*k, 0*) çizgesi kullanılarak *y* düzleminde aynı şekilde her özyinelemede boyut iki katına çıkartılır ve en son CSNG(*k, m-1*) ve bu çizgenin *x* düzleminde aynalanmış hali olan CSNG'(*k, m-1*) etiketlerinin başlarına sırasıyla 0 ve 1 biti eklenerek birleştirilir. CSNG(*k, m*) çizgesinin 2 boyutlu etiketleri düğümlerin koordinatlarına uygun olarak elde edilir.

Haritalama algoritmasında temelde dört durum vardır. *M* algoritma sonunda dönen 2 boyutlu dizinin adıdır. Temel durumda *M*, CSNG'nin temel durumu olan S(2)'nin haritası yani etiketleri olan [00 01; 10 11] değeridir. Diğer durumlar S(2)'nin *x, y* ve çapraz düzlemde yani hem *x* hem de *y* düzleminde yansıtılarak elde edilen haritalardır.

Algoritmanın girdi olarak aldığı diğer değerler *i* ve *j* değerleridir. Bu değerler yansıtılma düzlemine göre 1 veya 0 değerini alırlar. Eğer değişkenlerin ikisi de 0 değerini almışlarsa bu alt problemin aslını almaktadır. Eğer *i* değişkeninin değeri 0 ve *j* değişkeninin değeri 1 ise CSNG(0, 0) *x* eksenine göre, *i* değişkeninin değeri 1 ve *j* değişkeninin değeri 0 ise CSNG(0, 0) *y* eksenine göre, eğer her iki değişken de 1 değerini almışsa o zaman CSNG(0, 0)'nin hem *x* hem de *y* eksenine göre aynalanmış hali alınacaktır. Temel durumlar aşağıda sırasıyla verilmiştir:

- M = [00 01; 10 11]
- M = [01 00; 11 10]
- M = [10 11; 00 01]
- M = [11 10; 01 00]

Algoritma 3.2. Bu algoritma, böl ve fethet yöntemini kullanarak CSNG(*k*, *m*) düğümlerinin etiketlenmesini yapmaktadır.

```
Data: k, m
Result: CSNG(k, m) çizgesinin etiketlemesi (haritası)
Function Map (k, m, i, j):
      if k>0 then
            M(1 : 2^k,:) = j || Map(k - 1, m, 0, j)
            M(2^k+1: 2^(k+1),:) = !j || Map(k-1, m, 0, !j)
            return M
      end
      if m > 0 then
            M(:,1 : 2^m) = i || Map(k, m -1,0, j)
            M(:,2^m + 1 : 2^(m+1)) = !i || Map(k, m – 1,1,j)
            return M
      end
      if i == 0 and j == 0 then
            return M =[00 01; 10 11]
      end
      if i == 1 and j == 0 then
            return M =[01 00; 11 10]
      end
      if i == 0 and j == 1 then
            return M =[10 11; 00 01]
      end
      if i == 1 and j == 1 then
            return M =[11 10; 01 00]
      end
end function
```

Örnek 3.2.

CSNG(2, 2) çizgesi için haritayı bulmak yinelemeli bir süreçtir. Algoritma 3.2 için i ve j değerlerine göre 4 farklı temel durum vardır. (i, j) için $(0, 0), (1, 0), (0, 1)$ ve $(1, 1)$ olan bu durumların algoritmanın temel durumdaki yani CSNG(0,0) için dönüş değeri olan *M* haritasını belirler.

Temel durumda M, (i, j) değerlerine göre sırasıyla $[00\ 01;\ 10\ 11], [01\ 00;\ 11\ 10]$, $[10\ 11;\ 00\ 01], [11\ 10; 01\ 00]$ değerlerini alır. *i* değişkeni, çizgenin *y* eksenine göre ayna tersini hesaplamak için kullanılırken, *j* değişken, ise *x* eksenine göre çizgenin ayna tersini hesaplamak için kullanılır.

Haritalama algoritmasında, işlev yinelemeli olarak her çağrıldığında, haritanın boyutu iki katına çıkar. Öncelikle CSNG(0, $m-1$) ve CSNG(0, $m-1$)'nin ters sıralanmış hali etiketlerin başına sırasıyla 0 ve 1 eklendikten sonra birleştirilerek CSNG(0, *m*) elde edilir ve çizge yatay olarak büyütülür. Ardından CSNG($k-1$, *m*) ve CSNG($k-1$, *m*)'nin ters sıralanmış halinin etiketlerinin başına sırasıyla 0 ve 1 eklendikten sonra elde edilenler birleştirilerek CSNG(*k*, *m*) elde edilir ve çizge dikey olarak büyütülür.

Algoritma ilk çağrıda *i* ve *j* 0 olmalıdır. Adım adım Map(2, 2, 0, 0) aşağıdaki gibi elde edilir:

CSNG(0,0):

M=[00 01 ; 10 11]

CSNG(0,1):

M=[000 001 101 100 ; 010 011 111 110]

CSNG(0,2):

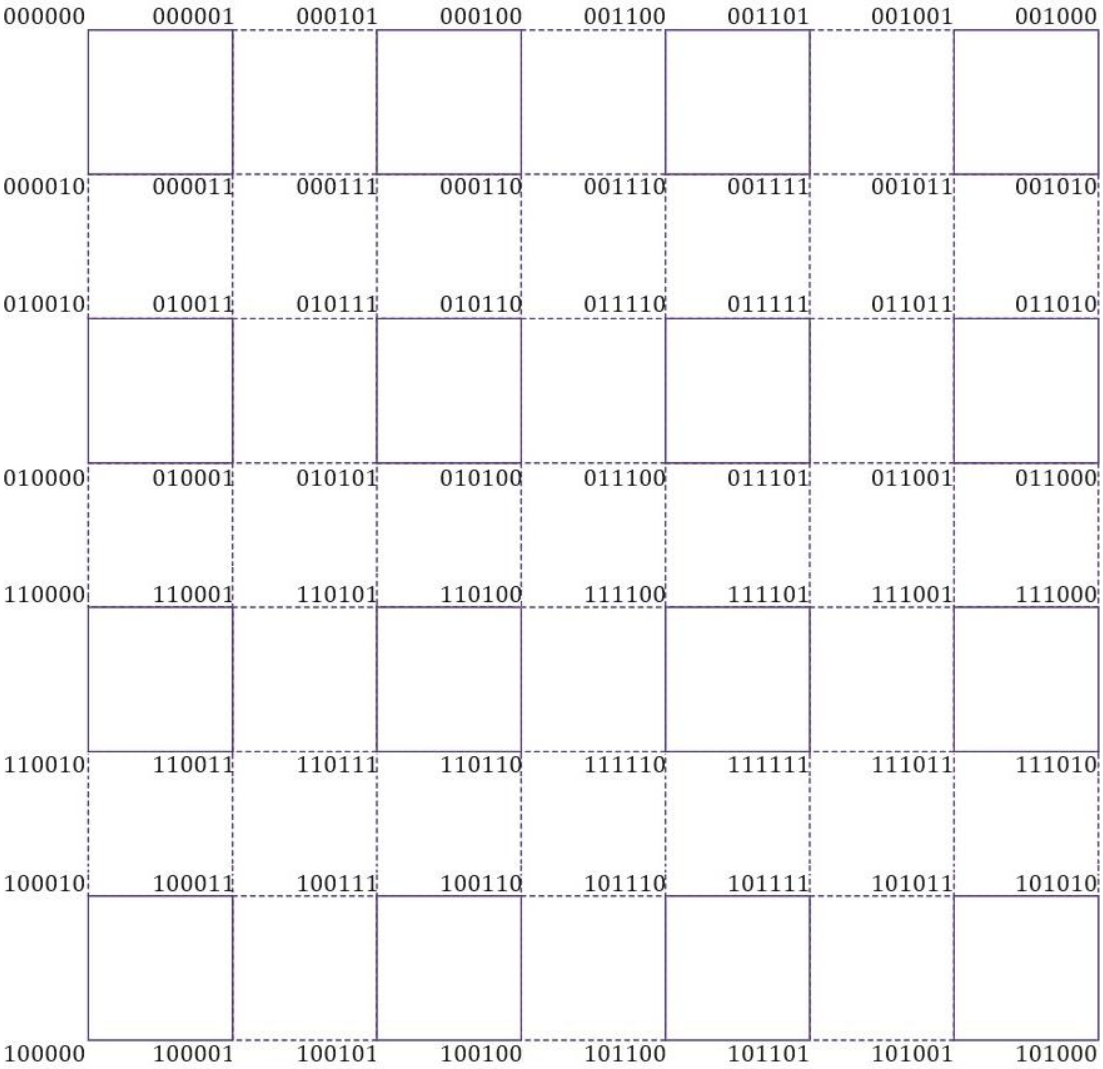
M=[0000 0001 0101 0100 1100 1101 1001 1000;
0010 0011 0111 0110 1110 1111 1011 1010]

CSNG(1,2):

M=[00000 00001 00101 00100 01100 01101 01001 01000;
00010 00011 00111 00110 01110 01111 01011 01010;
10010 10011 10111 10110 11110 11111 11011 11010;
10000 10001 10101 10100 11100 11101 11001 11000]

CSNG(2,2):

M=[000000 000001 000101 000100 001100 001101 001001 001000;
000010 000011 000111 000110 001110 001111 001011 001010;
010010 010011 010111 010110 011110 011111 011011 011010;
010000 010001 010101 010100 011100 011101 011001 011000;
110000 110001 110101 110100 111100 111101 111001 111000;
110010 110011 110111 110110 111110 111111 111011 111010;
100010 100011 100111 100110 101110 101111 101011 101010;
100000 100001 100101 100100 101100 101101 101001 101000]



Şekil 3.8. Örnek 3.2'de elde edilen CSNG(2,2)'nin etiketleri.

3.3.2. Tek Yöne Yayın Yönlendirme Algoritması

Ağlar üzerindeki düğümler diğer düğümlerle iletişim kurarken çeşitli yöntemlerle gönderilen mesajın kime ulaşacağına karar verirler. Bazı mesajlar doğrudan bir düğüme gönderilir, bazılarının belirli bir düğüm kümesine ulaşması amaçlanır, bazıları ise aynı ağdaki tüm düğümlere ulaştırılır.

Ağa gönderilecek olan mesaj doğrudan tek bir düğümü ilgilendiriyorsa ve düğümün adresi biliniyorsa, mesaj tek yöne yayın (unicast) olarak tasarlanmış bir algoritma kullanılarak gönderilir. Her yöne yayın (broadcast) algoritmaları birden-hepsine

olarak mesajı bir düğümden ağdaki tüm düğümlere ulaştırırken çoklu yayın (multicast) algoritmaları bire-çok yayın olarak ağdaki grup içi iletişim için kullanılır, yani mesaj sadece belirli bir grupta bulunan düğümlere iletilir.

Tek yöne yayın yönlendirme yöntemi en yaygın yönlendirme şeklidir. Bu yöntem, yerel alan ağları (LAN'lar) ve geniş alan ağları (WAN'lar) dahil olmak üzere çeşitli ağ ortamlarında yaygın olarak kullanılır.

CSNG(*k, m*) üzerinde tek noktaya yönlendirme algoritmasında hiperküp için E-cube yönlendirme algoritmasından farklı bir strateji izlenecektir çünkü hiperküpteki düğümler arasında bulunan bütün bağlantılar CSNG'de bulunmamaktadır. Klasik yöntem, önce kaynaktan (*S*) hedefe (*D*) kaç bitin değiştirildiğini belirlemektir. Bu tezde aşağıdaki adımlardan oluşan farklı bir strateji izlenmiştir;

Adım 1:

İlk adım olarak CSNG(*k, m*)'yi oluşturan tüm karelerin etiketlerinin hesaplanması gerekir. Algoritma 3.2 Etiketleme (Labeling) algoritması ile bu çizge için yeni etiketleme yöntemi verilmiştir. Etiketleme algoritması sonucunda elde edilen harita, tüm düğümlerin konumunu göstermektedir ve Şekil 3.8'deki gibi 2 boyutlu bir dizidir.

Adım 2:

Kaynak *S* ve hedef *D* düğümleri arasındaki tek noktaya yayın yönlendirme yolu, Algoritma 3.3 (yol bulma algoritması) kullanılarak Adım 1'de hesaplanan düğümlerin yeni etiketlerinin yardımıyla ilgili karelerden önce satırları ve ardından sütunları tarayarak elde edilir.

Algoritma 3.3. Bu algoritma CSNG(k, m) için tek yöne yayın yönlendirmeyi hesaplar (özyinelemeli süreç).

```
Data: S = source, D = destination, M: Map, k, m, MSG
Result: MSG mesajı S'den D'ye iletilir
Function Route (S, D, M, k, m):
      for i = 0 to 2^(k+1) - 1 do
            for j = 0 to 2^(m+1) - 1 do
                  if M[i, j] == S then
                        SI = [i, j]
                  end
                  if M[i, j] == D then
                        DI = [i, j]
                  end
            end
      end
      if SI(1) != D(1) then
            for i = S(1) to D(1) do
                  P = P ∪ M(i, SI(2))
            end
      end
      if S2(1) != D(2) then
            for i = S(2) to D(2) do
                  P = P ∪ M(DI(1), i)
            end
      end
end function
for i = 0 to L - 1 do
      P[i + 1] ⟵(MSG) P[i]
end
```

Örnek 3.3.

CSNG(2, 2) etiket kümesinden seçilen iki düğüm $S: (kaynak\ düğüm) = 101111$ ve $D: (hedef\ düğüm) = 010010$ olsun. Bu çizgenin haritası, Algoritma 3.2

kullanılarak Şekil 3.8'deki gibi elde edilir. Yöntem, öncelikle harita üzerinde S ve D'nin konumunu belirler. Düğümlerin konumunu tespit etmenin yolu çizgeyi baştan sona taramaktır.

Bu örnek için, S'nin konumu $SI = \{7,6\}$ ve D'nin konumu $DI = \{3,1\}$ şeklindedir. Bu konumlardan S noktasından D'ye ulaşmak için en kısa yolun uzunluğu

$$|SI(1) - DI(1)| + |SI(2) - DI(2)| = 4 + 5 = 9$$

olacak şekilde iki konum arasındaki farkı almaktır. Bu fark düğümlerin x ve y düzlemindeki konumların farkları toplanarak elde edilir. Bu örnekte elde edilen farkın 9 olması, kaynaktan hedefe giden en kısa yolun 9 düğüm üzerinden geçeceği anlamına gelir. S ve D konumları bulunduktan sonra Algoritma 3.3 ile önce dikey sonra yatay hareket edilerek yol oluşturulur. Bu rota üzerindeki düğümlerin konumları:

Dikey yöndeki hareket esnasında üzerinden geçilen düğümler;

$$\{7,6\} \rightarrow \{6,6\} \rightarrow \{5,6\} \rightarrow \{4,6\} \rightarrow \{3,6\}$$

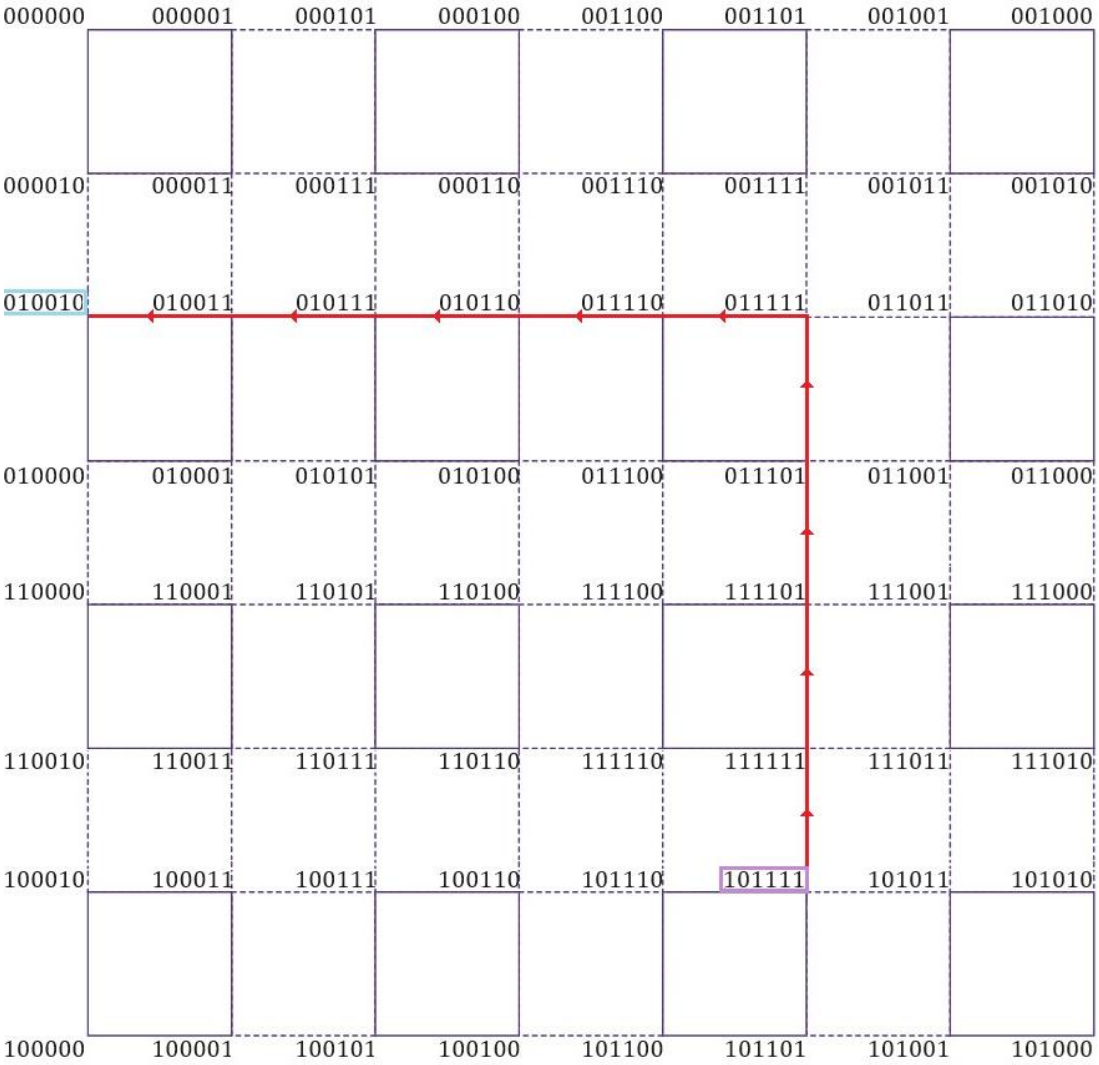
Yatay yöndeki hareket esnasında üzerinden geçilen düğümler;

$$\{3,5\} \rightarrow \{3,4\} \rightarrow \{3,3\} \rightarrow \{3,2\} \rightarrow \{3,1\}$$

şeklindedir. Burada kaynak düğümün konumu $SI = \{7,6\}$ ve hedef düğümün konumu $DI = \{3,1\}$ olarak hesaplanmıştır.

Algoritma sonucu kaynak ve hedef arasında elde edilen rota aşağıdaki şekildedir;

$$101111 \rightarrow 111111 \rightarrow 111101 \rightarrow 011101 \rightarrow 011111 \rightarrow 011110 \rightarrow 010110 \rightarrow 010111 \rightarrow 010011 \rightarrow 010010$$



Şekil 3.9. Örnek 2.3.2'de $S:(Kaynak) = 101111$ ve $D:(Hedef) = 010010$ arasında bulunan yol.

Açıklama 3.2

Algoritma 3.2 (etiketleme algoritması), CSNG(k, m) çizgesi için düğümlerin etiketlerini Şekil 3.8'deki gibi 2B düzlemde eşler. Algoritma özyinelemelidir ve [2]'deki etiketleme algoritmasıyla aynı şekilde çalışır. Bu algoritma, 2B düzlemdeki etiketlerin konumu çizgedeki köşe sayısına eşit olacak şekilde iki boyutlu bir dizi olarak döndürür. Algoritma 3.1'in zaman karmaşıklığının hesaplanmasına benzer olarak hesaplandığı zaman Algoritma 3.2'nin çalışma süresinin zaman karmaşıklığı $T(k,m) = 2k + 1 - 1 + 2m + 1 - 1$'dir. $k > m$ olduğunda çalışma süresi

$O(2^{k+1} - 1)$ ve $k < m$ olduğunda çalışma süresi $O(2^{m+1} - 1)$ olacaktır. Yani, Algoritma 3.2'nin çalışma süresinin aşağıdaki gibi iki durumu vardır;

- Eğer k ve m farklıysa: Algoritma 3.2'nin çalışma süresi $O(z)$'dir, burada

$$z = max(2^{k+1} - 1, 2^{m+1} - 1)$$

- Eğer k ve m eşitse: Algoritma 3.2'nin çalışma süresi $O(z)$dir, burada

$$z = 2^{k+1} - 2$$

Algoritma 3.3, Algoritma 3.2'nin yardımıyla CSNG(k, m) için tek noktaya yayın yönlendirmesini bulur. Algoritma 3.3'ün zaman karmaşıklığı algoritma 3.2'ye eşittir. Bunun nedeni Algoritma 3.2'nin çalışma zamanının karmaşıklık olarak Algoritma 3.3'ün adımlarından çok daha büyük olmasıdır. Algoritma 3.3'ün çalışma zamanı $O(2^{k+m} + 2)$'dir.

XY yönlendirme algoritması, 2B örgü ağlarında kullanılan dağıtılmış bir yönlendirme algoritmasıdır [83]. Bu algoritma mevcut düğüm adresini ve hedef düğüm adresini kullanarak yol üzerinde bir sonraki gidilecek olan düğümü hesaplar. Bir sonraki düğümün hesaplanması sabit zaman alır ve yol üzerindeki her düğümde yapılır. Dolayısıyla, CSNG(k, m) üzerindeki XY yönlendirme algoritmasının zaman karmaşıklığı $\Theta(2^{k+m} + 2)'$dir. Yani, Algoritma 3.3 ve XY yönlendirme algoritması aynı zaman karmaşıklığına sahiptir.

3.3.2. Algoritmaların Paralelleştirilmesi

Paralelleştirme, bir görevi veya hesaplamayı aynı anda yürütülebilecek daha küçük, bağımsız parçalara ayırma sürecini ifade eder. Paralel algoritmalar, bir hesaplamanın genel performansını ve verimliliğini artırmak için birden fazla işlemci, çekirdek veya dağıtılmış bilgi işlem kaynakları gibi paralel işleme özelliklerinden yararlanmak üzere tasarlanmıştır. Paralelleştirme özellikle büyük ölçekli sorunların ele alınması

ve mevcut kaynakların kullanımının optimize edilmesi için önemlidir. Ulaşılabilecek paralellik derecesi, algoritmanın yapısına ve mevcut donanım altyapısına bağlıdır.

Paralel programlama sayesinde daha büyük problemler daha kısa sürede çözülebilir ve performans arttırılabilir. Paralel programlama yöntemi, bilimsel gelişmelerle ortaya çıkan karmaşık ve büyük problemlerin çözümü için problemlerin farklı kısımlarını farklı işlemcilere bölerek gerekli ivme ve verimliliğin sağlanmasına yardımcı olur. Paralel programlamada bir problemin çözümü aşağıdaki adımlarla gerçekleştirilir [61];

- Eşzamanlı olarak yapılabilecek görevlerin belirlenmesi.
- Eşzamanlı olan görevlerin paralel süreçlerle eşleştirilmesi.
- Programın ara veri giriş ve çıkışlarının dağıtımı. İşlemciler tarafından paylaşılan verilerin kullanımının yönetilmesi.
- Paralel programın yürütülmesi boyunca çeşitli noktalarda işlemcilerin senkronize hale getirilmesi.

Paralel programlamanın en temel tercih sebebi, tek bir işlemci yerine birden fazla işlemci kullanarak bellekten en uygun şekilde yararlanarak bilgisayardaki yavaşlamaların önüne geçmektir. Bu sayede bilgisayarlar hızlı bir şekilde hesaplamalar yapabilmekte ve bu da bilgisayarlarda performans artışı sağlamaktadır.

CSNG(k, m) için etiketlerin eşlenmesi, verimliliği artırmak ve çalışma süresini azaltmak için paralel olarak gerçekleştirilebilir. Her yinelemede, her etiket farklı bir işlemcide hesaplanırken CSNG(k', m') alt çizgesinin bir haritası oluşturulabilir. Fonksiyon kendisini çağırdığında problemi ikiye böler ve problem boyutu bir olduğunda CSNG(0, 0) için haritayı hesaplar. Daha sonra algoritma, bu alt problemin sonucunu kullanarak bu problemi çağıran iki katı büyüklükteki bir üst problemin sonucunu hesaplar.

Algoritma 3.2'nin paralel mimaride çalışma süresi için düğümlerin adres uzunluğuna ve işlemci sayısına bağlı olarak iki farklı durum vardır. Burada CSNG(k, m) için

adres uzunluğu $k + m + 2$'dir. Algoritma 3.3 için çalışma zamanı karmaşıklığı için sadece bir durum vardır.

Durum 1.

Durum 1'de, işlemci sayısı 2^n, düğüm sayısından büyük veya ona eşittir. Algoritma 3.2 paralel düğümlerde çalıştırılırsa, bu durumda çalışma süresi $k + m$ olacaktır. Bunun nedeni, etiketler hesaplanırken i, 2'den $k + m$'ye kadar değerler alırken her seferinde 2^i etiketin hesaplanabilmesidir.

Haritayı hesaplamak için CSNG(*k, m*), CSNG(*k*−1, *m*) veya CSNG(*k, m*−1) kullanılır ve bu adım farklı işlemcilerde *O(1)* zamanında paralel olarak gerçekleştirilir. CSNG(*k, m*) çizgesinin haritası temel durum CSNG(0, 0)'dan hesaplandığında, algoritmanın çalışma zamanı $O(k + m)$ olur. $2^{k+m} \leq 2^n$ ve $n \in N$ olduğunda, Algoritma 3.2'nin çalışma süresi;

$$T(n) = k + m.$$

Algoritma 3.3 için çalışma süresi, yol boyunca mesaj gönderme süresine eşit olacaktır. Kaynak ve hedef düğümlerin harita üzerindeki yerlerinin bulunması O(1) zaman alır. Her düğümde kaynak ve hedef düğüm değeri bulunduğu düğümün etiket değeri ile karşılaştırılır ve daha sonra aynı olan değerlerin satır ve sütun numaraları düğümün kordinatları olarak döndürülür. Düğümlerin konumu için paralel algoritmanın çalışma süresini hesaplamak için yalnızca karşılaştırma süresi kullanılır. Kaynak ve hedef düğümün haritadaki konumu bilindiğinde, haritanın bir satırındaki ve bir sütunundaki köşe sayısı $2^{k+1} - 1 + 2^{m+1} - 1$ algoritmasının çalışma süresidir, yani en kötü durum senaryosunda 2B çizgede düğümden düğüme mesaj gönderme süresi $O(2^k + 2^m)$ olacaktır. $2^{k+m} \leq 2^n$ ve $n \in N$ olduğunda, Algoritma 3.3'ün çalışma süresi;

$$T(n) = O(2^k + 2^m).$$

Durum 2.

Durum 2'de işlemci sayısı 2^{k+m+2} değerinden az ise, Algoritma 3.2'nin çalışma süresi $n + 2^{adres\ uzunluğu-n}$ değerine eşit olacaktır, burada adres uzunluğu $m + k + 2$ ve işlemci sayısı 2^n'dir. Çizge boyutu 2^n olana kadar her düğümün etiketini hesaplamak n zaman alır, ancak boyut 2^n 'den büyük olduğunda, hesaplama sayısı 2'nin kuvveti ile orantılı olarak artar. $2^{k+m} > 2^n$ ve $n \in N$ olduğunda, Algoritma 3.2'nin çalışma süresi;

$$T(n) = O(n + 2^{adres\ uzunluğu-n}) = O(n + 2^{k+m+2-n})$$

Algoritma 3.3'ün çalışma süresi Durum 1 ile aynıdır, yani bir CSNG(k, m) çizgesinde bir sütundaki köşe sayısı ile bir satırdaki köşe sayısının toplamı olan $O(2^k + 2^m)$, en kötü durum senaryosunda 2B çizgede düğümden düğüme mesaj göndermeye eşittir. $k + m > 2^n$ ve $n \in N$ olduğunda, Algoritma 3.3'ün çalışma süresi;

$$T(n) = O(2^k + 2^m).$$

3.4. YAYIN ALGORİTMALARI

Yayın algoritması, bir bilgisayar ağında bir veya birden fazla kaynaktan birden fazla hedefe bilgi yaymak için kullanılan yöntem veya kurallar kümesidir. Yayın, tek bir göndericinin (kaynak) bir mesajı veya tüm kaynaklardaki mesajları ağdaki diğer tüm düğümlere (hedefler) ilettiği bir iletişim modelidir. Bu iletişim modeli genellikle güncellemelerin dağıtılması, senkronizasyon veya belirli bir olay hakkında tüm düğümlerin bilgilendirilmesi gibi görevler için kullanılır. Yayın algoritmaları, grup iletişimi, ağ yönetimi ve dağıtık sistemlerde senkronizasyon gibi çeşitli uygulamalar için çok önemlidir. Bir yayın algoritmasının seçimi, ağ topolojisi, iletişim gereksinimleri ve verimlilik ile büyüklük arasında istenen denge gibi faktörlere bağlıdır.

Bu bölümde, hiperküpün yayın algoritmaları yardımıyla CSNG(*k, m*) çizgesi üzerinde çalışan yayın algoritmalarına odaklanılacaktır. Bunlar birden-hepsine ve hepsinden-hepsine yayın algoritmalarıdır.

Birden-hepsine yayında, tek bir kaynak ağdaki diğer tüm düğümlere bir mesaj gönderir. Bu, tek noktadan herkese iletişim modelidir. Birden-hepsine yayını gerçekleştirmek için çeşitli algoritmalar ve teknikler kullanılabilir.

Hepsindenden-hepsine yayında, ağdaki her düğüm diğer tüm düğümlere bir mesaj gönderir. Bu, herkesten-herkese yayın iletişim modelidir. Herkese yayın yapmak, özellikle düğüm sayısı arttıkça, birden-hepsine yayından daha karmaşık olabilir.

Bu iletişim modelleri arasındaki seçim, uygulamanın özel gereksinimlerine ve düğümler arasında istenen etkileşime bağlıdır. Birden-hepsine yayın yöntemi genellikle merkezi bir düğümün tüm düğümlere bilgi yayması gereken senaryolar için kullanılırken, her düğümün ağdaki diğer tüm düğümlerle iletişim kurması gerektiğinde hepsinden-hepsine yayın yöntemi kullanılır. Farklı senaryolar ve ağ mimarileri için farklı yayın algoritmaları uygundur.

Algoritma 3.4. Bu algoritma, böl ve yönet yaklaşımını kullanarak CSNG(*k, m*) için hepsinden hepsine yayını hesaplar.

```
Data: my_label = 0, t = 0
Result: MSG mesajı tüm düğümlere iletilir
Function All_to_all(k, m, my_label, my_MSG, t):
      if t < k + m + 2 then
          partner_label = my_label ⊕ 2
          send my_MSG to partner_label
          receive MSG from partner_label
          my_MSG = my_MSG ∪ MSG
          All_to_all(k, m, my_label, my_MSG, t +1)
     end
end function
```

Algoritma 3.5. Bu algoritma, böl ve yönet yaklaşımını kullanarak CSNG(k, m) için birden hepsine yayın hesaplar.

```
Data: my_label = 0, t = k + m +1,
      mask = (2^(k+m+2) − 1) ⊕ 2^(m+k+1)
Result: MSG mesajı bir düğümden tüm düğümlere iletilir
Function OnetoAll(k, m, my_label, MSG, mask, t):
     if t >= 0 then
          if my_label ∩ mask == 0 then
               if my_label ∩ 2^t == 0 then
                    destination_label = my_label ⊕ 2
                    send MSG to destination_label
               end
               else
                    source_label = my_labe ⊕ 2^t
                    receive MSG from source_label
               end
          end
          mask = mask ∩ 2^(t-1)
          OnetoAll(k, m, my_label, MSG, mask, t - 1)
     end
End Function
```

CSNG(*k*, *m*) için birden-hepsine ve hepsinden-hepsine yayın algoritmalarının iteratif versiyonu, [3]'deki Algoritma 3.2 ve Algoritma 3.3'te bulunan $2k + 2$ değerinin yerine $k + m + 2$ yazılarak benzer şekilde elde edilebilir. Algoritma 3.4 ve Algoritma 3.5, [61]'deki hiperküp yayın algoritmalarının böl ve yönet yaklaşımı kullanılarak yeniden tasarlanmış şekilleridir.

Bu 2 algoritmanın maliyeti [61]'da Algoritma 4.2 ve Algoritma 4.7'deki zaman karmaşıklığı denkleminde $logp$ yerine $k + m + 2$ yazarak elde edilebilir.

[61]'deki maliyet analizine göre, birden-hepsine yayın için toplam süre *logp* çarpı noktadan noktaya basit mesaj aktarımının zaman maliyetine eşittir. t_s'nin yayını başlatma için gereken süre ve t_w'nun bir düğümden diğerine bir kelime gönderme

süresidir. Bir düğümden tüm düğümlere n boyutunda bir mesaj göndermek için, zaman karmaşıklığı

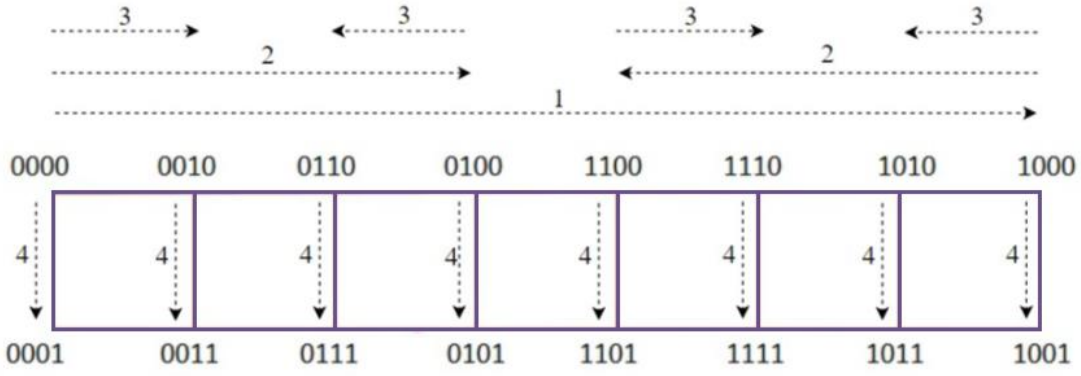
$$T = (t_s + t_w n)(k + m + 2)$$

olacaktır. Bu algoritma, örgü ağlar için yayın algoritmasıyla aynı zaman karmaşıklığına sahiptir. Hepsinden-hepsine yayın algoritmaları için, zaman karmaşıklığı hiperküp ile benzerdir ve zaman karmaşıklığı denklemi

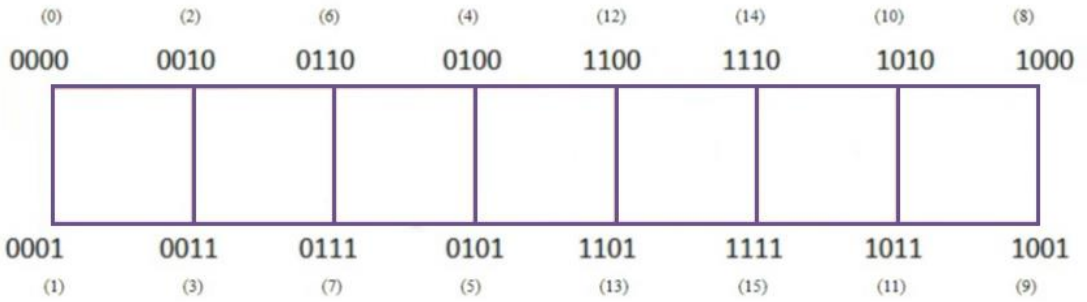
$$T = t_s(k + m + 2) + t_w n(m + k + l)$$

şeklindedir. Örgü ağ için hepsinden hepsine yayın algoritması için çalışma zamanı, $T = 2t_s(\sqrt{p} - 1) + t_w n(p - 1)$'e eşittir; burada p, ağdaki düğüm sayısıdır. Çok sayıda p için $2t_s\left(\sqrt{p} - 1\right) > +t_s logp$ olduğundan, CSNG(*k, m*) için hepsinden-hepsine yayın algoritması örgü ağdan daha iyi performansa sahiptir.

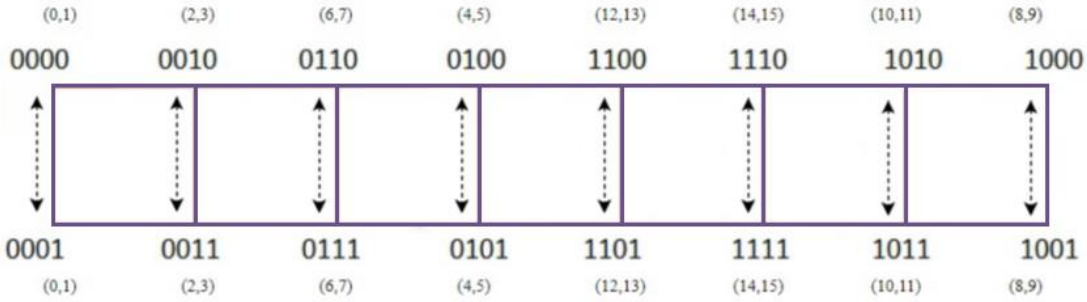
Aslında, bu ağ yapısı CSNG(*k, m*) bir hiperküp varyantı olarak yapılandırıldığından, bu ağ yapısı için yayın algoritmaları, standart örgü algoritmalarından farklı davranış sergiler. Şekil 3.10'da birden-hepsine yayın yönteminin CSNG(*k, m*) çizgesindeki davranışını göstermektedir. CSNG(*k, m*) çizgesindeki hepsinden-hepsine yayın yönteminin davranışı Şekil 3.11'den Şekil 3.15'e kadar olan şekillerde sırasıyla adım adım verilmiştir.



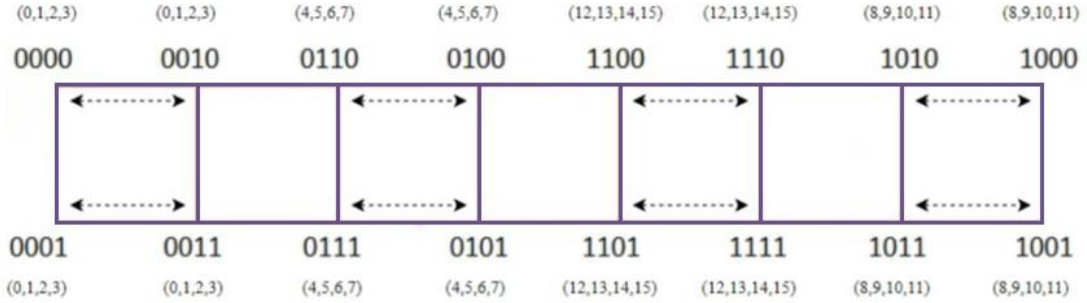
Şekil 3.10. CSNG(*k, m*) için birden-hepsine yayın için tüm adımlar.



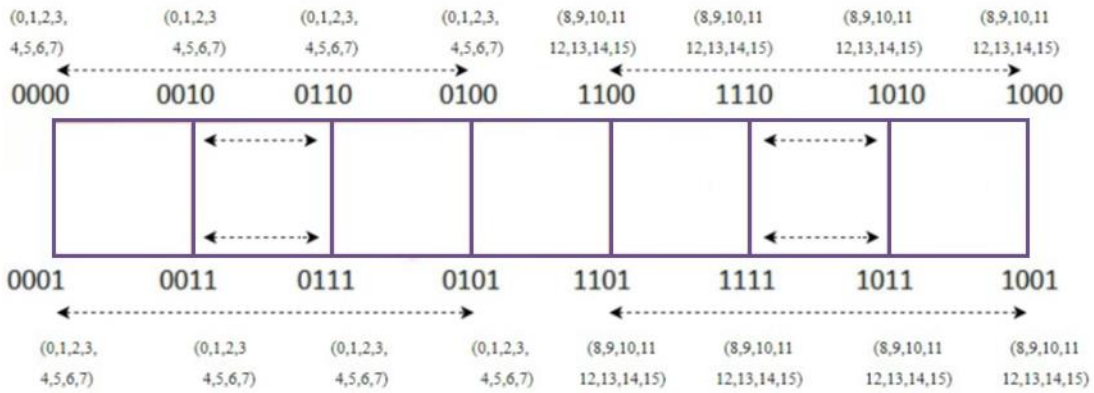
Şekil 3.11. CSNG(*k*, *m*) için hepsinden-hepsine yayın için temel durum.



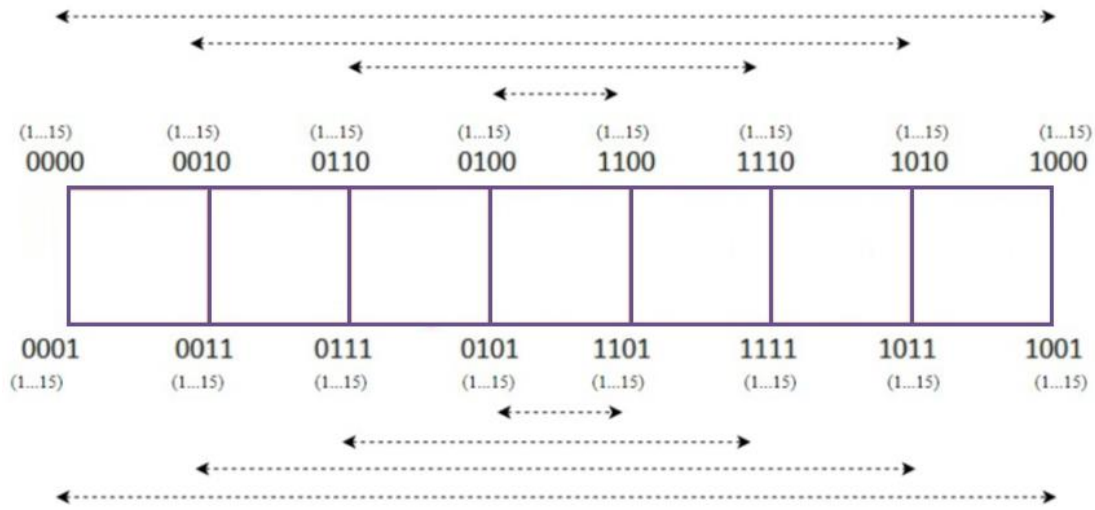
Şekil 3.12. CSNG(*k*, *m*) için hepsinden-hepsine yayın için ilk adım.



Şekil 3.13. CSNG(*k*, *m*) için hepsinden-hepsine yayın için ikinci adım.



Şekil 3.14. CSNG(*k*, *m*) için hepsinden-hepsine yayın için üçüncü adım.



Şekil 3.15. CSNG(*k*, *m*) için hepsinden-hepsine yayın için dördüncü adım.

BÖLÜM 4

FRAKTAL KÜBİK AĞ ÇİZGELERİ (FCNG)

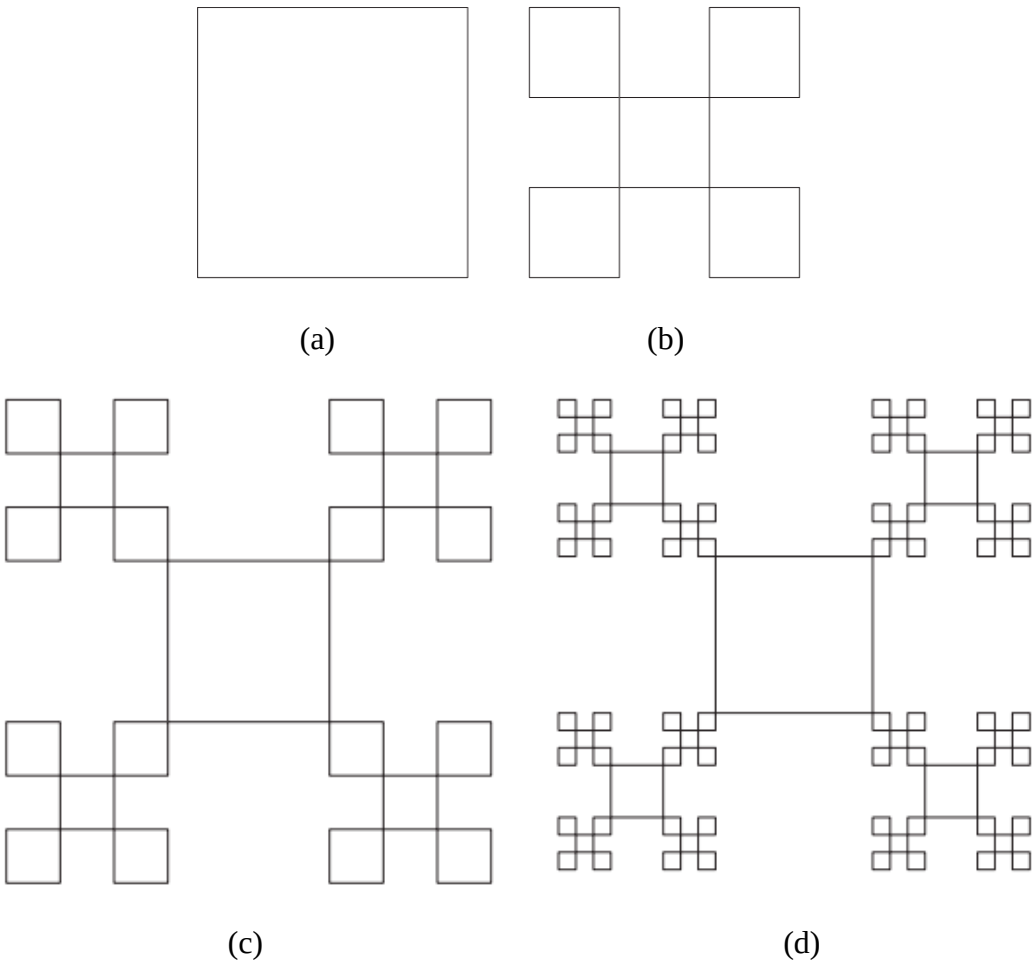
Matematiksel bir kavram olan hiperküp *n* boyutlu küp veya basitçe *n*-küp olarak da bilinen, kare (2-küp) ve küp (3-küp) kavramlarının daha yüksek boyutlara genelleştirilmesidir. Bilgisayar bilimi ve paralel hesaplama bağlamında, hiperküpler genellikle paralel bilgisayarlar için ara bağlantı ağlarını modellemek ve tasarlamak için kullanılan bir topolojidir. Hiperküpteki her bir düğüm bir işlem elemanına karşılık gelir ve kenarlar doğrudan iletişim bağlantılarını temsil eder. Hiperküpler, paralel hesaplamada, yüksek performanslı hesaplama sistemlerinde işlem elemanlarını organize etmek ve bağlamak için yapılandırılmıştır ve verimli bir yol sağlar.

Son yıllarda, *n* boyutlu hiperküplerin H(*n*) ve ana özellikleri kapsamlı bir şekilde incelenmiştir ve üzerine çalışılmıştır [7][39][40][84][85][86][87][88]. Ayrıca hiperküplerin performans artışını sağlamak için hiperküp çeşitleri türetilmiştir. Bunlar, Katlanmış Hiperküp [41][42][89], Çapraz Küp (Bükülmüş Küp) [43][44][45] ve Hiyerarşik Kübik Ağdır [46][47][48][49] en popüler hiperküp türevleridir. Pseudocubes [38] ve spined cube [90][91] hiperküpün üzerinde çalışılmış diğer çeşitleridir. Bu yaklaşımların amacı, hiperküpün çapını azaltırken, avantajlarını korumaktır.

Bir iletişim ağı genellikle bir *G=(E,V)* çizgesi ile temsil edilir; burada *V*, düğümler kümesini ve *E*, kenarlar kümesini temsil eder. İletişim ağlarında kullanılmış olan hiperküp çizgesi H(*n*) yinelemeli olarak tanımlanabilir bir çizgedir. *n* boyutunda H(*n*), her biri *n*-bitlik adresle benzersiz bir şekilde etiketlenebilen 2^n düğüme kadar bağlanır. Düğümlerin *n*-bitlik adresleri arasında tam olarak bir bit fark varsa, iki

düğüm arasında doğrudan bir bağlantı kurulur. H(n)'in popülaritesinin nedeni, topolojik özellikleri, basit yönlendirme algoritmalarını kullanma becerisi ve yaygın olarak yeniden kablolanan ara bağlantı modellerinin yerleştirilmesine izin verme yeteneğidir.

Karcı ve Selçuk [3] tarafından ortaya konulmuş olan Fraktal Kübik Ağ Çizgesi $FCNG_r(n)$ hiper-kübik ağlar olarak tanımlanır ve oluşturulur. $FCNG_r(n)$, Şekil 4.1'deki gibi bir fraktal yapı tarafından oluşturulan yinelemeli bir çizgedir. $FCNG_2(n)$ bir Hamilton çizgesi iken $FCNG_1(n)$ bir Hamilton çizgesi değildir.



Şekil 4.1. a) k=0, b) k=1, c) k=2 ve k=3 için fraktal yapı [3].

Fraktal, klasik geometri ile temsil edilemeyen düzensiz şekiller ve yüzeyler üretmek için daha küçük ölçeklerde tekrarlanan geometrik bir modeldir.

Fraktallardan elde edilen çeşitli çizge türleri vardır [92][93][94][95][96]. [97]'de, Komjathy ve ark. fraktallardan hiyerarşik ölçekten bağımsız çizgelerin oluşturulmasını göstermiştir. [3]'de, Şekil 4.1'deki fraktal yapıdan bir ağ topolojisi üretilmiştir.

Bu bölümde Karcı ve Selçuk'un (2015) üzerinde çalıştığı fraktal kübik ağ çizgeleri (FCNG) yeniden ele alınmıştır. Öncelikle FCNG çizgesinin oluşturulması, ağın analitik özellikleri verilmiş, ardından FCNG çizgesinin yeni topolojik özellikleri sunulmuştur. İkinci olarak, FCNG çizgesinde yönlendirme için yeni bir strateji verilmiş, bir örnek ile açıklanmış ve daha sonra bu stratejiyi kullanan özyinelemeli bir algoritma verilmiştir. Üçüncü olarak, FCNG çizgesi için en kısa yol problemine yönelik bir strateji sunulmaktadır. Benzer bir yönlendirme stratejisi ile FCNG çizgesinin en kısa yol hesabı örnekle gösterilmiş ve daha sonra en kısa yolu hesaplayan özyinelemeli bir algoritma verilmiştir. En kısa yol hesabında kullanılan haritalama algoritması ve en kısa mesafeli bağlantı hesabını yapan alt algoritmalar da örneklerle açıklanmıştır. Tüm algoritmalar için çalışma süreleri hesaplanmıştır.

4.1. ÇİZGENİN OLUŞTURULMASI

4.1.1. $FCNG_1(n)$ Çizgesinin Oluşturulması

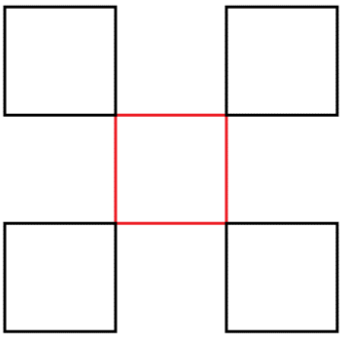
Fraktal Kübik Ağ Çizgeleri aşağıda verildiği gibi tanımlanabilir [3].

$FCNG_1\ (k)\ =\ (V_1\ (k);\ E_1\ (k))\ k\ >\ 0$ için:

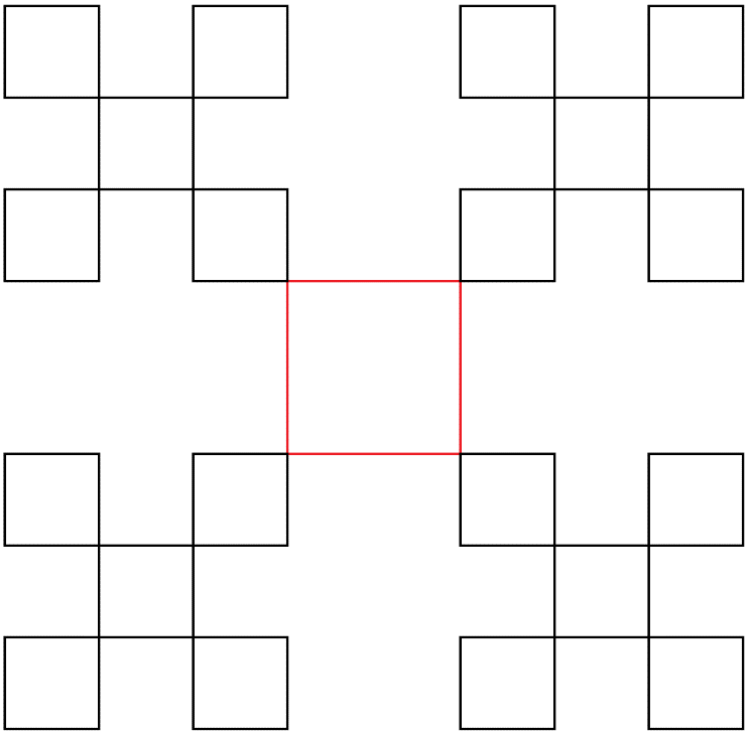
$$\begin{aligned} FCNG_1(k) \;=\; & 11 \,||\, FCNG_1(k\text{-}1) \cup 01 \,||\, FCNG_1(k\text{-}1) \cup \\ & 10 \,||\, FCNG_1(k\text{-}1) \cup 00 \,||\, FCNG_1(k\text{-}1) \end{aligned} \tag{4.1}$$

$$V_1\ (k) = 11 \,||\, V_1(k\text{-}1) \cup 01 \,||\, V_1(k\text{-}1) \cup 10 \,||\, V_1(k\text{-}1) \cup 00 \,||\, V_1(k\text{-}1) \tag{4.2}$$

$$E_1\ (k) = 11 \,||\, E_1(k\text{-}1) \cup 01 \,||\, E_1(k\text{-}1) \cup 10 \,||\, E_1(k\text{-}1) \cup 00 \,||\, E_1(k\text{-}1) \cup E' \tag{4.3}$$



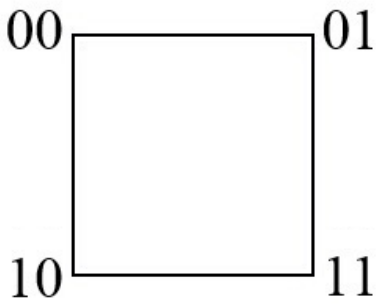
(a)



(b)

Şekil 4.2. a) $FCNG_1(1)$ ve b) $FCNG_1(2)$ çizgelerinde iç bağlantı kenarları E'.

E' kenarları, birbirine eklenen $FCNG_1(k\text{-}1)$ çizgeleri arasındaki iç bağlantı kenarlarıdır ve fraktallar oluşturulurken bir alt sevideki fraktalların birbirine bağlanmasını sağlamaktadır. Şekil 4.2'de $FCNG_1(1)$ ve $FCNG_1(2)$ çizgelerinde iç bağlantı kenarları kırmızı renk ile gösterilmektedir.



Şekil 4.3. $FCNG_1(0)$ etiketlemesi, H(2) ile aynı yapıda [3].

Şekil 4.4'te görülen $FCNG_1(1)$, 4 tane $FCNG_1(0)$ çizgesinin birbirine bağlanmasıyla oluşturulmuştur ve H(4)'ün bir alt çizgesidir. $FCNG_1(1)$ çizgesinde bulunan kenarların nasıl oluşturulduğu Eşitlik 4.4'de adım adım gösterilmektedir:

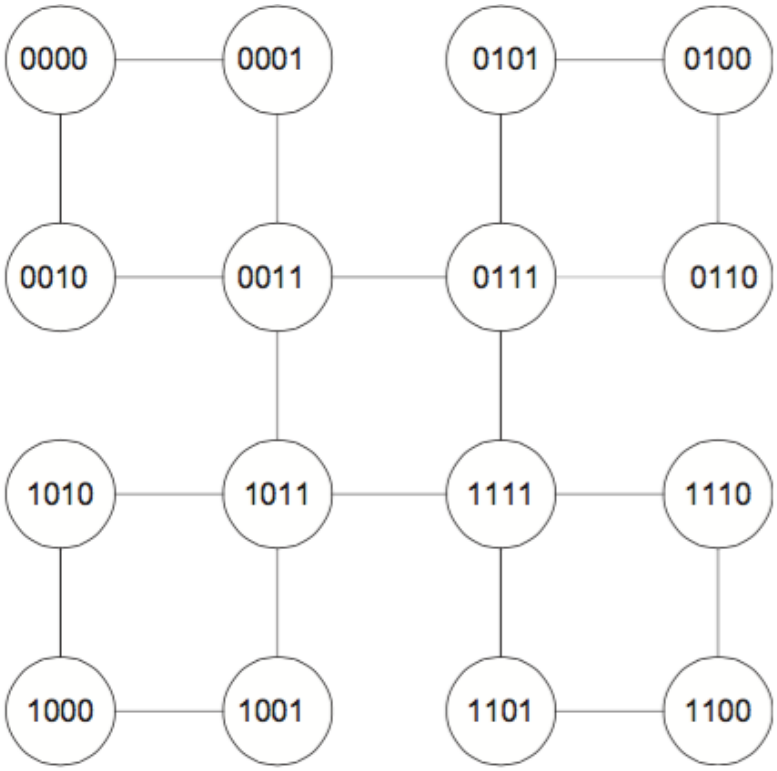
$$
\begin{aligned}
E_1(1) &= 11 \,\|\, E_1(0) \cup 01 \,\|\, E_1(0) \cup 10 \,\|\, E_1(0) \cup 00 \,\|\, E_1(0) \cup E' \\
&= 11 \,\|\, \{(00,01), (00, 10), (10, 11), (01, 11)\} \cup \\
&\quad 01 \,\|\, \{(00,01), (00, 10), (10, 11), (01, 11)\} \cup \\
&\quad 10 \,\|\, \{(00,01), (00, 10), (10, 11), (01, 11)\} \cup \\
&\quad 00 \,\|\, \{(00,01), (00, 10), (10, 11), (01, 11)\} \cup E'
\end{aligned}
\tag{4.4}
$$

İşlem 4.4'ün sonucunda elde edilen $FCNG_1(1)$ çizgesindeki kenarlar ve iç bağlantı kenarları aşağıdaki gibidir;

$\{(1100, 1101), (1100, 1110), (1110, 1111), (1101, 1111)\} \cup$
$\{(0100, 0101), (0100, 0110), (0110, 0111), (0101,0111)\} \cup$
$\{(1000, 1001), (1000, 1010), (1010, 1011), (1001, 1011)\} \cup$
$\{(0000, 0001), (0000, 0010), (0010, 0011), (0001,0011)\} \cup E'$

İç kenarlar:

$$E' = \{(0011, 0111), (0011, 1011), (1011, 1111), (0111, 1111)\} \tag{4.5}$$



Şekil 4.4. $FCNG_1(1)$'in oluşturulması [3].

Lemma 4.1

Fraktal yapıdan elde edilen ağ topolojisinde $FCNG_1(k)$ çizgesinde toplamda 2^{2k+2} düğüm bulunmaktadır [3].

Teorem 4.1

$FCNG_1(k)$ çizgesi $H(2k+2)$'nin alt çizgesidir [3].

Teorem 4.2

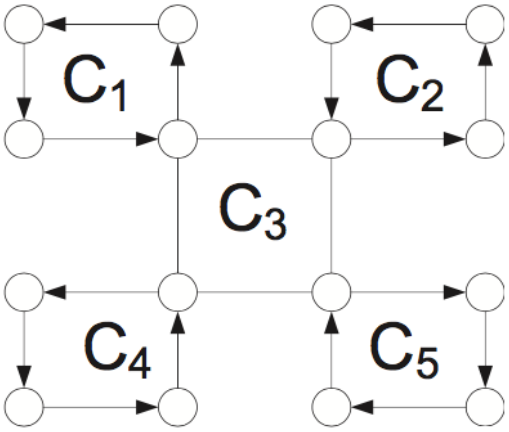
Fraktal yapıdan k değeri için elde edilen ağ topolojisi, $2n + 2$ bitlik etiketler kullanılarak ağ yapısının tüm düğümlerinin gezilmesiyle üretilen bir Euler çizgesidir. $FCNG_1(k)$ 'daki toplam kenar sayısı [3];

$$E_1(k) = 4(4^{k+1}-1)/3 \quad (4.6)$$

ve toplam düğüm sayısı

$$V_1(k) = 2^{2k+2} \tag{4.7}$$

Bir çizgenin Euler çizgesi olabilmesi için çizgedeki tüm kenarlardan sadece bir kere geçmesi, Hamilton çizgesi olabilmesi için bir düğüme yalnız bir kere uğranması gerekmektedir. Şekil 4.5'te $FCNG_1(k)$'nın Euler çizgesi olduğu ve Hamilton çizgesi olmadığı görülmektedir. Bunun nedeni Şekil 4.5'te C_3'ün düğümlerine iki defa uğranmasıdır.



Şekil 4.5. $FCNG_1(1)$ Euler çizgesidir ve $FCNG_1(k)$'ya da uygulanır [3].

Şekil 4.1'de $FCNG_1(0)$, $FCNG_1(1)$, $FCNG_1(2)$ ve $FCNG_1(3)$'ün örnekleri, Şekil 4.6'da $FCNG_1(3)$'ün ve Şekil 4.7'de $FCNG_1(4)$'ün örnekleri verilmiştir. Verilen tüm örneklerde de tüm çizge dolaşılırken kenarlardan sadece bir defa geçildiği için Euler çizgesidir, ama çizgedeki bazı düğümlere birden fazla uğranması gerektiğinden çizge Hamilton çizgesi değildir.

Teorem 4.3

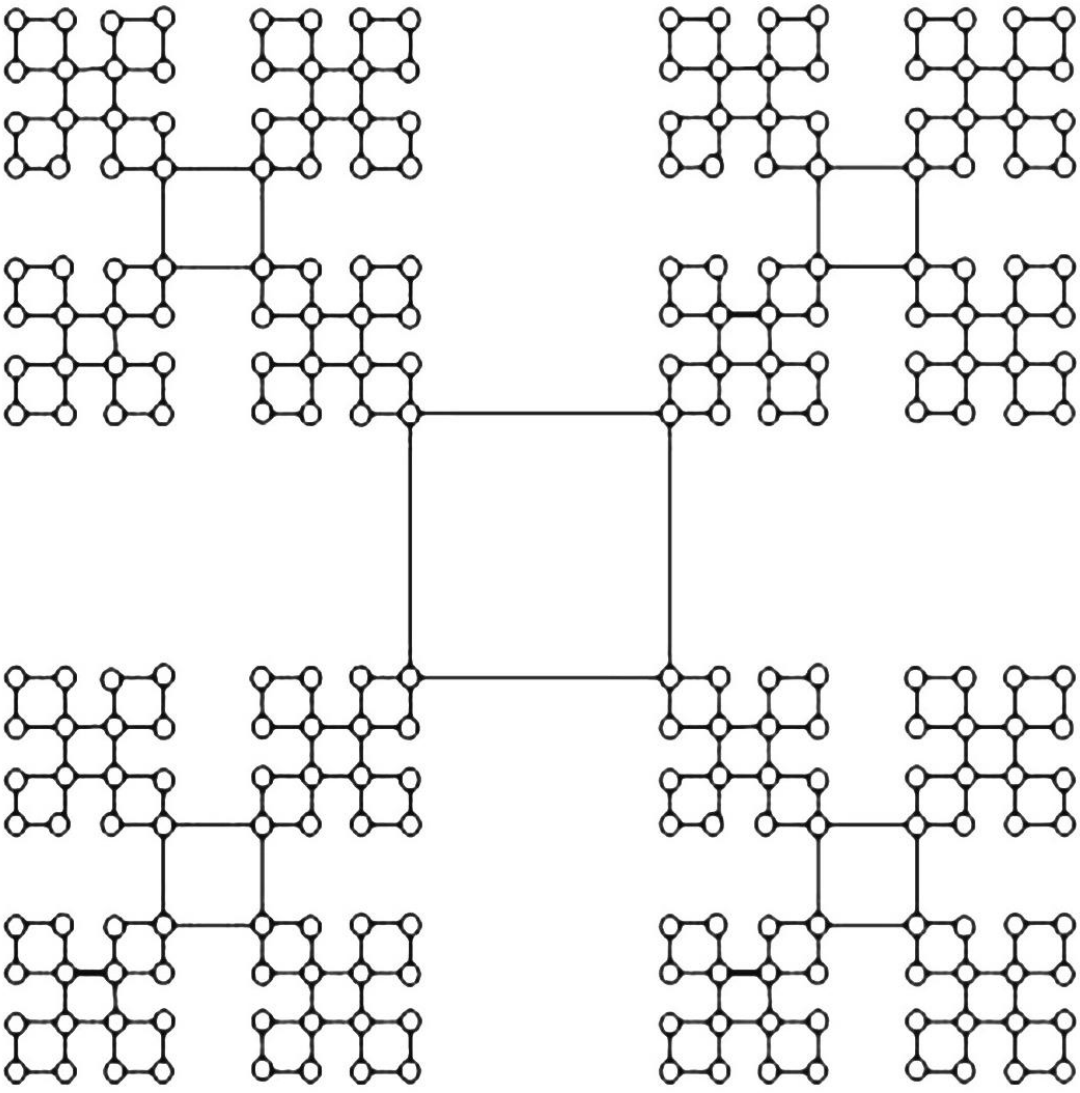
$FCNG_1(k)$'nın kenar sayısı, Eşitlik 4.8'deki gibi bir yineleme bağıntısı olarak tanımlanır ve Eşitlik 4.9'da verilmiştir [3].

$|E_1(k)| = 4|E_1(k\text{-}1)| + 4$ (4.8)

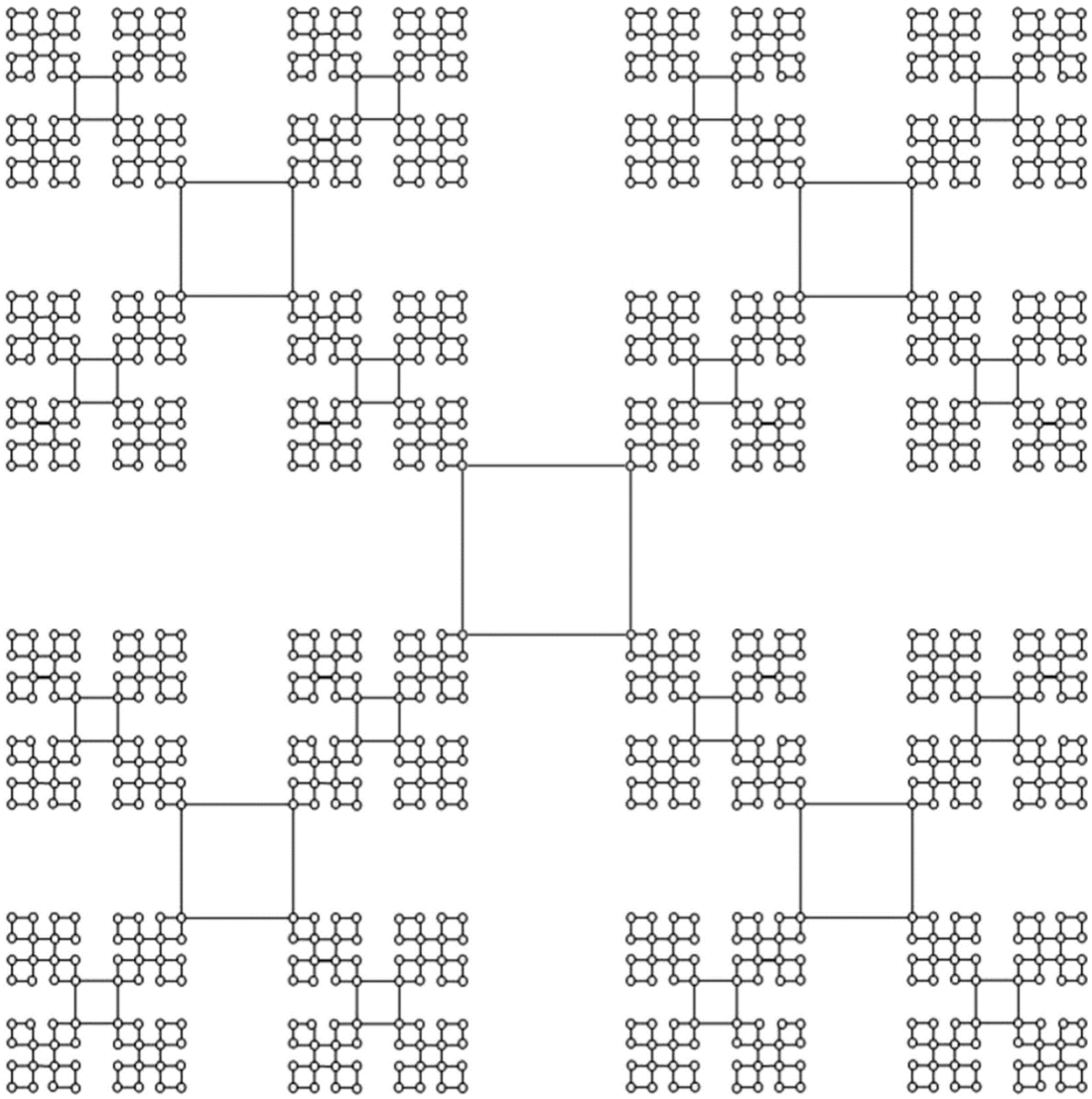
$|E_1(k)| = (4^{k+2}\text{-}4)/3$ (4.9)

Çizelge 4.1. $FCNG_1(k)$ için kenar-düğüm ilişkisi.

deg(2), deg(2)	deg(3), deg(4)	deg(4), deg(4)	Toplam kenar sayısı
2.4^k	2.4^k	$\frac{4^{k+1}-4}{3}$	$\frac{4^{k+2}-4}{3}$



Şekil 4.6. $FCNG_1(3)$'ün fraktal yapısı [3].



Şekil 4.7. $FCNG_1(4)$'ün fraktal yapısı.

Teorem 4.4

$FCNG_1(k)$ çizgesinde bulunan düğümlerin dereceleri sadece 2 ve 4'tür, derecesi 3 olan düğüm $FCNG_1(k)$ çizgesinde bulunmamaktadır. $FCNG_1(k)$ çizgesinde derecesi 2 olan düğüm sayısı $(2.4^{k+1} + 4)/3$ ve derecesi 4 olan düğüm sayısı $4(4^k-1)/3$'tür.

Çizelge 4.2. $FCNG_1(k)$'nın düğüm dereceleri 2 ve 4 olan düğümlerin sayısı [3].

Boyut *(2k+2)*	Düğüm Sayısı	Derecesi *2* olan düğüm sayısı	Derecesi *4* olan düğüm sayısı
0	4	$2^2 = 4$	0
1	16	$2^4 - 2^2 = 12$	4
2	64	$2^6 - 2^4 - 2^2 = 44$	$2^4 + 2^2 = 20$
3	256	$2^8 - 2^6 - 2^4 - 2^2 = 172$	$2^6 + 2^4 + 2^2 = 84$
4	1024	$2^{10} - 2^8 - 2^6 - 2^4 - 2^2 = 684$	$2^8 + 2^6 + 2^4 + 2^2 = 340$
5	4096	$2^{12} - \sum_{i=1}^{5} 2^{2i} = 2732$	$\sum_{i=1}^{5} 2^{2i} = 1364$
...	...	...	...
k	2^{2k+2}	$2^{2k+2} - \sum_{i=1}^{k} 2^{2i} = (2.4^{k+1} + 4)/3$	$\sum_{i=1}^{k} 2^{2i} = 4.(4^k - 1)/3$

4.1.2. $FCNG_2(k)$ Çizgesinin Oluşturulması

$FCNG_2(k) = (V_2(k), E_2(k))$ çizgesi, dört tane $FCNG_2(k\text{-}1)$ çizgesi kullanılarak (k = 1,2,3,...) oluşturulabilir [3].

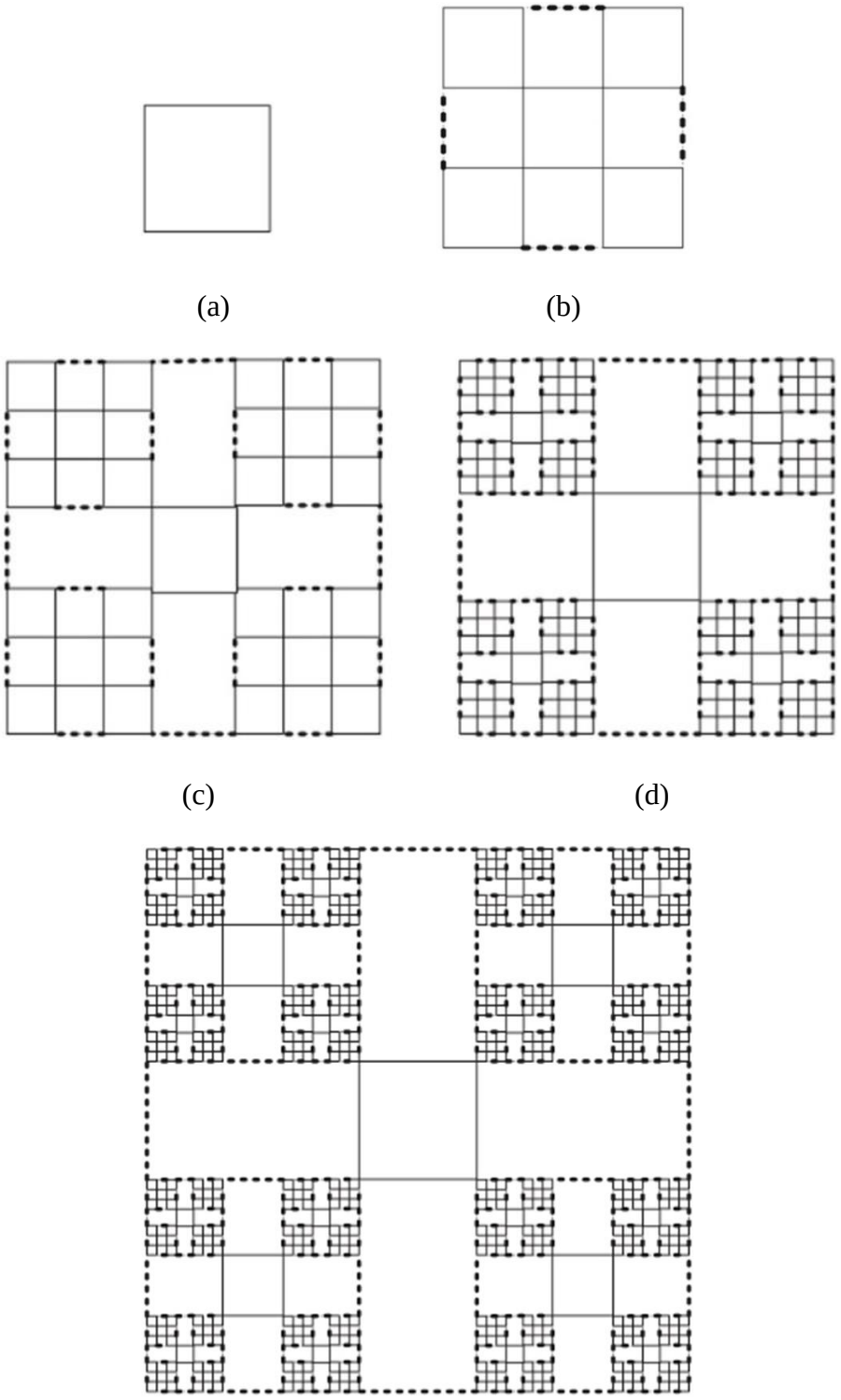
$$FCNG_2(k) = 11 \,||\, FCNG_2(k\text{-}1) \cup 01 \,||\, FCNG_2^{-1}(k\text{-}1) \cup 00 \,||\, FCNG_2(k\text{-}1) \cup 10 \,||\, FCNG_2^{-1} \cup (k\text{-}1) \quad (4.10)$$

$FCNG_2^{-1}(k-1)$ çizgesinin etiketlemesi $FCNG_2(k-1)$ çizgesinin etiketlemesinin tersidir.

$$V_2(k) = 11 \,||\, V_2(k\text{-}1) \cup 01 \,||\, V_2(k\text{-}1) \cup 00 \,||\, V_2(k\text{-}1) \cup 10 \,||\, V_2(k\text{-}1) \quad (4.11)$$

$$E_2(k) = 11 \,||\, E_2(k\text{-}1) \cup 01 \,||\, E_2(k\text{-}1) \cup 00 \,||\, E_2(k\text{-}1) \cup 10 \,||\, E_2(k\text{-}1) \cup E' \quad (4.12)$$

E' kenarları birbirine eklenen $FCNG_1$ (k-1) çizgeleri arasındaki dış bağlantı kenarlarıdır.

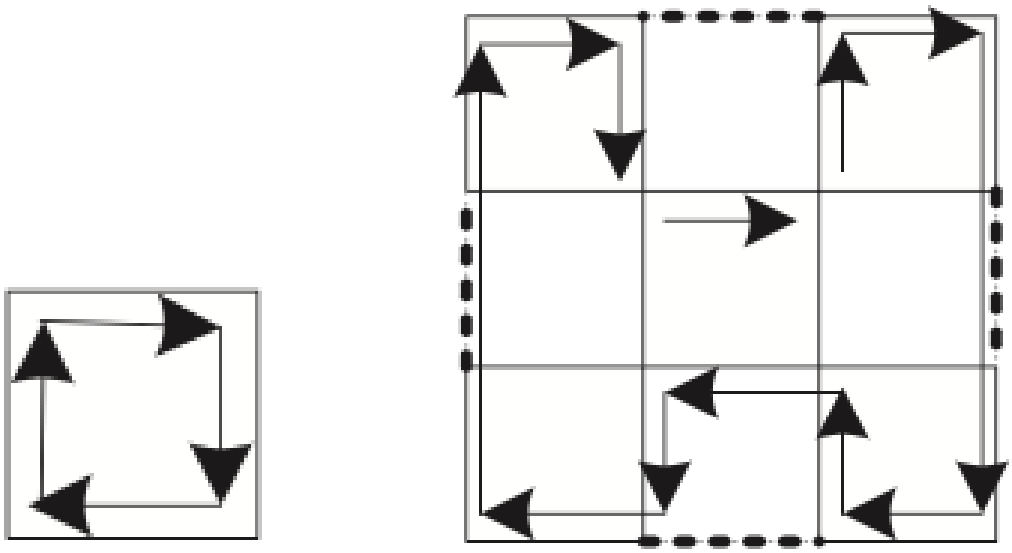


Şekil 4.8. $FCNG_2(k)$ Çizgesinin örnekleri a) k=0, b) k=1, c) k=2, d) k=3, e) k=4 [3].

Şekil 4.8'de $FCNG_2(k)$ Çizgesinin bazı örnekleri yer almaktadır ve noktalarla gösterilen kenarlar dış bağlantı kenarlarıdır. $FCNG_2(k)$ çizgesi bazı düğümleri tek sayıda dereceye sahip olduğu için Euler çizgesi değildir. Şekilde görüldüğü gibi bu çizge fraktal bir yapıya sahiptir.

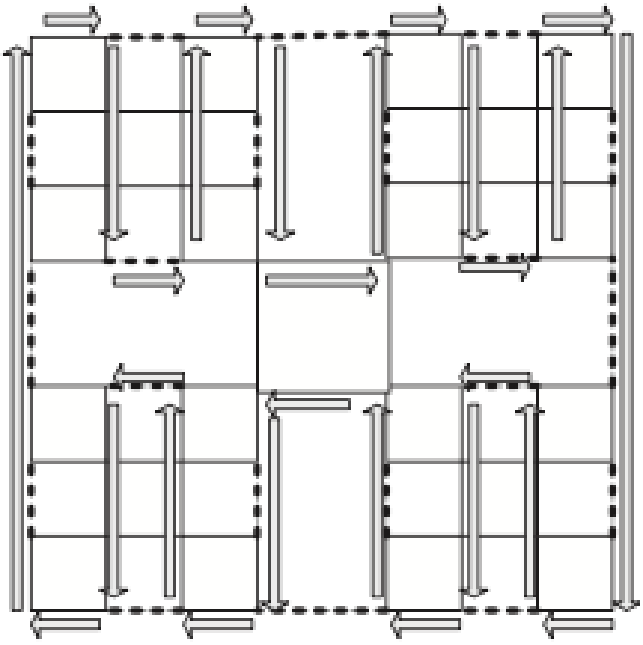
Teorem 4.5

$FCNG_2(k)$ çizgesi bir Hamilton çizgesidir. Şekil 4.9'de sırasıyla k değeri 1, 2 ve 3 için $FCNG_2(k)$ çizgesinin bir Hamilton çizgesi olduğunu gösteren matematiksel tümevarımın temel adımları ve Şekil 4.10'da kanıtı bulunmaktadır. Bir çizgenin Hamilton çizgesi olabilmesi için çizgedeki tüm düğümlere sadece bir defa uğraması gerekmektedir. FCNG(k) çizelgesinde tüm düğümlere birer kez uğrayarak bütün çizge üzerinde gezilebilmektedir [3].



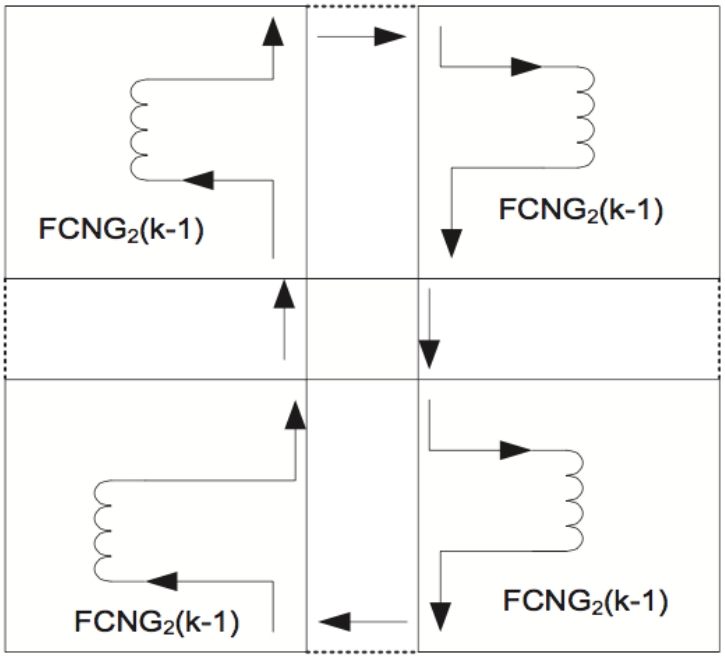
(a) (b)

Şekil 4.9. Sırasıyla a) k=0, b) k=1 için $FCNG_2(k)$ çizgesinin Hamilton çizgesi olduğunu gösteren tümevarımın temel adımları [3].



(c)

Şekil 4.9. (devam ediyor) c) k=3 için $FCNG_2(k)$ çizgesinin Hamilton çizgesi olduğunu gösteren tümevarımın temel adımları [3].



Şekil 4.10. $FCNG_2(k)$ çizgesinin Hamilton çizgesi olduğunun kanıtı [3].

Teorem 4.6

$FCNG_2(k)$'nın kenar sayısı, Eşitlik 4.13'teki gibi bir yineleme bağıntısı olarak tanımlanır ve bu yineleme bağlantısının sonucu Eşitlik 4.14'te verilmiştir [3].

$$|E_2(k)| = 4|E_2(k\text{-}1)| + 8 \tag{4.13}$$

$$|E_2(k)| = (5.4^{k+1}\text{-}8)/3 \tag{4.14}$$

Çizelge 4.3. $FCNG_2(k)$ için kenar-düğüm ilişkisi

deg(2), deg(3)	deg(3), deg(3)	deg(3), deg(4)	deg(4), deg(4)	Toplam kenar sayısı
2.4^k	$\frac{4^{k+1}-4}{3}$	2.4^k	$\frac{4^{k+1}-4}{3}$	$\frac{5.4^{k+1}-8}{3}$

Teorem 4.7

$FCNG_2(k)$ çizgesinde bulunan düğümlerin dereceleri 2, 3 ve 4'tür. Derecesi 2 olan düğüm sayısı 4, derecesi 3 olan düğüm sayısı $8(4^k\text{-}1)/3$'tür ve derecesi 4 olan düğüm sayısı, $4(4^k\text{-}1)/3$'tür [3].

Çizelge 4.4'te çizgenin boyutu, düğüm sayısı ve düğümlerin derecelerine göre derecesi 2, 3, ve 4 olan düğümlerin sayıları tümevarım yöntemi ile adım adım hesaplanmıştır. Öncelikle boyutu 0'dan 5'e kadar olan çizgelerin boyutları tek tek hesaplanmış, daha sonra çizelgenin en altındaki toplam formülü elde edilmiştir. Her adımda elde edilen toplam formülünün sonucu hesaplanarak çizelgeye eklenmiştir.

Çizelge 4.4. $FCNG_2$(k)'nın düğüm dereceleri 2, 3 ve 4 olan düğümlerin sayısı [3].

Boyut *(2k+2)*	Düğüm Sayısı	Derecesi *2* olan düğüm sayısı	Derecesi *3* olan düğüm sayısı	Derecesi *4* olan düğüm sayısı
0	4	$2^2 = 4$	0	0
1	16	$2^2 = 4$	$2^3 = 8$	$2^2 = 4$
2	64	$2^2 = 4$	$2^5 + 2^3 = 40$	$2^4 + 2^2 = 20$
3	256	$2^2 = 4$	$2^7 + 2^5 + 2^3 = 168$	$2^6 + 2^4 + 2^2 = 84$
4	1024	$2^2 = 4$	$2^9 + 2^7 + 2^5 + 2^3 = 680$	$2^8 + 2^6 + 2^4 + 2^2 = 340$
5	4096	$2^2 = 4$	$\sum_{i=1}^{5} 2^{2i+1} = 2728$	$\sum_{i=1}^{5} 2^{2i} = 1364$
…	…	…	…	…
k	2^{2k+2}	$2^2 = 4$	$\sum_{i=1}^{k} 2^{2i+1} = 8(4^k - 1)/3$	$\sum_{i=1}^{k} 2^{2i} = 4.(4^k - 1)/3$

4.2. TOPOLOJİK ÖZELLİKLER

Bir çizgenin topolojik özellikleri, yapısal özelliklerini ve bağlantı modellerini tanımlar. Bu özellikler, bir çizgedeki düğümlerin ve kenarların arasındaki ilişkileri anlamak için temeldir.

Topolojik özellikler, paralel ve dağıtık bilgi işlem sistemleri için ara bağlantı ağları tasarlanırken dikkate alınması gereken kritik hususlardır. Ara bağlantı çizgesinin seçimi, sistemin ölçeklenebilirliğini, performansını ve hataları ve iletişimi verimli bir şekilde ele alma yeteneğini etkiler.

Bu bölümde FCNG(*k*) çizgeleri için daha önce incelenmemiş yeni topolojik özelliklerden bahsedilecektir. Öncelikle r = 1, 2 olmak üzere $FCNG_r$ (*k*)'nın yarıçapı Teorem 4.8'de belirtildiği biçimdedir.

Teorem 4.8

(i) $FCNGr\,(k)$'nın çapı $2^{k+2}-2$,

(ii) $FCNGr\,(k)$'nın yarıçapı 2^{k+1}.

Kanıt

(i) Kanıt için matematiksel tümevarım kullanıldığında aşağıdaki adımlar izlenir:

Temel adım ($k = 1$). $FCNG_r(1)$'nın çapı aşağıdaki eşitlikte verildiği gibidir ve temel durumda değeri 6'ya eşittir.

$$\text{diam} = \max\{ec(v) |\ v \in V(G)\} = 6 \tag{4.15}$$

Hipotez adımı ($k = m-1$). $FCNGr\,(m-1)$'in çapı Eşitlik 4.16 ile ifade edilir.

$$\text{diam}(FCNGr\,(m\text{-}1)) = 2^{m+1}\text{-}2 \tag{4.16}$$

Son adım ($k = m$). [3] Bölüm 7'de açıklandığı gibi $\text{diam}(FCNG_r\,(0)) = 2$ ve $r = 1, 2$ olduğunda, $FCNG_r\,(m)$'nin çapı Eşitlik 4.17'de verildiği gibidir.

$$\text{diam}(FCNG_r\,(m)) = 2.\text{diam}(FCNG_r\,(m-1)) + 2 \tag{4.17}$$

Böylelikle $FCNGr\,(k)$'nin çapı Eşitlik 4.18 olarak elde edilir.

$$\text{diam}(FCNG_r\,(m)) = 2.(2^{m+1}-2) + 2 = (2^{m+2}-2) \tag{4.18}$$

(ii) Şekil 1 ve Şekil 2'den de anlaşılacağı gibi, $FCNG_r\,(k)$'nın yarıçapı ve çapı arasında aşağıdaki eşitlik aşağıda verilen eşitliktir.

$$\text{rad}(FCNG_r\,(k)) = \text{diam}(FCNG_r\,(k-1)) + 2 \tag{4.19}$$

Bunun nedeni $FCNG_r(k)$, $FCNG_r(k-1)$ alt problemlerini bir kenara bağlayan

4 merkezi düğüme sahiptir ve bir karenin tüm düğümlerinin dış merkezliği 2 olduğundan yarıçapı 2'dir. Böylece $FCNG_r(k)$'nın yarıçapı Eşitlik 4.20'deki gibi hesaplanır.

$$\text{rad}(FCNG_r(k)) = 2^{k+1} - 2 + 2 = 2^{k+1} \tag{4.20}$$

Örnek 4.1

$k = 2$, $G = (FCNG_r(k))$ ve $r = 1, 2$ olsun. Teorem 2.1 yardımıyla, $y \in V(G)$ olduğunda çizgenin yarıçapı ve çapı sırasıyla;

$$rad = min\ ec(y) = 8 \tag{4.21}$$

$$diam = max\ ec(y) = 14 \tag{4.22}$$

Bir köşeden diğer köşeye olan maksimum mesafeyi hesaplayarak dışmerkezliği belirleyebiliriz. Yani tepe noktasının dışmerkezliği, bir tepe noktasından diğer tüm köşe noktalarına olan maksimum mesafe olacaktır.

Örnek 4.1'de verilen çizgenin toplam dışmerkezliği (eccentricity):

$$\sum_{y \in V(G)} ec(y) = 4.14 + 16.13 + 20.12 + 8.11 + 4.10 + 8.9 + 4.8 = 736 \tag{4.23}$$

Sonuç 4.1

Teorem 4.8'den Çizelge 4.2 elde edilmiştir. $FCNG_2(k)$ çizgesi 2B Kare Örgü(m) çizgesine göre daha seyrek bir yapıda tanımlandığından maliyet $2(m^2 - m)$ yerine $4(2^{k+2} - 2)$ olarak elde edilir. Bu iki çizgeyi aynı boyutta yapmak için $m = 2^{k+1}$ olduğunu varsayalım. Bu şekilde m değeri yerine 2^{k+1} değeri konulduğunda, 2B Kare Örgü(m) çizgesinin maliyeti $2(m^2 - m) = 2(2^{2k+2} - 2^{k+1})$, $k > 1$ olduğunda $4(2^{k+2} - 2)$'den daha büyüktür.

Çizelge 4.5. Ara bağlantı ağlarının karşılaştırılması.

Ağ	Boyut	Derece	Çap	İkiye bölme genişliği	Maliyet
2B Kare Örgü(m)	m^2	4	2(m-1)	m	$2(m^2-m)$
Hiperküp H(n)	2^n	n	n	2^{n-1}	$n2^{n-1}$
$FNCG_2(k)$	2^{2k+2}	4	$2^{k+2}-2$	2	$4(2^{k+2}-2)$

4.3. YÖNLENDİRME ALGORİTMALARI

$FCNG_r(k)$ üzerindeki yönlendirme algoritmaları, hiperküplerdeki yönlendirme algoritmalarına benzer. $FCNG_r(k)$ $(2k + 2)$ bit etiketlere sahiptir ve yönlendirme algoritmaları bu uzunluğa göre tasarlanmıştır. Yönlendirme algoritmaları, birden-hepsine yayın, hepsinden-hepsine yayın ve hepsinden-hepsine kişiselleştirilmiş olacak şekilde 3 farklı uygulaması olan iletişim algoritmalarıdır. [3]'de verilen algoritmaların tümü, hiperküpler için yönlendirme algoritmalarının yeniden yapılandırılmış şekli olmaktadır.

Temel olan üç algoritma vardır ve bunların zaman karmaşıklıkları analiz edilebilir. Birinci algoritma, birden-hepsine yayın algoritmasıdır ve amacı, kaynaktan diğer tüm düğümlere mesaj göndermektir. t_s'nin başlangıç zamanı ve t_w'nun bir düğüme kenar üzerinden mesaj gönderme zamanı olduğunu varsayalım. Mesaj gönderme stratejisinin sakla ve ilet olduğunu varsayalım. Mesaj gönderme adımı sayısı için üst sınır $2k + 2$'dir. $FCNG_1(k)$ ve $FCNG_2(k)$ için $mT_{birden-hepsine}$ boyutunda mesaj gönderme zaman karmaşıklığı;

$$T_{birden-hepsine} \leq 2(2k+2)(t_s + mt_w) \tag{4.24}$$

şeklindedir. İkinci algoritma hepsinden-hepsine yayın algoritmasıdır ve $mT_{hepsinden-hepsine}$ boyutundaki mesajları herkese göndermek için zaman karmaşıklığı

$$mT_{hepsinden-hepsine} \leq (2k+2)(t_s + (2k+2)mt_w) \quad (4.25)$$

m boyutlarındaki mesajların gönderilmesinde hepsinden-hepsine kişiselleştirilmiş yayın algoritması için zaman karmaşıklığı Eşit 4.25'teki verildiği gibi elde edilmektedir.

$$mT_{hepsinden-hepsine_{kişisel}} \leq (2k)\left(t_s + 2^{2k}mt_w\right) + 2\left(t_s + 3.2^{2k}mt_w\right) + 2\left(t_s + 3.2^{2k}mt_w\right) \quad (4.25)$$

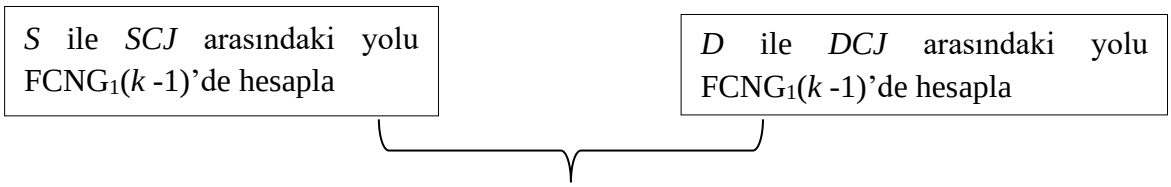
4.3.1. Tek Noktaya Yönlendirme için Yeni Strateji

Tek noktaya yayın iletiminde, veriler tek bir göndericiden (tek bir kaynak düğümden) tek bir alıcıya (tek bir hedef düğüme) aktarılır. Kaynak ve hedef düğümün adresleri bellidir ve mesajı iletmek için bu iki düğüm arasındaki diğer düğümler üzerinden mesaj kaynaktan hedefe iletilir. Bu, ağlar üzerinden veri aktarımının en yaygın şeklidir. $FCNG_1(k)$ çizgesi üzerinde uygulanacak tek noktaya yayın algoritması tüm $FCNG_r(k)$ çizgeleri üzerinde uygulanabilmektedir çünkü $FCNG_1(k)$ diğer $FCNG_r(k)$ çizgelerinin alt çizgesidir. $FCNG_1(k)$ için rekürsif tek noktaya yayın algoritması öncelikle hedef ve kaynak arasında gidilecek olan yolu hesaplar ardından bu yol üzerinden mesajı iletir.

Algoritma $FCNG_1(k)$ için yol hesaplarken önce kaynak düğüm ile kaynağın bulunduğu k-1 derecesine sahip $FCNG_1(k\text{-}1)$ fraktalının bağlantı noktası olan düğüm arasındaki yolu hesaplar. Ardından kaynağın bulunduğu $FCNG_1(k\text{-}1)$ boyutundaki fraktalın bağlantı noktası olan düğüm ile hedefin bulunduğu $FCNG_1(k\text{-}1)$ boyutundaki fraktalın bağlantı noktası olan düğüm arasındaki yolu daha önce hesaplanmış olan yolun devamına ekler. Son olarak hedefin bulunduğu $FCNG_1(k\text{-}1)$ boyutundaki fraktalın bağlantı düğümü ile hedef düğüm arasındaki yolu hesaplayarak tüm yolu oluşturur. İlk ve son hesaplamalar $FCNG_1(k\text{-}1)$, $FCNG_1(k)$'nın alt problemleridir ve bu alt problemler aynı algoritma kullanılarak özyinelemeli olarak hesaplanır. İki fraktalın bağlantı düğümleri arasındaki mesafe 1 ya da 2'dir.

- *S*: Kaynak
- *SCJ*: Kaynağın bulunduğu $FCNG_1(k\text{-}1)$ boyutlu fraktalın bağlantı düğümü
- *D*: Hedef
- *DCJ*: Hedefin bulunduğu $FCNG_1(k\text{-}1)$ boyutlu fraktalın bağlantı düğümü
- *MSG*: Gönderilen ileti

Verilen değişkenlere göre $FCNG_1(k)$ için rekürsif tek yöne yayın algoritmasının çalışma yapısı aşağıdaki gibidir:



SCJ ile *DCJ* arasındaki yolu $FCNG_1(k)$'da hesapla

Yukarıda gösterilen algoritma ismi RoutePath ve içine aldığı girdiler *S, D, k, P, MSG* olacak şekilde verilen düğüm tanımları kullanılarak aşağıdaki eşitlik ile ifade edilebilir;

$$RoutePath(S,D,k,P,MSG) = \begin{cases} [RoutePath(S,SCJ,k-1,P,MSG) \cup \\ SCJ \oplus 10(0 \dots 0) \cup \\ DCJ \oplus 01(0 \dots 0) \cup \\ RoutePath(DCJ,D,k-1,P,MSG)] \\ ||[RoutePath(S,SCJ,k-1,P,MSG) \cup \\ SCJ \oplus 10(0 \dots 0) \cup \\ RoutePath(DCJ,D,k-1,P,MSG)] \\ ||[RoutePath(S,SCJ,k-1,P,MSG) \cup \\ SCJ \oplus 01(0 \dots 0) \cup \\ RoutePath(DCJ,D,k-1,P,MSG)] \end{cases} \quad (4.26)$$

Algoritma 4.1. Bu algoritma, $FCNG_1(k)$ veya $FCNG_2(k)$ için tek noktaya yayın yönlendirmesini hesaplar.

```
Data: S=kaynak, D=hedef, k=boyut, P Yol, MSG=mesaj.
Result: MSG mesajı S'den D'ye iletilir
Function Routing (S, D,k, P, MSG):
  T=S
  X = Dk ⊕ Sk
  if k> 0 then
    /* Dış fraktallar arasındaki mesafe 0 değilse */
    if X != 00 then
      /* Bir sonraki hedefi bağlantı düğümü olarak
         hesaplayın */
      Tk-1=11
      if k> 1 then
        for i=k-2 to 0 do
           Ti=00
        end
      end
      /* Kaynaktan bağlantı düğümüne giden bir yol
         hesaplayın */
      Routing (S, T, k-1, P, MSG)
      /* Hedef düğümün ve kaynak düğümün bağlantı düğümleri
         arasındaki yolu hesaplayın */
      if XL == 1 then
        Tk=Tk⊕01
        add to path(P,T)
      end
      if XH == 1 then
        Tk=Tk⊕10
        add to path(P,T)
      end
      /* Hedef fraktal bağlantı düğümünden hedefe giden bir
         yol hesaplayın
      Routing(T, D, k - 1, P, MSG)
    end
    else
```

```
        /*  Kaynak  ve  hedef  düğümler  aynı  FCNG(k-1)
            fraktalında ise bir yol hesaplayın */
        Routing(S, D, k - 1, P, MSG)
      end
    end
    else
       /* FCNG1(0)'da hedeften kaynağa giden yolu hesaplayın
           */
      if X=1 then
        Tk=Tk⊕01
        add to path(P,T)
      end
      if XH == 1 then
        Tk=Tk⊕10
        add to path(P,T)
      end
    end
end function
```

Açıklama 4.1

Algoritma 4 . 1'in ilişkisel denklemi

$$T(k) = 2T(k - 1) + O(k) \tag{4.27}$$

olup, Algoritma 4.1'in zaman karmaşıklığı tüme varım yöntemi yardımıyla hesaplanabilmektedir. Bu formülde

$$T(k - 1) = 2T(k - 2) + \Theta(k - 1) \tag{4.28}$$

yerine konulursa

$$T(k) = 2T(k - 2) + 2\Theta(k) + \Theta(k - 1) \tag{4.29}$$

elde edilir.

Bu özyinelemeli hesap genelleştirilirse, h = 1, 2, ..., k – 1 olduğunda;

$$T(k) = 2^h T(k-h) + 2^{h-1}\Theta(k-(h-1)) + 2^{h-2}\Theta(k-(h-2)) + ... + \Theta(k) \quad (4.30)$$

h = k – 1 için;

$$T(k) = 2^k T(0) + 2^{k-1}\Theta(1) + 2^{k-2}\Theta(2) + ... + 2\Theta(k-1) + \Theta(k) \quad (4.31)$$

eşitlikleri elde edilir. Algoritma 4.1'in zaman karmaşıklığı $\Theta(2^k)$ olup başlangıç değeri $T(0) = \Theta(1)$.

Örnek 4.2

$FCNG_1(2)$ veya $FCNG_2(2)$ çizgesinde kaynak düğümün V_S = {00 10 00} ve hedef düğümün V_D = {11 01 00} olduğu durumda algoritma çalışması şu şekildedir (*P* kaynak ile hedef arasında oluşturulacak yol, ilk başta boştur ({})):

Routing(00 10 00, 11 01 00, 2, *P*, *MSG*)
  /* Kaynak ile kaynağın bulunduğu fraktalın bağlantı düğümü arası yolu hesapla
  Routing(00 10 00, 00 11 00, 1, *P*, *MSG*)
    Routing(00 10 00, 00 10 11, 0, *P*, *MSG*)
      P = {00 10 01}
      P = {00 10 01, 00 10 11}
      P = {00 10 01, 00 10 11, 00 11 11}
    Routing(00 11 11, 00 11 00, 0, *P*, *MSG*)
      P = {00 10 01, 00 10 11, 00 11 11, 00 11 10}
      P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00}
  /* Kaynak ile hedefin bulunduğu fraktalların bağlantı düğümleri arası yolu hesapla
  P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00}
  P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00}

/* Hedefin bulunduğu fraktalın bağlantı düğümü ile hedef arası yolu hesapla

Routing(11 11 00, 11 01 00, 1, *P*, *MSG*)

Routing(11 11 00, 11 11 11, 0, *P*, *MSG*)

P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01}

P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 11 11}

P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 11 11, 11 01 11}

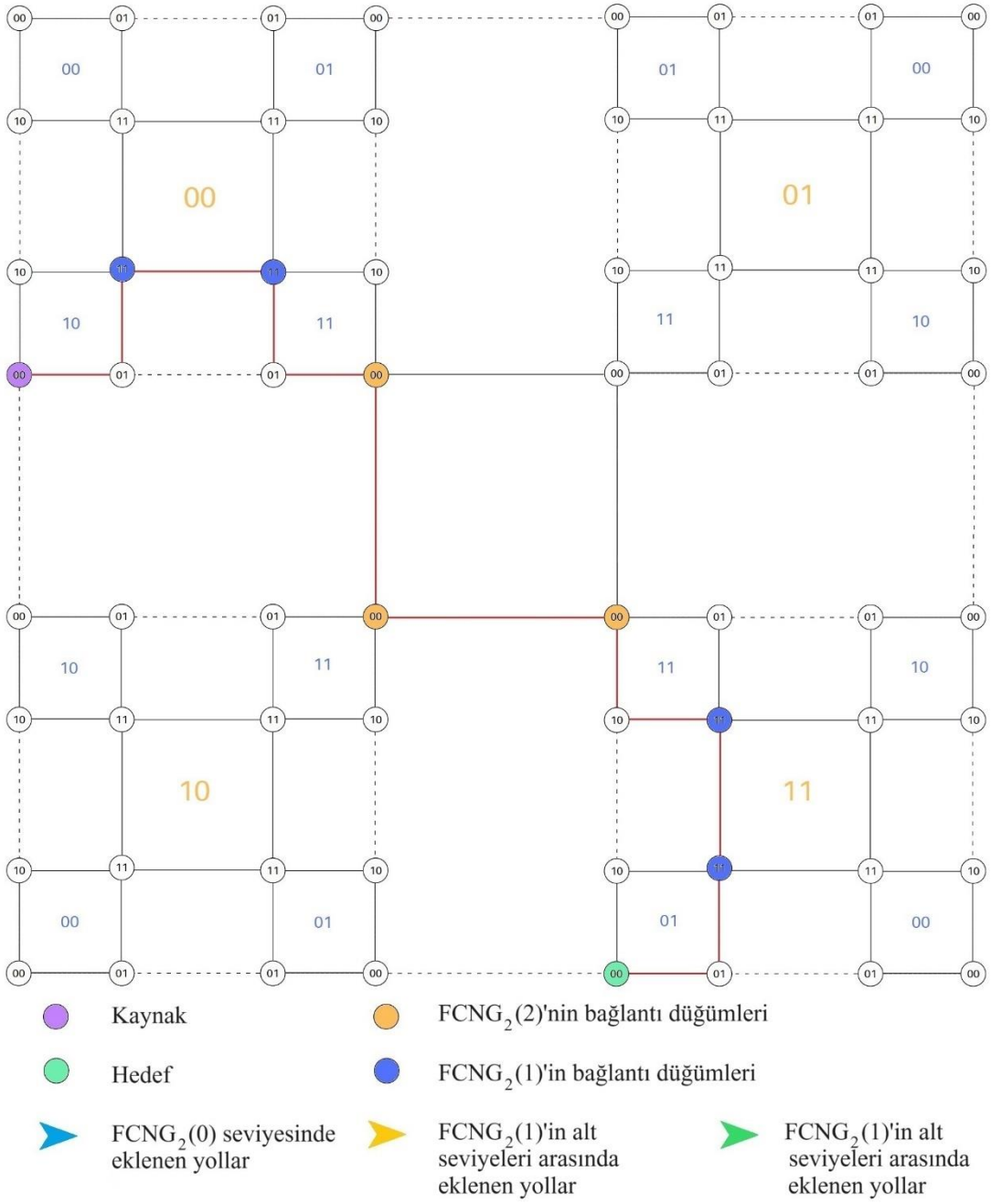
Routing(11 01 11, 11 01 00, 0, *P*, *MSG*)

P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 11 11, 11 01 11, 11 01 10}

P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 11 11, 11 01 11, 11 01 10, 11 01 00}

Sonuç olarak elde edilen yol aşağıdaki gibidir ve Şekil 4.8'de görülebilir:

P = {00 10 01, 00 10 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 11 11, 11 01 11, 11 01 10, 11 01 00}



Şekil 4.11. $FCNG_2(2)$ için kaynak düğüm V_S ={00 10 00} ve hedef düğüm V_D={11 01 00} arasındaki yol

4.4. EN KISA YOL

Kaynak ve hedef düğümler arasındaki en düşük maliyetli rotayı belirleyen yöntem, en kısa yol algoritması olarak bilinir. Çizgede en aza indirilmeye çalışılan maliyet değeri zaman, mesafe, düğüm sayısı gibi değerleri ifade edebilir. $FCNG_2(k)$ çizgesi için en kısa yol, iki düğüm arasındaki yolda geçen kenarların sayısı olarak hesaplanır. Fraktal yapıya sahip olan $FCNG_2(k)$ çizgesinde en kısa yolu hesaplamak için kullanılan algoritma özyinelemelidir.

4.4.1. En Kısa Yol Algoritması

$FCNG_2(k)$ için önerilen en kısa yol algoritması girdi olarak beş argüman alır. Bunlar $FCNG_2(k)$ çizgesinin haritası *M*, kaynak düğümü *S*, hedef düğümü *D*, çizgenin derecesi *k* ve çizgenin en kısa yolunu saklayan *P*'dir.

Algoritma ilk olarak kaynak ve hedef düğümlerin dış fraktalları arasındaki farkı hesaplar. Farkın 00 olması *S* ve *D* düğümlerinin aynı fraktal üzerinde olduğunu gösterir. Bu durumda, algoritma kendisini aynı çizge için çağırır ancak derecesi $k–1$ olduğundan algoritma bir alt problem için çağırılmış olur. Eğer fark 11 ise S ve *D* çapraz fraktallardadır ve en kısa yol merkezi bağlantı düğümünden geçer. 10 veya 01'lik bir fark, *S* ve *D* düğümlerinin sırasıyla üst üste veya yan yana olduğu anlamına gelir. Bu durumda minimum mesafeli bağlantı noktası, *S* ve *D*'nin dış yan kenar ve iç yan kenarlardan uzaklıklarına göre hesaplanır.

Hesaplanan bu bağlantı noktaları iki fraktal arasında köprü görevi gördüğünden, bu kesişim noktalarına göre problem orijinal problemden 1 derece daha düşük 2 alt probleme bölünebilir. Bu nedenle sorun, önce kaynaktan en yakın bağlantı düğümüne giden yolu hesaplayarak, ardından kaynak düğümün en yakın bağlantı düğümünden hedef düğümün en yakın bağlantı düğümüne kadar olan yolu hesaplayarak ve son olarak da kaynak düğümden en yakın bağlantı düğümüne kadar olan yolu hesaplayarak çözülebilir.

Temel durumda derece 0'dır, dolayısıyla algoritma yalnızca alt iki biti mesafe

açısından kontrol eder ve yolu bulur. Bu yolun uzunluğu 1 veya 2'dir.

$FCNG_2(k)$'nin kaynak ve hedef düğümleri arasındaki en kısa yolu hesaplamak için aşağıdaki adımlar uygulanır;

- Kaynak ve hedef düğümlerden dış fraktallar arasındaki iki bağlantı noktasına kadar olan minimum mesafeler hesaplanır ve düğümlerin konumlarına göre en yakın bağlantı noktası için iç bağlantı veya dış bağlantı olarak bir değer döner.
- $FCNG_2(k-1)$ için kaynaktan kaynağın bulunduğu fraktaldaki en yakın bağlantı düğümüne giden yol hesaplanır.
- $FCNG_2(k)$'nin kaynak fraktal bağlantı düğümünden hedef fraktal bağlantı düğümüne kadar olan yol hesaplanır.
- $FCNG_2(k-1)$ için hedeften hedef fraktaldaki en yakın bağlantı düğümüne kadar olan yol hesaplanır.

Kaynak ve hedef düğümler çapraz fraktallarda bulunuyorsa, ilk adım atlanır ve sonraki adımlar için en yakın bağlantı düğümleri yerine merkez bağlantı düğümleri alınır.

- *S*: Kaynak
- *D*: Hedef
- *SNJ*: Kaynağın bulunduğu $FCNG_1(k\text{-}1)$ boyutlu fraktalın hedefin olduğu fraktala olan bağlantı düğümlerinden hedefe en yakın olan düğüm
- *DNJ*: Hedefin bulunduğu $FCNG_1(k\text{-}1)$ boyutlu fraktalın kaynağın olduğu fraktala olan bağlantı düğümlerinden hedefe en yakın olan düğüm
- *k*: Derece
- *P*: Yol

Verilen değişkenlere göre en kısa yol algoritması aşağıdaki gibi bir sonuç üretmektedir;

$$ShortestPath(M,S,D,k,P) = \begin{cases} [ShortestPath(M,S,SNJ,k-1,P) \cup \\ SNJ \oplus 10(0 \dots 0) \cup DNJ \oplus 01(0 \dots 0) \cup \\ ShortestPath(M,DNJ,D,k-1,P)] \\ |||[ShortestPath(M,S,SNJ,k-1,P) \cup \\ SNJ \oplus 10(0 \dots 0) \cup \\ ShortestPath(M,DNJ,D,k-1,P)] \\ |||[ShortestPath(M,S,SNJ,k-1,P) \cup \\ SNJ \oplus 01(0 \dots 0) \cup \\ ShortestPath(M,DNJ,D,k-1,P)] \end{cases} \quad (4.27)$$

Algoritma 4.2. En Kısa Yol Algoritması

```
Data: S = kaynak, D = hedef, k=boyut, P = yol (başta boş)
Result: S'den D'ye en kısa yol
Function Shortest_Path(M, S, D, k, P):
  T = S and X = Dk ⊕ Sk
  if k > 0 then
    if X! = 00 then
      if X == 11 then
        Tk-1 = 11
      end
      if X == 01 then
        minP = Calculate_Min(M, S, D, k, 0)
        if minP == 1 then
          Tk-1 = 11
        end
        else
          Tk-1 = 01
        end
      end
      if X == 10 then
        minP = Calculate_Min(M, S, D, k, 1)
        if minP == 1 then
          Tk-1 = 11
```

```
          end
          else
            T_{k-1} = 10
          end
        end
        if k > 1 then
          for i = k-2 to 0 do
            T_i = 00
          end
        end
        P = Shortest_Path(M, S, T, k - 1, P)
        if X_L == 1 then
          T_k = T_k ⊕ 01 and add to path(P, T)
        end
        if X_H == 1 then
          T_k = T_k ⊕ 10 and add to path(P, T)
        end
        P = Shortest_Path(M, T, D, k - 1, P)
        return P
      end
      else
        return Shortest_Path(M, S, D, k - 1, P)
      end
    end
    else
      if X_L == 1 then
        T_k = T_k ⊕ 01 and add to path(P,T)
      end
      if X_H == 1 then
        T_k = T_k ⊕ 10 and add to path(P,T)
      end
      return P
    end
end function
```

Örnek 4.3

$FCNG_2(2)$'nin kaynak düğümü, hedef düğümü ve derecesinin sırasıyla 001000, 111010 ve 2 olduğunu varsayalım. Algoritma 4.2'nin ilk çağırılması Shortest_Path(001000, 111010, 2, *P*) şeklinde olacaktır. *P* oluşturulacak yoldur ve girdi olarak ilk çağrıda boş bir düğüm etiketi dizisi şeklinde algoritmaya verilir. Algoritma adım adım çalıştırılırsa bu örnek için aşağıdaki sonuç elde edilir;

Shortest_Path(*M*, 11 10 10, 2, *P*)

/* Kaynak ve kaynağın en yakın olduğu $FCNG_2(1)$ çizgesinin iç bağlantı düğümü arasındaki en kısa yolu hesapla

Shortest_Path(*M*, 00 00 11, 00 11 00, 1, *P*)

Shortest_Path(*M*, 00 00 11, 00 00 11, 0, *P*)

P = {00 01 11}

P = {00 01 11, 00 11 11}

Shortest_Path(*M*, 001111, 001100, 0, *P*)

P = {00 01 11, 00 11 11, 00 11 10}

P = {00 01 11, 00 11 11, 00 11 10, 00 11 00}

/* Kaynak ve hedefin bulunduğu $FCNG_2(1)$ çizgelerinin iç bağlantı düğümleri arasındaki yolu hesapla

P = {00 01 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00}

P = {00 01 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00}

/* Hedefin en yakın olduğu $FCNG_2(1)$ çizgesinin iç bağlantı düğümü ile hedef arasındaki yolu hesapla

ShortestPath(*M*, 11 11 00, 11 10 10, 1, *P*)

/* *S* ve *D* düğümleri yan yana fraktallarda olduğu için *t* değeri 0

minP = calculateMin(*M*, 11 11 00, 11 10 10, 1, 0)

/*0 dış bağlantı düğümü anlamına gelmektedir

minP = 0

Shortest_Path(*M*, 11 11 00, 11 11 01, 0, *P*)

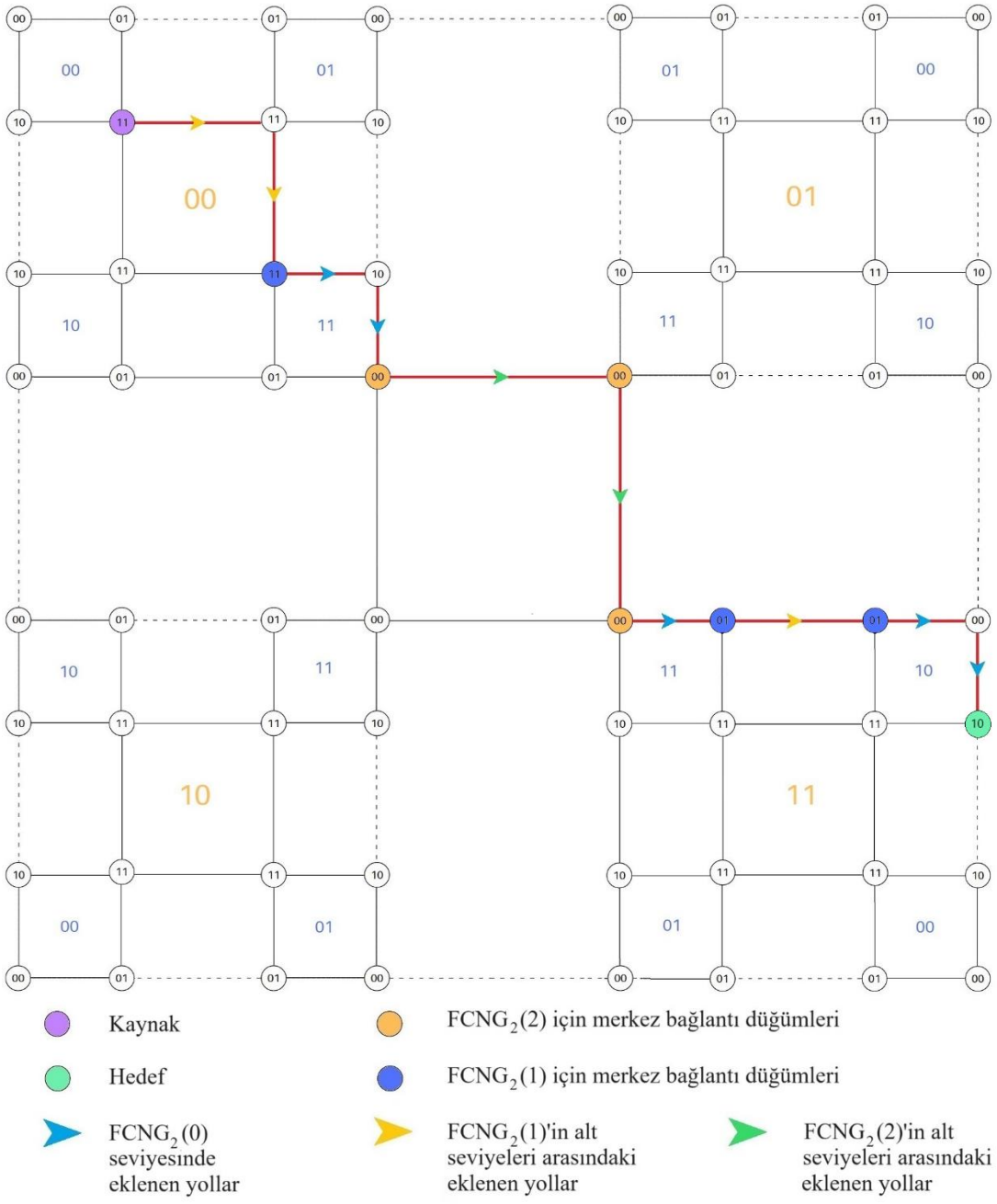
P = {00 01 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01}

P = {00 01 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 10 01}

ShortestPath(*M*, 11 10 01, 11 10 10, 0, *P*)

P = {00 01 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 10 01, 11 10 00}

P = {00 01 11, 00 11 11, 00 11 10, 00 11 00, 01 11 00, 11 11 00, 11 11 01, 11 10 01, 11 10 00, 11 10 10}



Şekil 4.12. $FCNG_2(2)$ için kaynak düğüm V_S ={00 00 11} ve hedef düğüm V_D={11 10 10} arasındaki en kısa yol

4.4.2. Haritalama Algoritması

En kısa yol algoritması $FCNG_2(k)$ üzerindeki etiketlerin konumlarının bulunduğu bir harita kullanır. $FCNG_2(k)$ çizgesinin haritasını bulmak için, bu çizgenin etiketlerini, düğümlerin konumuna karşılık gelen 2 boyutlu bir dizi olarak döndüren bir eşleme algoritması geliştirdik. Haritalama algoritması, çizgenin derecesi *k,* fraktalların ayna ekseni olarak da *i* ve *j*'yi girdi olarak alır. Algoritma verilen değerlere göre $FCNG_2(k)$'nin haritasını hesaplar ve döndürür. Haritalama algoritması özyinelemeli bir algoritmadır çünkü $FCNG_2(k)$ doğası gereği fraktal yapıda olan bir çizgedir. FCNG(k) çizgesi, $k > 1$ olmak üzere dört FCNG($k - 1$) kullanılarak oluşturulur. Dolayısıyla bu algoritma kendisini dört kez bir alt dereceye sahip çizge için özyinelemeli olarak çağırır.

$FCNG_2(k)$'nin haritası boyut 0'a eşitse [00 01; 10 11] olur. Boyut 0'dan büyükse algoritmada dört durum vardır. Bunlar [00 01; 10 11] temel durum haritası, y ekseni üzerinde yansıtılmış temel durum haritası [01 00; 11 10], x ekseni üzerinde yansıtılmış temel durum haritası [10 11; 00 01] ve köşegen boyunca yansıtılmış temel durum haritası [11 10; 01 00] şeklindedir.

$$\mathrm{FCNG}_2(k) = \begin{cases} [00 \cup \mathrm{FCNG}_2(k-1), 01 \cup \mathrm{FCNG}_2{}'(k-1) \\ 10 \cup \mathrm{FCNG}_2{}''(k-1), 11 \cup \mathrm{FCNG}_2{}'''(k-1)] \end{cases} \quad (4.28)$$

Eşitlik 4.28'de FCNG(k −1) temel alt çizgedir, FCNG'(k −1) y ekseni üzerinde yansıtılan, FCNG"(k − 1) x ekseni üzerinde yansıtılan ve FCNG'"(k − 1) çapraz eksene göre yansıtılan (hem x hem de y eksenine göre yansıtılan) temel alt çizgelerdir. 4 tane *k-1* boyutunda alt problemi çağırdığı ve bu problemlerin sonuçlarını sabit zamanda birleştirdiği için algoritmanın çalışma zamanı $\Theta(2^{2k})$'dır.

Örnek 4.4

$FCNG_2(0)$ çizgesi, $FCNG_2(1)$ çizgesinin bir alt problemidir. $FCNG_2(1)$'i oluşturmak için öncelikle yansıtılmış çizgeler olan $FCNG_2$'(0), $FCNG_2$"(0) ve $FCNG_2$'"(0) çizgeleri hesaplanmalıdır.

$$\mathrm{FCNG}_2\,(0)\;=\;[00\;01;\;10\;11] \tag{4.29}$$

$$\mathrm{FCNG}_2{}'\,(0)\;=\;[01\;00;\;11\;10] \tag{4.30}$$

$$\mathrm{FCNG}_2{}''\,(0)\;=\;[10\;11;\;00\;01] \tag{4.31}$$

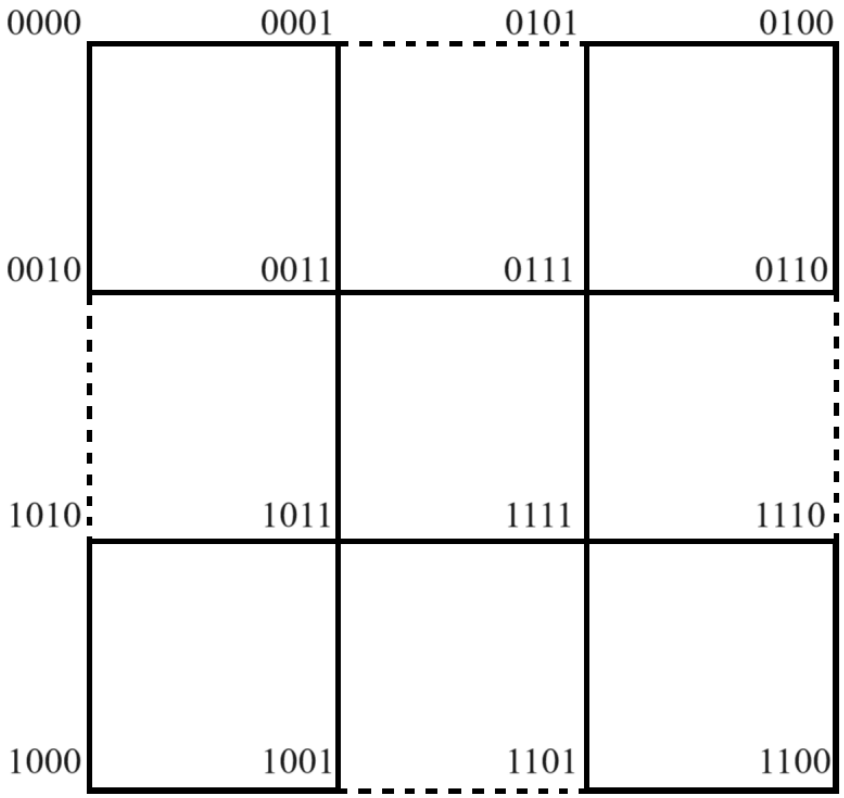
$$\mathrm{FCNG}_2{}'''(0)\;=\;[11\;10;\;01\;00] \tag{4.32}$$

$$\mathrm{FCNG}_2(1)=\begin{cases}[00\cup\mathrm{FCNG}_2(0),01\cup\mathrm{FCNG}_2'(0);\\ 10\cup\mathrm{FCNG}_2{}''(0),11\cup\mathrm{FCNG}_2{}'''(0)]\end{cases} \tag{4.33}$$

$$\mathrm{FCNG}_2(1)=\begin{cases}[00\cup[00\;01;10\;11],01\cup 01\;\cup\;[01\;00;\;11\;10];\\ \quad 10\;[10\;11;00\;01],11\;\cup\;[11\;10;\;01\;00]]\end{cases} \tag{4.34}$$

$$\mathrm{FCNG}_2(1)=\begin{cases}[[0000\;0001;0010\;0011],[0101\;0100;0111\;0110];\\ [1010\;1011;\;1000\;1001],[1111\;1110;\;1101\;1100]]\end{cases} \tag{4.35}$$

$$\mathrm{FCNG}_2(1)=\begin{cases}[0000\;0001\;0101\;0100;0010\;0011\;0111\;0110;\\ 1010\;1011\;1111\;1110;\;1000\;1001\;1101\;1100]\end{cases} \tag{4.36}$$



Şekil 4.13. Örnek 4.4'te FCNG$_2$(1) için hesaplanan harita

Algoritma 4.3. Haritalama algoritması

Data: k, i, j
Result: M: k derece $FCNG_2(k)$ çizgesi için harita (2 boyutlu etiket dizisi)
Function Map (k, i, j):
    if k > 0 **then**
        $M(1: 2^k, 1: 2^k) = ij \,||\, Map(k-1,0,0)$
        $M(1: 2^k, 2^k+1: 2^{k+1}) = i\neg j \,||\, Map(k - 1,0,1)$
        $M(2^k+1: 2^{k+1},1: 2^k) = \neg ij \,||\, Map(k - 1,1,0)$
        $M(2^k+1: 2^{k+1},2^k+1: 2^{k+1}) = \neg i\neg j \,||\, Map(k-1,1,1)$
        return M
    end
    if i == 0 and j == 0 **then**
        M=[00 01; 10 11]
    end
    if i == 1 and j == 0 **then**
        M=[01 00; 11 10]
    end
    if i == 0 and j == 1 **then**
        M=[10 11; 00 01]
    end
    if i == 1 and j == 1 **then**
        M=[11 10; 01 00]
    end
    return M
End Function

4.4.3. Minimum Bağlantı Noktasını Hesaplama Algoritması

Calculate_Min, $FCNG_2(k)$'nın iki fraktalı arasındaki en kısa bağlantıyı bulan algoritmadır. Algoritma yalnızca fraktallar üst üste veya yan yanaysa çağrılır, çapraz fraktallar için çağırılmaz. Bunun nedeni çapraz fraktallar için en kısa yolun iç bağlantı noktasından geçmesidir.

Bu algoritma girdi olarak beş argüman alır ve iç veya dış bağlantıyı temsil eden bir değer döndürür. *M,* verilen çizgenin haritasıdır, *S* ve *D* kaynak ve hedef

düğümlerdir, *k* çizgenin derecesidir ve *t* kaynak ve hedef fraktalların birbirlerine göre konumudur. Fraktallar yan yana ise *t*'nin değeri 0, üst üste ise *t*'nin değeri 1 olarak alınır.

Algoritma, harita üzerinde kaynak ve hedef düğümlerin konumlarını bulur ve fraktalların göreceli konumlarına göre Procedure_1 ve Procedure_2 adı verilen ve benzer şekilde çalışan alt algoritmaları çağırır. Bu algoritmalar, düğümlerin fraktallar arası bağlantıya paralel en iç veya en dış kenara olan mesafelerini hesaplar ve iç veya dış bağlantıyı temsil eden minimum mesafe bağlantı değerini döndürür.

Algoritma 4.4. Minimum mesafeye sahip bağlantı noktası bulma algoritması

```
Data: M, S, D,k, t
Result: Verilen düğümlerin minimum mesafeye sahip
        bağlantıdüğümünün konumu (0/1)
Function Calculate_Min(M, S, D, k, t):
  n = Harita M'nin boyutu
  for i = 1 to n do
    for j = 1 to n do
      if M(i, j) == S then
        SI = [i j]
      end
      if M(i, j) == D then
        DI = [i j]
      end
    end
  end
  if t == 1 then
    return Procedure_1(SI,DI,n,k)
  end
  else
    return Procedure_2(SI,DI,n,k)
  end
end function
```

Örnek 4.5

Örnek 4.3'te kaynak düğümü 001000 ve hedef düğümü 111010 olan $FCNG_2(2)$ için en kısa yol algoritması çalıştırılmaktadır. Fraktal 11 alt probleminde en kısa yol 111100 ile 111010 arasında $FCNG_2(1)$ olarak hesaplanırken Calculate_Min fonksiyonu aşağıdaki şekilde çağrılır;

$$minP = Calculate\_Min(M, 111100, 111010, 1, 0) \tag{4.37}$$

1100 ve 1010 yan yana fraktallar olduğundan t değeri 0'a eşittir ve Prosedür_2 çağrılır. Fraktalın üst ve alt tarafının satır sayısı kaynak ve hedef düğümler için 2^k birimlik aralığa bağlı olarak bulunur. Bu örnekte üst taraf 5, alt taraf 6'dır. Daha sonra kaynak ve hedef düğümlerin minimum mesafeleri hesaplanır ve bu minimum değerler karşılaştırılarak fraktallar arasındaki en kısa yol bağlantısı bulunur. Kaynak düğüm için minimum mesafe yukarıya 0, hedef düğüm için minimum mesafe ise aşağıya 0'dır. Kaynağın ve hedefin minimum mesafeleri aynıysa veya kaynağın hesaplanan mesafesi daha azsa algoritma S(k−1)'in üst bitini alır.

$$S = \underbrace{11}_{S(2)}\ \underbrace{11}_{S(1)}\ \underbrace{00}_{S(0)} \tag{4.38}$$

S(0)'ın üst biti 0 olduğundan dış bağlantı, S(1)'in 11'e eşit olduğu 1. derecedeki fraktallar arasındaki en kısa yol için seçilir.

Algoritma 4.5. Üst üste fraktalların minimum mesafe bağlantı noktasını hesaplamak için Calculate_Min'in alt algoritması

```
Data: SI, DI, n, k
Result: Üst üste binmiş fraktalların iç veya dış birleşme
        noktası
Function Procedure1(SI, DI,n,k):
  for i = 1 to n increase by 2^k do
    if SI[2] < i then
      u = i-1
      l = i-2^k
    end
  end
  if (u - SI[2]) <2^k then
    minS = u - SI[2]
  end
  else
    minS = SI[2] - l
  end
  if (u - DI[2]) <2^k then
    minD = u - DI[2]
  end
  else
    minD = DI[2] - l
  end
  if minS ≤ min D then
    return S(k-1)(lower bit)
    end
  else
    return D(k-1)(lower bit)
  end
end function
```

Algoritma 4.6. Yan yana fraktalların minimum mesafe bağlantı noktasını hesaplamak için Calculate_Min'in alt algoritması

```
Data: SI, DI, n, k
Result: Yan yana fraktalların iç veya dış birleşme noktası
Function Procedure2(SI, DI,n,k):
  for i = 1 to n increase by 2^k do
    if SI[1] < i then
      u = i-1
      l = i-2^k
    end
  end
  if (u - SI[1]) <2^k then
    minS = u - SI[1]
  end
  else
    minS = SI[1] - l
  end
  if (u - DI[1]) <2^k then
    minD = u - DI[1]
  end
  else
    minD = DI[1] - l
  end
  if minS ≤ minD then
    return S(k-1)(upper bit)
    end
  else
    return D(k-1)(upper bit)
  end
end function
```

Açıklama 4.2

Algoritma 4.2, kaynak ile hedef arasındaki en kısa yolun hesaplanmasına yönelik özyinelemeli bir algoritmadır. *k* dereceli bir çizge için çalıştırılan algoritma, *k* – 1

derecesine sahip 2 alt problem için kendisini . Özyinelemeli çağrıdan önce, kaynak ve hedef fraktallar arasındaki 2^k zaman alan minimum mesafe bağlantı noktasının hesaplanması için Algoritma 4.4 çalıştırılır. Dolayısıyla Algoritma 4.2'nin ilişkisel denklemi Eşitlik 4.39'daki gibidir.

$$T(k) = 2T(k-1) + \Theta(2^k) \tag{4.39}$$

Algoritma 4.2'nin zaman karmaşıklığı tümevarım yöntemi yardımıyla hesaplanabilmektedir. Bu formülde T(k−1)'i yerine konulduğunda;

$$T(k-1) = 2T(k-2) + \Theta(2^{k-1}) \tag{4.40}$$

$$\begin{aligned} T(k) &= 2\left(2T(k-2) + \Theta\left(2^{k-1}\right)\right) + \Theta\left(2^k\right) \\ &= 2^2T(k-2) + 2\Theta\left(2^{k-1}\right) + \Theta\left(2^k\right) \\ &= 2^2T(k-2) + 2\Theta\left(2^k\right) \end{aligned} \tag{4.41}$$

h = 1, 2, ..., k olduğunda bu özyinelemeli hesap genelleştirilirse;

$$T(k) = 2^hT(k-h) + h\Theta\left(2^k\right) \tag{4.42}$$

$h = k$ için;

$$T(k) = 2^kT(k-h) + k\Theta\left(2^k\right) = 2^k\Theta(1) + k\Theta\left(2^k\right) \tag{4.43}$$

Algoritma 4.2'nin zaman karmaşıklığı $\Theta((k+1)2^k)$ olup başlangıç değeri $T(0) = \Theta(1)$'dır.

4.4. ALGORİTMALARIN ZAMALARININ KARŞILAŞTIRILMASI

FCNG için önerilen yönlendirme algoritması, haritalama algoritması ve en kısa yol algoritması Python kullanarak uygulanıp, algoritmaların çalışma zamanları karşılaştırıldı.

Çizelge 4.6. $FCNG_2(2)$, $FCNG_2(3)$ ve $FCNG_2(4)$ için örnek veriler.

$FCNG_2(2)$			$FCNG_2(3)$			$FCNG_2(4)$		
#	**Kaynak Düğüm**	**Hedef Düğüm**	**#**	**Kaynak Düğüm**	**Hedef Düğüm**	**#**	**Kaynak Düğüm**	**Hedef Düğüm**
1	000011	111010	11	00001010	01010101	21	0100011110	1101010011
2	010011	110010	12	00011111	11100111	22	1110101000	1110111101
3	001011	000010	13	10011110	10101011	23	1110000111	1000111100
4	101100	010111	14	01001000	10111110	24	1010001011	0000111001
5	100001	101111	15	00001001	10111010	25	1001100011	1001110010
6	111000	101010	16	10111110	01101100	26	0011001100	0010110001
7	011011	101100	17	11001111	10010001	27	0001101001	0000010010
8	100100	001101	18	01010100	00101001	28	0101100110	0111100011
9	110001	001011	19	00110101	10000110	29	1111100011	1110101000
10	001011	010011	20	01110010	01010111	30	1100101011	1111101101

Çizelge 4.7. $FCNG_2(5)$ ve $FCNG_2(6)$ için örnek veriler.

$FCNG_2(5)$			$FCNG_2(6)$		
#	**Kaynak Düğüm**	**Hedef Düğüm**	**#**	**Kaynak Düğüm**	**Hedef Düğüm**
31	000000001111	011010000101	41	11100100100001	11101111100110
32	001111010111	011001110010	42	11110110110101	10011010011000
33	010000100010	010100001101	43	10010010110100	10110100001000
34	000010011111	011011000001	44	11010000111110	01111100100111
35	000010011000	100010101000	45	11010110010101	11101110110100
36	011100110100	010001000100	46	10000111010000	11111010100110
37	010101111011	110100000000	47	00010010101111	11000001000000
38	110010000000	100010101011	48	11000100010001	00010001010101
39	110000101001	100010101000	49	00101001011111	01000110011110
40	101010010010	100000000000	50	11010011100101	11110100010001

Algoritmaları çalıştırmak için oluşturulan örnek veriler, *k* değeri 2 ile 6 arasında olan $FCNG_2(k)$ çizgesi üzerinde uygulanmıştır. Çizgenin her boyutu için rastgele 10 farklı kaynak ve hedef düğümler oluşturulmuş ve bu 3 algoritmalar her bir örnek ile 100'er defa çalıştırılmıştır. Örnek düğümler Çizelge 4.6 ve 4.7'de verilmiştir.

Çizelge 4.8. 1-25 arası örnek veriler için elde edilen çalışma zamanları (ms).

#	**Haritalama**	**En Kısa Yol**	**Yönlendirme**
1	0.327919	0.120463	0.022235
2	0.289013	0.257521	0.023668
3	0.283198	0.093126	0.012543
4	0.290451	0.107701	0.025458
5	0.308905	0.018382	0.020213
6	0.302925	0.281527	0.030515
7	0.299637	0.119269	0.015008
8	0.328741	0.220671	0.016527
9	0.367341	0.151534	0.022573
10	0.283945	0.178912	0.020132
11	1.429312	0.313616	0.020969
12	1.334732	0.306873	0.017867
13	1.998775	0.439582	0.015929
14	2.308846	0.556154	0.018179
15	2.439668	1.098254	0.022531
16	2.324781	0.536191	0.010173
17	2.323096	0.561299	0.018198
18	2.432089	1.059818	0.022533
19	2.342303	1.082757	0.022404
20	2.165058	0.972445	0.021448
21	5.919628	0.016968	0.00946
22	6.062906	2.13346	0.016813
23	6.081488	1.084111	0.014665
24	6.055496	2.192941	0.020957
25	6.22947	3.225799	0.022449

Çizelge 4.9. 26-50 arası örnek veriler için elde edilen çalışma zamanları (ms).

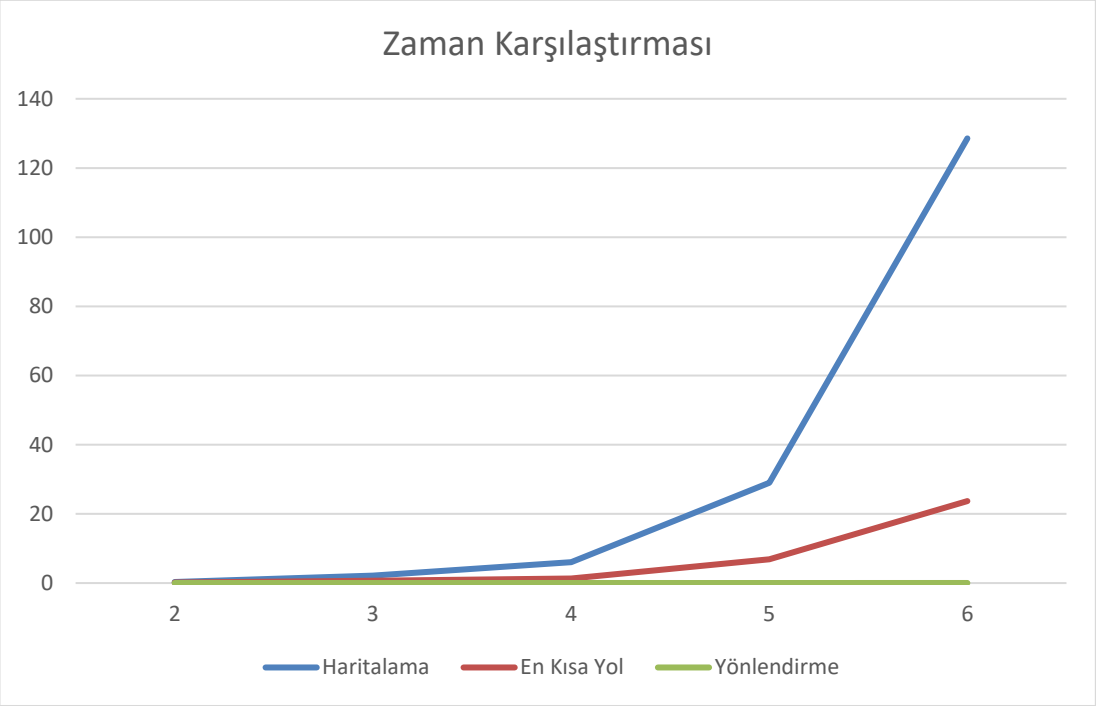
#	**Haritalama**	**En Kısa Yol**	**Yönlendirme**
26	6.116502	0.027401	0.016375
27	5.908053	1.111107	0.02399
28	6.062503	1.081831	0.008969
29	6.080532	1.081512	0.008876
30	5.84748	1.069651	0.008833
31	29.70436	4.717009	0.008285
32	31.62022	9.79382	0.021484
33	26.75101	4.047372	0.016661
34	26.59266	4.159613	0.018268
35	34.42853	10.89735	0.023506
36	26.41651	8.218827	0.024278
37	26.74636	4.268715	0.02403
38	32.46372	10.348	0.020819
39	28.62996	0.018244	0.00459
40	26.27756	12.35448	0.023754
41	125.0068	17.95828	0.008831
42	142.9207	20.33767	0.022366
43	125.7763	35.21635	0.0246
44	124.1588	35.25007	0.015597
45	131.8625	55.24062	0.021956
46	125.2957	18.1023	0.023177
47	125.0163	17.6537	0.020094
48	125.0221	17.87727	0.008247
49	125.738	0.021422	0.005305
50	134.6639	19.33029	0.023742

Çizelge 4.8 ve Çizelge 4.9'da çalıştırılan algoritmaların her bir örnek için 100'er defa çalıştırılması sonucunda elde edilen ortalama değerler ms cinsinden verilmiştir.

Çizelge 4.10. Yönlendirme, haritalama ve en kısa yol algoritmalarının her boyut için ortalama çalışma zamanları (ms).

Çizge Boyutu k $FCNG_2(k)$	Haritalama	En Kısa Yol	Yönlendirme
2	0.308208	0.154911	0.015
3	2.109866	0.692699	0.014
4	6.036406	1.302478	0.015
5	28.96309	6.882343	0.019
6	128.5461	23.6988	0.017

Şekil 4.14. Yönlendirme, haritalama ve en kısa yol algoritmalarının her boyut için ortalama çalışma zamanları (ms).



$FCNG_2(k)$ çizgesinde haritalama ve en kısa yol algoritmalarının çalışma zamanları çizgenin boyutu arttıkça artarken yönlendirme algoritmasının çalışma zamanı boyut arttıkça çok küçük farklar ile hafif bir artış görülmektedir.

BÖLÜM 5

SONUÇ VE DEĞERLENDİRME

Yapılan bu tezde, hiperküp ağı türevi olarak tanımlayabileceğimiz iki farklı ağ çeşidi ele alınmıştır. Bunlardan ilki olan Bağlı Kare Ağ Çizgeleri (CSNG) hiperküp özelliği göstermektedir ve aynı zamanda 2B örgü çizgesinin bir alt çizgesidir. Diğer üzerinde çalışılan çizge Fraktal Kübik Ağ Çizgeleri (FCNG) ise fraktal bir yapıya sahiptir ve 2 boyutlu bir çizge olarak kare örgü ağ çizgesinin bir alt kümesidir.

CSNG(k, m)'nin en önemli özelliği bir hiperküp varyantı olmasıdır. Bu çalışmanın odak noktası CSNG(k, m)'yi bir hiperküp varyantı olarak ele almak ve önemli problemlere çözüm içeren algoritmalar geliştirmektir. Dolayısıyla bu çizge için elde edilen bu algoritmalar klasik örgü algoritmalarından çok hiperküp algoritmalarına benzemektedir. İlk algoritma CSNG için dinamik programlama kullanarak alternatif Hamiltonian yolunu bulmaya yöneliktir. Bu algoritmanın çalışma zamanı $O(2^{k+m+2})$'dir. İkinci algoritma CSNG(k, m) için düğüm etiketlerinin eşlenmesini hesaplar ve çalışma süresi $O(2^{k+m+2})$'dir. Algoritma 3.3, zaman karmaşıklığı $O(\max(2^{k+1} - 1, 2^{m+1} - 1))$ olan bir tek noktaya yayın yönlendirme algoritmasıdır.

Paralel ve dağıtık mimari kullanımı, işlemci sayısına göre çalışma zamanında performans iyileştirmeleri sağlar. Algoritma 3.2'nin paralel olarak uygulanmasında üç farklı durum vardır. Bu üç durum için çalışma zamanı hesaplamalarının en geneli $O(n + 2^{k+m+2-n})$'ye eşittir. Paralel olarak uygulanan Algoritma 3.3'ün çalışma zamanı, CSNG(k, m) çizgesinde adres uzunluğunun $k + m + 2$ olduğu durumda $O(2^k + 2^m)$'dir. CSNG için yayın algoritmaları, hiperküp yayın algoritmalarına benzer bir strateji ile kolayca elde edilebilir. İki boyutlu ağların özel bir durumu için, iyi bilinen hiperküp algoritmaları kullanılarak farklı bir yaklaşım sunulmuştur.

Diğer hiperküp özelliği taşıyan FCNG çizgesi, fraktallar yardımıyla geliştirilen

yeni bir bilgisayar mimarisi yapısıdır. İlk olarak FCNG için yeni topolojik özellikler elde edilmiştir. Bu şekilde FCNG ve diğer benzer ağların bazı önemli topolojik özellikleri karşılaştırılabilecektir. Algoritma 4.1'de bu ağ için tek noktaya yayın yönlendirme sürecine yönelik yeni bir strateji önerilmektedir. FCNG'deki en kısa yol problemini çözmek için, [3]'deki haritalama algoritmasının revize edilmiş bir versiyonu ve yönlendirme algoritmasının benzer bir versiyonu kullanılmıştır. En Kısa Yol Algoritması ve Yönlendirme Algoritması özyinelemeli olup benzer mantıkla çalışır. Her iki algoritma da problemi 1 derece düşük olan 2 alt probleme böler ve bu alt problemlerin çözümlerini birleştirerek ana sonucu bulur. İki algoritma arasındaki temel fark bu 2 alt problem arasındaki yolun belirlenmesidir. Yönlendirme algoritmasında fraktallar arasındaki bağlantı noktası her zaman merkezi bağlantı noktası olarak seçilirken, en kısa yol algoritmasında fraktallar arasındaki bağlantı noktası en kısa yola göre iç veya dış bağlantı noktası olarak seçilir. Bu yüzden calculateMin fonksiyonu, en kısa yol için bağlantı düğümünü hesaplamakta kullanılır. Okuyucuların kolay takibi için, calculateMin algoritması ile birlikte iki alt algoritma verilmiştir. Yönlendirme Algoritmasının çalışma süresi $\Theta(2^k)$'dır. Haritalama algoritmasının çalışma süresi $\Theta(2^{2k})$'dır. En kısa yol algoritmasının çalışma süresini hesaplamak için, computeMin algoritmasının ve iki prosedürün bilinmesi gerekir. En kısa yol algoritmasının çalışma süresi $\Theta((k+1)2^k)$ olarak bulunur.

2B örgü ağları ve bu ağların alt kümesi olan FCNG ağları üzerinde geliştirilen en kısa yol algoritmalarının çalışma süresi aynı olsa da bu ağlar aynı boyutlarda olduğunda FCNG ağının oluşturulması daha az maliyet ile gerçekleştirilebilmektedir. Bu iki çizgeyi aynı boyutta yapmak için 2B Kare Örgü(m) ağında m değerinin 2^{k+1} olduğunu varsayarsak Örgü çizgesinin maliyeti $2(m^2 - m) = 2(2^{2k+2} - 2^{k+1})$, $k > 1$ olduğunda $4(2^{k+2} - 2)$'den daha büyüktür.

KAYNAKLAR

1. Dally, W. J. and Towles, B., "Principles and Practices of Interconnection Network", IEEE TRANSACTIONS ON COMPUTERS, ***Morgan Kaufmann Publishers Inc.***, San Francisco, (2004).

2. Selçuk, B., "Connected Square Network Graphs", ***Universal Journal Of Mathematics And Applications***, 5 (2): 57–63 (2022).

3. Karci, A. and Selçuk, B., "A new hypercube variant: Fractal Cubic Network Graph", ***Engineering Science And Technology, An International Journal***, 18 (1): 32–41 (2015).

4. Hennessy, J. L. and Patterson, D. A., "Computer Architecture: A Quantitative Approach", San Francisco, CA, USA, (2012).

5. Pinkston, T. M. and Duato, J., "Appendix F interconnection networks", .

6. Nielsen, F., "Topology of Interconnection Networks", 63–97 (2016).

7. Harary, F., Hayes, J. P., and Wu, H. J., "A survey of the theory of hypercube graphs", ***Computers & Mathematics With Applications***, 15 (4): 277–289 (1988).

8. Atta, S. and Sen, G., "A new variant of the p-hub location problem with a ring backbone network for content placement in VoD services", ***Computers & Industrial Engineering***, 159: 107432 (2021).

9. Eydi, A., Khaleghi, A., and Barzegar, K., "Ring hierarchical hub network design problem: Exact and heuristic solution methods", ***EURO Journal On Transportation And Logistics***, 11: 100096 (2022).

10. Fu, T., Li, C., Wu, L., and Zou, L., "A specific type of irregular ring-and-hub network structure and the average shortest distance of its rings", ***Heliyon***, 8 (11): e11470 (2022).

11. Dai, Q., "Exploration of bifurcation and stability in a class of fractional-order super-double-ring neural network with two shared neurons and multiple delays", ***Chaos, Solitons & Fractals***, 168: 113185 (2023).

12. Yu, J., Chen, W., Leng, J., Wang, C., and Yi, Z., "Weight matrix as a switch between line attractor and plane attractor of ring neural networks", ***Neurocomputing***, 521: 181–188 (2023).

13. Storm, P. J., Kager, W., Mandjes, M., and Borst, S., "Stability of a stochastic ring network", ***Performance Evaluation***, 162: 102355 (2023).

14. Yang, X. H., Chen, G., Sun, B., Chen, S. Y., and Wang, W. L., "Bus transport network model with ideal n-depth clique network topology", ***Physica A: Statistical Mechanics And Its Applications***, 390 (23–24): 4660–4672 (2011).

15. Fadigas, I. S. and Pereira, H. B. B., "A network approach based on cliques", ***Physica A: Statistical Mechanics And Its Applications***, 392 (10): 2576–2587 (2013).

16. Hao, F., Park, D. S., Min, G., Jeong, Y. S., and Park, J. H., "K-Cliques mining in dynamic social networks based on triadic formal concept analysis", ***Neurocomputing***, 209: 57–66 (2016).

17. Wang, L., Lin, F. V., Cole, M., and Zhang, Z., "Learning Clique Subgraphs in Structural Brain Network Classification with Application to Crystallized Cognition", ***NeuroImage***, 225: 117493 (2021).

18. Oliveira, D. A., Senna, V. de, and Pereira, H. B. de B., "Indices of textual cohesion by lexical repetition based on semantic networks of cliques", ***Expert Systems With Applications***, 237: 121580 (2024).

19. Sobehart, L. A., Martínez Alcalá, S., Chacoma, A., and Zanette, D. H., "Structural properties of random networks of cliques", ***Physica A: Statistical Mechanics And Its Applications***, 624: 128998 (2023).

20. Chai, Y. and Zeng, X. J., "Regional condition-aware hybrid routing protocol for hybrid wireless mesh network", ***Computer Networks***, 148: 120–128 (2019).

21. Inam, O., Al Khanjari, S., and Vanderbauwhede, W., "Shortest Path Routing Algorithm for Hierarchical Interconnection Network-on-Chip", ***Procedia Computer Science***, 56 (1): 409–414 (2015).

22. Onaizah, A. N., Xia, Y., Hussain, K., and Mohamed, A., "Optimized shortest path algorithm for on-chip board processor in large scale networks", ***Optik***, 271: 170151 (2022).

23. Masdari, M., Qasem, S. N., and Pai, H. T., "Optimizing Network-on-Chip using metaheuristic algorithms: A comprehensive survey", ***Microprocessors And Microsystems***, 103: 104970 (2023).

24. Ji, N., Zhou, X., and Yang, Y., "A high-performance fully adaptive routing based on software defined network-on-chip", ***Microelectronics Journal***, 141: 105950 (2023).

25. Gogoi, A., Ghoshal, B., and Manna, K., "Fault-aware routing approach for mesh-based Network-on-Chip architecture", ***Integration***, 93: 102043 (2023).

26. Bani-Mohammad, S. and Ababneh, I., "On the performance of non-contiguous allocation for common communication patterns in 2D mesh-connected multicomputers", ***Simulation Modelling Practice And Theory***, 32: 155–165 (2013).

27. Ma, C., Zhao, Q., Li, G., Deng, L., and Wang, G., "A deadlock-free physical mapping method on the many-core neural network chip", ***Neurocomputing***, 401: 327–337 (2020).

28. Amin, W., Hussain, F., and Anjum, S., "IHPSA: An improved bio-inspired hybrid optimization algorithm for task mapping in Network on Chip", ***Microprocessors And Microsystems***, 90: 104493 (2022).

29. Manzoor, M., Mir, R. N., and Hakim, N. ud din, "PAAD (Partially adaptive and deterministic routing): A deadlock free congestion aware hybrid routing for 2D mesh network-on-chips.", ***Microprocessors And Microsystems***, 92: 104551 (2022).

30. Al-Shammari, M. M., Haque, A., and Rahman, M. M. H., "Midimew Connected Torus Network for Next Generation Massively Parallel Computer System", ***Procedia Computer Science***, 179: 590–597 (2021).

31. Kodama, C., Terai, M., Noda, A. T., Yamada, Y., Satoh, M., Seiki, T., Iga, S. I., Yashiro, H., Tomita, H., and Minami, K., "Scalable rank-mapping algorithm for an icosahedral grid system on the massive parallel computer with a 3-D torus network", ***Parallel Computing***, 40 (8): 362–373 (2014).

32. Mukosey, A., Semenov, A., and Tretiakov, A., "Graph based routing algorithm for torus topology and its evaluation for the Angara interconnect", ***Journal Of Parallel And Distributed Computing***, 183: 104765 (2024).

33. Wang, Y. G., Du, H. M., and Shen, X. B., "Virtual channel load-balanced routing algorithm for Torus networks", ***The Journal Of China Universities Of Posts And Telecommunications***, 18 (1): 70–76 (2011).

34. Wang, Y. G., Du, H. M., and Shen, X. B., "Topological properties and routing algorithm for semi-diagonal torus networks", ***The Journal Of China Universities Of Posts And Telecommunications***, 18 (5): 64–70 (2011).

35. Abd El-Baky, M. A., "A tree-based algorithm for multicasting in 2D torus networks", ***Egyptian Informatics Journal***, 16 (1): 45–53 (2015).

36. Wang, N. C. and Hung, Y. P., "Multicast communication in wormhole-routed 2D torus networks with hamiltonian cycle model", ***Journal Of Systems Architecture***, 55 (1): 70–78 (2009).

37. Hayes, J., Mudge, T. N., and Stout, Q. F., "Architecture of a Hypercube Supercomputer", (1986).

38. Nieminen, J., Peltola, M., Ruotsalainen, P., and Mieminen, J., "On graphs like hypercubes", ***Tsukuba Journal Of Mathematics***, 32 (1): 37–48 (2008).

39. Chae, G. B., Palmer, E. M., and Robinson, R. W., "Counting labeled general cubic graphs", ***Discrete Mathematics***, 307 (23): 2979–2992 (2007).

40. Mao, W. and Nicol, D. M., "On k-ary n-cubes: theory and applications", ***Discrete Applied Mathematics***, 129 (1): 171–193 (2003).

41. El-Amawy, A. and Latifi, S., "Properties and performance of folded hypercubes", ***IEEE Transactions On Parallel And Distributed Systems***, 2 (1): 31–42 (1991).

42. Chang, H.-Y. and Chen, R.-J., "Incrementally extensible folded hypercube graphs", (1998).

43. Dong, Q., Yang, X., Zhao, J., and Tang, Y. Y., "Embedding a family of disjoint 3D meshes into a crossed cube", ***Information Sciences***, 178 (11): 2396–2405 (2008).

44. Efe, K., "The crossed cube architecture for parallel computation", ***IEEE Transactions On Parallel And Distributed Systems***, 3 (4): 513–524 (1992).

45. Lai, C. J., Tsai, C. H., Hsu, H. C., and Li, T. K., "A dynamic programming algorithm for simulation of a multi-dimensional torus in a crossed cube", ***Information Sciences***, 180 (24): 5090–5100 (2010).

46. Abd-El-Barr, M. and Al-Somani, T. F., "Topological properties of hierarchical interconnection networks: A review and comparison", ***Journal Of Electrical And Computer Engineering***, 2011: (2011).

47. Ghose, K. and Desai, K. R., "Hierarchical cubic networks", ***IEEE Transactions On Parallel And Distributed Systems***, 6 (4): 427–435 (1995).

48. Karci, A., "Hierarchical extended fibonacci cubes", ***Iranian Journal Of Science & Technology, Transaction B, Engineering***, 29 (B1): 117–125 (2005).

49. Karci, A., "Hierarchic graphs based on the fibonacci numbers", ***Istanbul University Journal Of Electrical & Electronics Engineering***, 7 (1): 345–365 (2007).

50. Selçuk, B. and Karci, A., "Connected Cubic Network Graph", ***Engineering Science And Technology, An International Journal***, 20 (3): 934–943 (2017).

51. Wilson, R. J., "Introduction to Graph Theory", 4. Ed., ***Longman Group Ltd***, (1996).

52. Phisutthangkoon, N. and Werapun, J., "Shortest-path routing for optimal all-to-all personalized-exchange embedding on hierarchical hypercube networks", ***Journal Of Parallel And Distributed Computing***, 150: 139–154 (2021).

53. Fu, T., Li, C., Wu, L., and Zou, L., "A specific type of irregular ring-and-hub network structure and the average shortest distance of its rings", ***Heliyon***, 8 (11): e11470 (2022).

54. Lai, C. N., "Constructing all shortest node-disjoint paths in torus networks", ***Journal Of Parallel And Distributed Computing***, 75: 123–132 (2015).

55. Manfrin, M., Birattari, M., Stützle, T., and Dorigo, M., "Parallel Ant Colony Optimization for the Traveling Salesman Problem", ***Interface***, 4150 (9): 224–234 (2006).

56. Shaheen, A., Sleit, A., and Al-Sharaeh, S., "Chemical Reaction Optimization for Traveling Salesman Problem Over a Hypercube Interconnection Network", (2019).

57. Shaheen, A., Sleit, A., and Al-Sharaeh, S., "Travelling Salesman Problem Solution Based-on Grey Wolf Algorithm over Hypercube Interconnection Network", ***Modern Applied Science***, 12 (8): 142 (2018).

58. Paler, A., Zulehner, A., and Wille, R., "NISQ circuit compilation is the travelling salesman problem on a torus", ***Quantum Science And Technology***, 6 (2): (2021).

59. Benni, N. S. and Manvi, S. S., "Channel Allocation Scheme to Mitigate Interference in 5G Backhaul Wireless Mesh Networks", (2022).

60. Kozierok, C. M., "The TCP IP Guide A Comprehensive, Illustrated Internet Protocols Reference", 1. Ed., ***No Starch Press***, (2005).

61. Grama, A., Gupta, A., Karypis, G., and Kumar, V., "Introduction to Parallel Computing", 2. Ed., ***Addison Wesley***, (2003).

62. Maurer, A. and Tixeuil, S., "Byzantine broadcast with fixed disjoint paths", ***Journal Of Parallel And Distributed Computing***, 74 (11): 3153–3160 (2014).

63. Samuylov, A., Moltchanov, D., Kovalchukov, R., Gaydamaka, A., Pyattaev, A., and Koucheryavy, Y., "GAR: Gradient assisted routing for topology self-organization in dynamic mesh networks", ***Computer Communications***, 190: 10–23 (2022).

64. De Marco, G. and Vaccaro, U., "Broadcasting in hypercubes and star graphs with dynamic faults", ***Information Processing Letters***, 66 (6): 321–326 (1998).

65. Ye, H. P., Xiao, W. J., and Wu, J. Y., "Broadcasting on the BSN-hypercube network", (2009).

66. Phisutthangkoon, N. and Werapun, J., "Optimal ATAPE task scheduling on reconfigurable and partitionable hierarchical hypercube networks", ***Parallel Computing***, 111: 102923 (2022).

67. Al-Dubai, A. Y. and Ould-Khaoua, M., "On the Design of Scalable Pipelined Broadcasting for Mesh Networks", (2002).

68. Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C., "Introduction to Algorithms", 4. Ed., ***The MIT Press***, Cambridge, Massachusetts, (2022).

69. Wang, X. and Mou, Z. G., "A divide-and-conquer method of solving tridiagonal systems on hypercube massively parallel computers", ***Proceedings Of The Third IEEE Symposium On Parallel And Distributed Processing***, 810–817 (1991).

70. Mayr, E. W. and Werchner, R., "Divide-and-conquer algorithms on the hypercube", ***Theoretical Computer Science***, 162 (2): 283–296 (1996).

71. Lor, S., Shen, H., and Maheshwari, P., "Divide-and-conquer mapping of parallel programs onto hypercube computers", ***Journal Of Systems Architecture***, 43 (6–7): 373–390 (1997).

72. Karci, A., "Generalized parallel divide and conquer on 3D mesh and torus", ***Journal Of Systems Architecture***, 51 (5): 281–295 (2005).

73. Valero-Garcia, M., Gonzalez, A., Diaz De Cerio, L., and Royo, D., "Divide-and-conquer algorithms on two-dimensional meshes", (1998).

74. Panja, R. and Vadhiyar, S. S., "HyPar: A divide-and-conquer model for hybrid CPU–GPU graph processing", ***Journal Of Parallel And Distributed Computing***, 132: 8–20 (2019).

75. Tripathy, P. K., Dash, R. K., and Tripathy, C. R., "A dynamic programming approach for layout optimization of interconnection networks", ***Engineering Science And Technology, An International Journal***, 18 (3): 374–384 (2015).

76. Crichigno, J., Khoury, J., Wu, M. Y., and and Shu, W., "A Dynamic Programming Approach for Routing in Wireless Mesh Networks", (2008).

77. Hou, Y., Wang, C.-M., Ku, C.-Y., and Hsu, L.-H., "Optimal Processor Mapping for Linear-Complement Communication on Hypercubes", ***IEEE Transactions On Parallel And Distributed Systems***, 12 (5): 514–527 (2001).

78. Strate, S. A. and Wainwright, R. L., "Parallelization of the Dynamic Programming Algorithm for the Matrix Chain Product on a Hypercube", (1990).

79. Goldman, A. and Trystram, D., "An efficient parallel algorithm for solving the Knapsack problem on hypercubes", ***Journal Of Parallel And Distributed Computing***, 64 (11): 1213–1222 (2004).

80. Itai, A., Papadimitriou, C. H., and Szwarcfiter, J. L., "Hamilton Paths in Grid Graphs", ***SIAM Journal On Computing***, 11 (4): 676–686 (1982).

81. Dong Chen, S., Shen, H., and Topor, R., "An efficient algorithm for constructing Hamiltonian paths in meshes", ***Parallel Computing***, 28 (9): 1293–1305 (2002).

82. Keshavarz-Kohjerdi, F., Bagheri, A., and Asgharian-Sardroud, A., "A linear-time algorithm for the longest path problem in rectangular grid graphs", ***Discrete Applied Mathematics***, 160 (3): 210–217 (2012).

83. Wang, Z., Ligang, H., Jinhui, W., Shuqin, G., and Wuchen, W., "Comparison research between XY and odd-even routing algorithm of a 2-dimension 3x3 mesh topology network-on-chip", (2009).

84. Chen, G.-H. and Huang, I.-I.-L., "Cube-Connected Modules: a Family of Cubic Networks", ***IEEE***, 0-8186–6507 (94): (1994).

85. Duh, D. R., Chen, G. H., and Hsu, D. F., "Combinatorial properties of generalized hypercube graphs", ***Information Processing Letters***, 57 (1): 41–45 (1996).

86. Efe, K., "A variation on the hypercube with lower diameter", ***IEEE Transactions On Computers***, 40 (11): 1312–1316 (1991).

87. Mahafzah, B. A., Alshraideh, M., Abu-Kabeer, T. M., Ahmad, E. F., and Hamad, N. A., "The Optical Chained-Cubic Tree interconnection network: Topological structure and properties", ***Computers & Electrical Engineering***, 38 (2): 330–345 (2012).

88. Saad, Y. and Schultz, M. H., "Topological properties of hypercubes", ***IEEE Transactions On Computers***, 37 (7): 867–872 (1988).

89. Guo, L. and Boruzanlı Ekinci, G., "Connectivity and super connectivity of folded hypercube-like networks", ***Theoretical Computer Science***, 976: 114151 (2023).

90. Zhou, W., Fan, J., Jia, X., and Zhang, S., "The spined cube: A new hypercube variant with smaller diameter", ***Information Processing Letters***, 111 (12): 561–567 (2011).

91. Yang, D. W., Xu, Z., Feng, Y. Q., and Lee, J., "Symmetric property and edge-disjoint Hamiltonian cycles of the spined cube", ***Applied Mathematics And Computation***, 452: 128075 (2023).

92. Bernardi, M. P., Bondioli, C., and Quaglia, M., "On Graph Directed Constructions Of Fractals", ***Electronic Notes In Discrete Mathematics***, 26: 9–14 (2006).

93. Brown, J. I., Hickman, C. A., and Nowakowski, R. J., "The independence fractal of a graph", ***Journal Of Combinatorial Theory, Series B***, 87 (2): 209–230 (2003).

94. Ejov, V., Filar, J. A., Lucas, S. K., and Zograf, P., "Clustering of spectra and fractals of regular graphs", ***Journal Of Mathematical Analysis And Applications***, 333 (1): 236–246 (2007).

95. Leung, A. Y. T., "Dynamic substructure method for elastic fractal structures", ***Computers & Structures***, 89 (3–4): 302–315 (2011).

96. Meier, J. and Reiter, C. A., "Fractal representations of Cayley graphs", ***Computers & Graphics***, 20 (1): 163–170 (1996).

97. Komjáthy, J. and Simon, K., "Generating hierarchial scale-free graphs from fractals", ***Chaos, Solitons & Fractals***, 44 (8): 651–666 (2011).

ÖZGEÇMİŞ

Ayşe Nur ALTINTAŞ TANKÜL ilkokula Sakarya'da başlamış, ilkokul ve ortaokulu tamamladıktan sonra liseyi Süleyman Demirel Çankırı Fen Lisesinde bitirmiştir. 2013 yılında burslu olarak kazandığı İhsan Doğramacı Bilkent Üniversitesi Bilgisayar Mühendisliği Bölümü'nde lisans derecesini almıştır. Yüksek lisans derecesini 2018 yılında Karabük Üniversitesi'nde Fen Bilimleri Enstitüsü'nde aldı. Karabük Üniversitesi Bilgisayar Mühendisliği Bölümü'nde araştırma görevlisi olarak çalışmaktadır. Birçok farklı derste yardımcı öğretim üyesi olarak görev almıştır. Ayrıca Bilgisayar Mühendisliği Bölümde çeşitli idari görevleri ve öğrenci danışmanlıkları bulunmaktadır. Araştırma alanları arasında ağlar, çizge teorisi, algoritmalar ve hesaplama teorisi yer almaktadır.