T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇİZGE ALGORİTMALARI VE ÇİZGE BÖLMELEME

Ali KARCI

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

ELAZIĞ
1998

T.C.
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇİZGE ALGORİTMALARI VE ÇİZGE BÖLMELEME

Ali KARCI

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

Bu tez, 25.08.1998... Tarihinde Aşağıda Belirtilen Jüri Tarafından Oybirliği/~~Oyçokluğu~~ ile Başarılı/~~Başarısız~~ Olarak Değerlendirilmiştir.

(imza)	(imza)	(imza)
	Yrd. Doç. Dr. Yalçın TATAR	Yrd. Doç. Dr. Erhan AKIN

Danışman
Y.Doç.Dr. Ahmet ARSLAN

ÖZET

Yüksek Lisans Tezi

Çizge Algoritmaları ve Çizge Bölmeleme

Ali KARCI

Fırat Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

1998, Sayfa : 100

Bu çalışmada paralel ortamlarda yük dengeleme şartlarını sağlayacak olan çizge bölmeleme işlemini ve bazı etkili çizge algoritmaların uygulamasını yapan bir yazılım geliştirilmiştir.

Yük dengeleme, paralel ortamlarda her işlemciye mümkünse eşit oranda iş yüklemek ve işlemciler arasındaki iletişimi minimum yapmak olarak tanımlanabilir.

Bu çalışmada ağırlıklı olarak çizge bölmeleme üzerinde durulmuştur. Çizge bölmeleme algoritmaları başlangıçta iki sınıfa ayrılabilir. Eğer çizgenin düğümlerinin koordinatları varsa uygulanan algoritmalar incelenmiştir. Çizgenin düğümlerinin koordinatları yoksa, bu durumda kullanılan algoritmalar incelenmiş ve lineer ve spektral algoritmaların uygulamaları yapılmıştır.

İncelenen yöntemler birbirleri ile kıyaslanarak avantajları ve dezavantajları verilmiştir.

ANAHTAR KELİMELER : Spektral çizge bölmeleme, gelişigüzel çizge bölmeleme, basit çizge indirgeme, çok seviyeli çizge bölmeleme, yük dengeleme, ağ bölmeleme, devre bölmeleme.

SUMMARY

Master Thesis

Graph Algorithms and Graph Partitioning

Ali KARCI

Fırat University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

1998, Pages : 100

In this work, graph algorithms and graph partitioning software, satisfied the load balancing conditions, has been developed.

The load balancing can be defined as each processor get equl work and transmission among processors is minimum.

In this work, graph partitioning haz been analyzed mostly. If vertices of graph have coordinate, then vertices of graph are sorted with respect to specified axis and they are partitioned into two parts. If they do not have coordinate, then Laplace matrix of graph is used to partition graph or graph is partitioned randomly.

The advantages and disadvantages of methods have been given by comparison of methods.

KEYWORDS : Spectral graph partitioning, random graph partitioning, simple graph reduction, multilevel graph partitioning, load balancing, network partitioning, circuit partitioning.

TEŞEKKÜR

Tez çalışması süresince bilgi ve tecrübelerinden yararlandığım Y.Doç.Dr. Ahmet ARSLAN Bey' e teşekkür ederim.

İÇİNDEKİLER

Sayfa

ŞEKİLLER LİSTESİ

Şekil no **Sayfa**

TABLOLAR LİSTESİ

Tablo no **Sayfa**

SİMGELER LİSTESİ

$\|.\|$	: Bir kümenin eleman sayısı.
$A=[a_{ij}]$	: Bitişiklik matrisi.
$ADJ(v)$	: v düğümünün bitişik düğümler kümesi.
$G=(V,E)$	: V ve E' den oluşan bir çizge.
Δ	: G çizgesindeki maksimum dereceli düğümün derecesi.
ΔE	: enerji farkı
$D=[d_{ij}]$	: Derece matrisi.
$d(v_i,v_j)$	: v_i düğümü ile v_j düğümü arasındaki en kısa yol.
$d(v_i)$	: v_i düğümünün derecesi.
E	: Ayrıtlar kümesi.
$E(t)$	: t anındaki sistemin enerji
ϕ	: Çizgenin izoperimetrik sayısı.
K_n	: n tane düğümü olan dolu çizge.
$K_{m,n}$	: İki parçalı ve dolu olan bir çizge.
$\kappa(G)$	: G çizgesinin parça sayısı.
$L=[l_{ij}]$	: Bir çizgenin Laplace matrisi.
λ	: Bir matrisin öz değeri.
nD	: n boyutlu uzay.
T	: G çizgesinden elde edilen minimum kapsar ağacı.
V	: düğümler kümesi.
$w(e_i)$	: e_i ayrıtın ağırlığı.
$w(v_i)$	: v_i düğümünün ağırlığı.
$\vec{x}$	: vektör.

1. GİRİŞ

Tek işlemcili ortamlarda problem çözümü ardışıl olarak yapılır. Matematiksel yönü çok yoğun olan problemlerin çözümü bu ortamlarda yapıldığında, işlemler istenilmeyen derecede fazla zaman almaktadır.

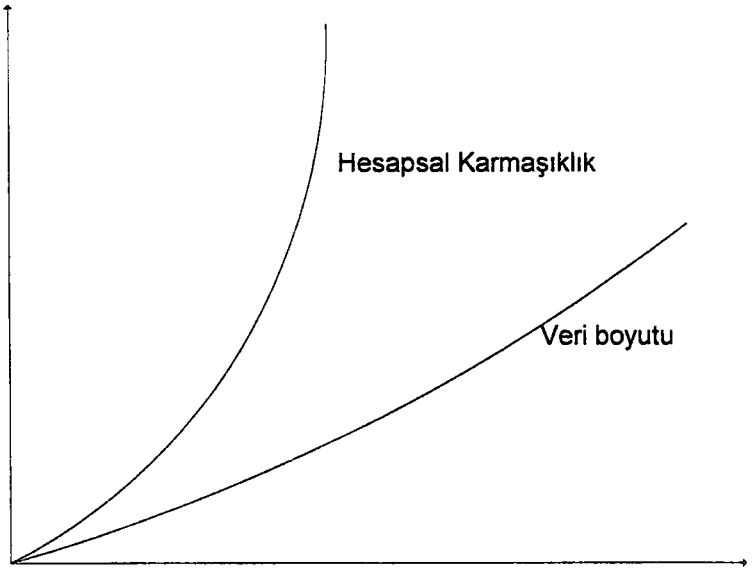
Yukarıda belirtilen problemin çözümü için paralel ortamlar kullanılabilir. Paralel ortamların özelliklerinden biri, bu tip yoğun işlemlerde iş bölümünün yapılmasıdır. Bu yolla işlemler paralel olarak yapıldığından zaman kaybının önüne geçilir.

Paralel ortamlarda iş bölümünün yapılması yollarından biri çizge bölmelemedir. Çizge bölmeleme işlemi, VLSI devre tasarımında [KIM, v.d., 1993], [SANCHIS, 1989], paralel ortamlarda yük dengeleme işleminde [GUPTA, 1997], [KARYPIS, v.d., 1996], mikro işlemci tasarımında, vb. gibi alanlarda kullanılmaktadır.

Paralel ortamlarda dikkatli bir yük paylaşımı yapılmazsa, bir veya birden fazla işlemci çok yoğun bir şekilde çalışırken, atıl durumda bekleyen işlemci veya işlemciler olabilecektir. Bu da yapılacak olan işlemler için fazladan zaman harcanması anlamına gelmektedir. Paralel ortamın amaçlarından biri olan zamanı en iyi şekilde kullanma hedefi yerine getirilmemiş olur. Bundan dolayı, paralel ortamlarda yük dengeleme konusu oldukça önem taşımaktadır.

Yük dengeleme işleminde dikkat edilmesi gereken ikinci nokta, problemin çözümü için işlemciler arasındaki iletişimin minimum olmasının sağlanmasıdır. İşlemciler arası iletişimin minimum olması için, işlemcilere dağıtılan işin işlemciler arası veri veya komut iletişimine mümkün olan en az oranda ihtiyaç duyulmalıdır. İş paylaşımı yapılırken buna dikkat edilmesi gerekmektedir. Bu iş için kullanılan yöntemlerin başında çizge bölmeleme gelmektedir.

Çizge bölmeleme problemi için günümüze kadar etkili bir algoritma geliştirilememiştir. Genellikle yaklaşımla çözüm yapılır ve bulunan çözüm bazen en iyi çözüm olabileceği gibi bazen de en iyi çözüme yaklaşık bir çözüm olabilir. Optimum



Şekil 1.1. NP-tamam tipinde olan bir algoritmanın veri boyutuna bağlı olarak hesapsal karmaşıklığının artışı

çözümü bulan yöntemler de vardır, fakat bu yöntemlerle çizge bölmeleme yapıldığında çok zaman kaybı olmaktadır. Eğer bu yöntemlerle çözüm yapılırsa, belki de verilen problemin ardışıl çözümü için harcanacak zamandan daha çok zaman harcanacaktır. Çizge bölmeleme işleminin amacı problemi çözmek değil problemin çözümü için harcanacak olan zamanın mümkünse minimum olmasını sağlamaktır.

Düğüm koordinatları olan çizgelerin bölmelenmesi daha kolay olmaktadır [LASZEWSKI, vd., 1993],[SPIELMAN, vd., 1996]. Düğümlerin eksenlere göre sıralanması ve bundan sonra iki eş parçaya bölünmesi ile bölmeleme işlemi yapılmış olur. Eğer düğümlerin koordinatları yoksa bu durumda başka yöntemlere başvurulmaktadır [SPIELMAN, vd., 1996], [LASZEWSKI, 1993], [KARYPIS, vd., 1996], [KARYPIS, vd., 1997]. Bölmeleme esnasında veya bölmeleme işleminden sonra çizgenin değişme ihtimali söz konusu ise bu durumda artımsal çizge bölmeleme yöntemi kullanılır [OU, vd., 1997].

Eğer çizge bölmeleme probleminin hesapsal karmaşıklığının mertebesi P-tamam tipinde bir algoritma geliştirilirse, bilgisayar mühendisliği uygulamalarında kullanılan NP-tamam olan

bütün algoritmalara P-tamam tipinde algoritmalar geliştirilir ki bu da bu dalda bir devrim niteliği taşımaktadır. P-tamam olan algoritmaların hesapsal karmaşıklığının mertebesi polinom tipinde olan bir fonksiyondur. Hesapsal karmaşıklığının mertebesi NP-tamam olan algoritmaların mertebesi eksponansiyeldir. Bundan dolayı problemin veri boyutu büyüdüğü zaman hesapsal karmaşıklığı daha hızlı büyür. Bilgisayarda yapılan algoritmalarda en önemli kriterlerden biri hız olduğuna göre bu demektir ki bu algoritmada veri boyutu büyüdüğü zaman çözümü bulmak daha çok zaman alır (Şekil 1.1).

Bu çalışmada daha çok çizge bölmeleme işlemi üzerinde durulacaktır. Bunun sebebi de çizge bölmeleme probleminin günümüze kadar çözümünü bulan etkili bir algoritmasının geliştirilememiş olmasıdır. Bunun yanında çizgenin analizini veya testini yapan birkaç algoritmanın da uygulaması yapılacaktır.

Bazı problemlerin çözümü için çizgelerin analizi veya testi doğrudan çözümü verir. Bunlara örnek olarak, çizgenin bağlılığı [THULASIRAMAN, v.d., 1992], çizgenin minimum kapsar ağacı [THULASIRAMAN, v.d., 1992], [MCHUGH, 1990], çizgenin iki parçalılığı [MCHUGH, 1990], düğümler arası en kısa yollar [GIBBONS, 1985], [THULASIRAMAN, v.d., 1992],[CORMEN, vd., 1991] gibi konular için etkili algoritmalar geliştirilmiştir.

Bu çalışmada çizge bölmeleme, çizge analizi ve testi yapan bir yazılım geliştirilmiştir. Etkili çizge algoritmalarından en kısa yol, iki parçalılık, minimum ağırlıklı kapsar ağaç ve çizgenin bağlılığını bulan algoritmaların birer uygulaması yapıldı. Çizge bölmeleme için iki yöntem kullanıldı; Lineer çizge bölmeleme yöntemi ve spektral çizge bölmeleme yöntemi. Bu iki algoritmanın çeşitli çizgeler üzerindeki uygulamalarının sonuçları alındı ve birbiriyle kıyaslaması yapıldı. Bu yazılım için Delphi 2.0 derleyicisi kullanılmıştır.

Klasik çizge algoritmalarından en kısa yol algoritmalarından Floyd' un bütün düğümler arası en kısa yolu bulan ve bu yolun hangi düğümlerden geçtiğini de bulan algoritmasının bir uygulamasını yapıldı [THULASIRAMAN, vd., 1992]. Minimum ağırlıklı kapsar ağacı bulan Prim algoritmasının

bir uygulamasını yapıldı. Çizgenin bağlılığı ve iki parçalı olup olmadığını test eden uygulamalar da yapıldı.

Tezin ikinci bölümünde, çizgeler ile ilgili Türkçe kaynak sıkıntısı olduğundan, çizgelerin temel kavramları hakkında kısa bilgiler verilmiştir. Üçüncü bölümde, çizge analizi ve testi yapan dört yöntemin teorisi ve algoritmaları verilmiştir. Dördüncü bölümde, çizge bölmelemede kullanılan önemli yöntemler anlatılmıştır. Beşinci bölümde, yapılan uygulamanın tanıtılmış ve seçilen çizgeler üzerindeki yapılan uygulamaların sonuçları verilmiştir. Aynı zamanda lineer ve spektral çizge bölmelemenin kıyası yapılmıştır. Altıncı bölümde, kullanılan yöntemler birbirleri ile kıyaslanarak iyi ve kötü yönleri tartışılmıştır. Ekler kısmında ise geliştirilen yazılımın kaynak kodu verilmiştir.

2. ÇİZGELERDE TEMEL KAVRAMLAR

Çizgeler, oluşlar ve bu oluşlar arasındaki ilişkilerin grafiksel olarak gösterilmesinde kullanılmaktadır. Bir çok bilimsel ve mühendislik problemleri bu yöntem ile ifade edilebilmektedirler (problemin çizge ile modellenmesi). Bu model üzerinde, geliştirilmiş etkili algoritmalar olduğu gibi yaklaşım ile problemlere çözüm bulan algoritmalar da geliştirilmiştir. Yaklaşım algoritmalarının geliştirilmesinin sebebi ise, bu tip problemlerin etkili algoritmasının günümüze kadar geliştirilememiş olmasıdır.

Çizge üzerinde yapılan işlemlerin algoritmalarının anlatımına geçmeden önce çizge kuramı ve çizgenin bilgisayar ortamında ifade edilmesi konularının açıklanması gerekmektedir.

2.1. Ayrıtlar, Düğümler ve Çizge Çeşitleri

Çizgeler, iki sonlu kümeden oluşurlar. Bunlardan biri sonlu düğümler kümesi, diğeri ise bu düğümler arasındaki ayrıtlar kümesidir. Genel olarak çizge $G=(V,E)$ ile gösterilir. $V=\{v_1, v_2, \ldots, v_n\}$ düğümler kümesidir ve $E=\{e_1, e_2, \ldots, e_m\}$ $(E \subseteq VxV)$ ayrıtlar kümesidir.

Eğer $e_i=(v_j,v_{j+1}) \in E$ ve $(v_{j+1},v_j) \notin E$ ise bu tip ayrıtlara yönlendirilmiş ayrıt denir ve $e_i=(v_j,v_{j+1})$ şeklinde gösterilir. Aynı zamanda v_j ve v_{j+1} düğümleri birbirine komşu (bitişik) düğümlerdir ve $ADJ(v_j)=v_{j+1}$, $ADJ(v_{j+1})=v_j$ olur. Eğer bir çizgenin ayrıtlarının hepsi yönlendirilmiş ise bu çizgeye yönlendirilmiş çizge denir. Eğer e_i gibi bir ayrıt yönlendirilmemiş ayrıt ise $e_i=(v_j,v_{j+1}) \in E$ ve $e_i=(v_{j+1},v_j) \in E$ olur. Yönlendirilmemiş ayrıtlar $e_i=\{v_j,v_{j+1}\}$ veya $e_i=\{v_{j+1},v_j\}$ şeklinde gösterilir. Eğer bir çizgenin bütün ayrıtları yönlendirilmemiş ise bu tip çizgelere yönlendirilmemiş çizge denir. Tez içerisinde, aksi belirtilmediği sürece yönlendirilmemiş çizge yerine çizge terimi

kullanılacaktır. $e_i=\{v_j,v_{j+1}\}$ ayrıtı için, v_j ve v_{j+1} düğümleri e_i ayrıtının uç noktalarıdır.

$e_i=\{v_j,v_{j+1}\}$ olsun. Eğer $v_j=v_{j+1}$ ise e_i'ye döngü (tek çevre) denir. $e_i=\{v_j,v_{j+1}\}$ ve $e_{i+1}=\{v_k,v_{k+1}\}$ olsun; eğer $v_k=v_j$, $v_{k+1}=v_{j+1}$ ve $v_k\neq v_{j+1}$, $v_{k+1}\neq v_j$ ise e_i ve e_{i+1} ayrıtlarına paralel (koşut bağlı) ayrıtlar denir. Uç noktaları farklı olan ayrıtlara tek ayrıt denir. Bir düğüme yalnız iki ayrıt bağlı ve bu ayrıtların öbür uç düğümleri birbirlerinden farklı ise bu ayrıtlara dizi bağlı ayrıtlar denir.

Bir çizgenin düğümler kümesinin eleman sayısı o çizgenin mertebesini verir. $G=(V,E)$ için eğer $|V|=n$ ise G çizgesinin mertebesi n' dir. v gibi bir düğüme giren ve çıkan ayrıt sayısına o düğümün derecesi (kertesi) denir ve $d(v)$ ile gösterilir.

$v_i\in V$ için eğer $d(v_i)=0$ ise v_i düğümü izole edilmiş bir düğümdür veya bu düğüme tek düğüm denir. Eğer $d(v_i)=1$ ise bu tip düğümlere pendant veya uç düğüm denir.

$G=(V,E)$, bir yönlendirilmemiş çizge olsun. Eğer $V=\emptyset$ veya $V\neq\emptyset$ ve $E=\emptyset$ ise bu tip çizgelere boş çizge denir ve $\forall v_i\in V$ için $d(v_i)=0$ olur. Bir adet tek düğümden veya tek çevreden, veya tek ayrıttan oluşan çizgelere ilkel çizge denir. Eğer bir çizge döngü ve paralel ayrıt içermiyorsa bu tip çizgelere basit çizge (yalın çizge) denir. Sadece çevreleri (döngüleri) içeren çizgelere sözde çizge denir.

Bir çizgede, bir düğüme bağlı olan bütün ayrıtların oluşturduğu kümeye, o düğümün tanımladığı çakışım kümesi denir.

Teorem 2.1. $G=(V,E)$, yönlendirilmemiş bir çizge olsun. G çizgesi için

$$\sum_{i=1}^{|V|} d(v_i)=2|E| \qquad (2.1)$$

olur.

İspat : Eğer $e_i\in E$ ise $e_i=\{ v_j, v_{j+1} \}$ olduğundan bir ayrıt iki ayrı düğümün derecelerini birer arttırır. Eğer ayrıt bir

döngü ise bu ayrıt bir düğümün derecesini iki arttırır. Bundan dolayı düğümlerin dereceleri toplanırken bir ayrıt iki kez sayılacağından dereceler toplamı ayrıt sayısının iki katı olur ■

Teorem 2.2. Bir çizgedeki derecesi tek olan düğüm sayısı her zaman çifttir.

İspat : Eğer tek dereceli düğümlerin dereceleri ve çift dereceli düğümlerin dereceleri ayrı ayrı toplanırsa

$$\sum_{i=1}^{|V|} d(v_i) = \sum_{\text{çift}} d(v_i) + \sum_{\text{tek}} d(v_k) \qquad (2.2)$$

elde edilir. (2.2) eşitliğinin sol tarafı çifttir. Eşitliğin sağ tarafındaki dereceleri çift olan düğümlerin derecelerinin toplamı da çift olur. Çift bir sayıdan çift bir sayı çıkarılırsa sonuç çift olur.

$$\sum_{\text{tek}} d(v_k) = \sum_{i=1}^{|V|} d(v_i) - \sum_{\text{çift}} d(v_i) \qquad (2.3)$$

(2.3) eşitliğinin sağ tarafının çift olabilmesi için çift sayıda tek dereceli düğümün derecelerinin toplanmış olması gerekir ■

Herhangi bir çizge için, eğer bütün düğümlerin dereceleri eşit ise bu tip çizgelere düzenli çizge denir. Eğer bir çizge paralel ayrıtlar içeriyorsa bu tip çizgelere çoğul çizge (multi-çizge) denir. Her türlü ayrıtı içeren çizgelere karmaşık çizge denir.

Tam olan bir çizgede her düğüm diğer bütün düğümler ile komşudur ve (2.4) bağıntısı v_i düğümünün komşu düğümler kümesini vermektedir.

$$\forall v_i \in V,\ \text{Adj}(v_i) = \{v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_{|V|}\} \qquad (2.4)$$

Düzenli bir çizgede, eğer $\forall v_i \in V$, $d(v_i)=r$ ise du düzenli çizgeye r-düzenli çizge denir.

Eğer $G=(V,E)$ çizgesinin düğümler kümesi $V=V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$ ve E' deki bütün ayrıtların bir ucu V_1' de ve diğer ucu V_2' de ise

bu çizgeye iki parçalı çizge denir. İki parçalı ve tam olan çizgeler $K_{m,n}$ şeklinde gösterilirler ($|V_1|=m$ ve $|V_2|=n$).

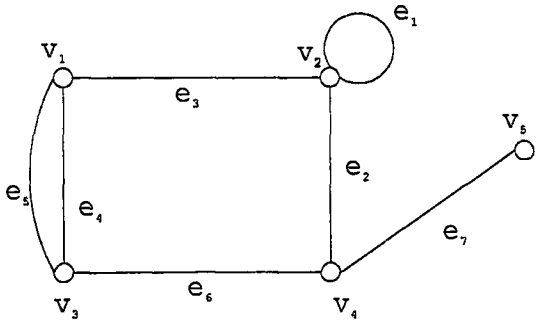
Eğer $G=(V,E)$ çizgesi k-parçalı ise V kümesi k tane öyle parçalara $V_1,\ldots,V_k$ bölünebilir ki, $V_i \cap V_j=\varnothing$ ve $i\neq j$. E içindeki bütün ayrıtların bir ucu V_i' de ve diğer ucu V_j' dedir, $i\neq j$. k-parçalı tam bir çizge, basit bir k-parçalı çizge olup $V=\{ V_1 , \ldots, V_k\}$, $\forall v_i \in V_r$, $\forall v_j \in V_s$, $r\neq s$, $1\leq r,s\leq k$, $\{v_i,v_j\}\in E$.

Örnek 2.1. Şekil 2.1' deki çizgede görüldüğü gibi $e_1=\{v_2,v_2\}$, $e_2=\{v_2,v_4\}$, $e_3=\{v_2,v_1\}$, $e_4=\{v_1,v_3\}$, $e_5=\{v_1,v_3\}$, $e_6=\{v_3,v_4\}$, $e_7=\{v_4,v_5\}$' dir. $e_1=\{v_2,v_2\}$ ve $v_2=v_2$ olduğundan e_1 ayrıtı bir döngüdür. e_4 ve e_5 ayrıtları paralel ayrıtlardır. Bu çizge bir basit çizge değildir, çünkü hem döngü ve hem de paralel ayrıtlar içermektedir. Aynı zamanda döngüden dolayı çoğul-çizge de değildir, bu karmaşık gelişi güzel bir çizgedir.

Ayrıtların dereceleri ise $d(v_1)=3$, $d(v_2)=4$, $d(v_3)=3$, $d(v_4)=3$, $d(v_5)=1$. Bu çizgede izole edilmiş düğüm yok ve bir tane uç düğüm vardır (v_5). Düğümlerin dereceleri birbirine eşit olmadığından bu çizge bir düzenli çizge değildir. Bazı düğümler arasında ayrıt olmadığından bu çizge aynı zamanda bir tam çizge de değildir ■

$G=(V,E)$, yönlendirilmemiş bir çizge olsun. $v_1,v_2\in V$ (v_1 ve v_2' nin farklı olması şart değildir)

$$v_1=v_i e_j v_{i+1} e_{j+1} v_{i+2} \ldots e_{j+n} v_{i+n}=v_2$$



Şekil 2.1. Her türlü ayrıtın var olduğu bir çizge.

bu değişen seriye yürüyüş (dolaşı) denir.

Bir yürüyüşün uzunluğu o yürüyüş içinde bulunan ayrıt sayısı ile ifade edilir. Eğer $v_1=v_2$ ve yürüyüşün uzunluğu sıfır ise bu yürüyüşe doğal yürüyüş denir. Eğer yürüyüşün uzunluğu sıfır değil ve $v_1=v_2$ ise bu yürüyüşe kapalı yürüyüş denir, aksi halde açık yürüyüş denir.

Eğer v_1-v_2 yürüyüşünde tekrar eden ayrıt yok ise bu yürüyüşe gezinti (gezi) denir. v_1-v_1 gezintisine çevre denir. Eğer bir yürüyüşün düğümleri birden fazla tekrar etmiyorsa, bu yürüyüşe yol denir. Kapalı v_1-v_1 yoluna devre denir.

Teorem 2.3. G=(V,E), yönlendirilmemiş bir çizge olsun ve $v_1,v_2\in V$, $v_1\neq v_2$ olsun. Eğer v_1' den v_2' ye bir gezi varsa v_1' den v_2' ye bir yol vardır.

İspat : $\{v_1,v_i\}$, $\{v_{i-1},v_{i+2}\}$, ..., $\{v_{i+n},v_2\}$ bir gezi olsun. Eğer bu gezi bir yol değilse, $\{v_1,v_i\},\{v_{i+1},v_{i+2}\},\ldots,\{v_{i+k-1},v_{i+k}\}$, $\{v_{i+k},v_{i+k+1}\},\ldots,\{v_{i+m-1},v_{i+m}\},\{v_{i+m},v_{i+m+1}\},\ldots,\{v_{i+n},v_2\}$ gibi bir gezi vardır, öyle ki $v_{i+k}=v_{i-m}$. $\{v_1,v_i\},\{v_{i+1},v_{i+2}\}$,..., $\{v_{i+k-1},v_{i+k}\}$, $\{v_{i-m},v_{i+m+1}\},\ldots,\{v_{i+n},v_2\}$ daha kısa bir gezi olur.

Yalın bir çizgenin her düğüm çifti birbirleriyle bitişik iseler bu tip çizgelere tam (dolu) çizge denir ve K_n ile gösterilir (n çizgenin mertebesidir). Dolu çizgenin ayrıt sayısı $|E|=n(n-1)/2$ olur ■

2.2. Alt Çizgeler ve Tamlayan (Tümleyen) Çizgeler

G=(V,E) bir çizge olsun. $G_2=(V_2,E_2)$ çizgesi G çizgesinin bir alt çizgesidir, eğer $E_2\subseteq E$ ve $V_2\subseteq V$. Eğer $E_2\subset E$ ise G_2 çizgesine G çizgesinin bir özalt çizgesi denir. Eğer $V_2=V$ ise G_2 çizgesine G çizgesinin bir kapsar alt çizgesi denir.

G=(V,E) yönlendirilmemiş bir çizge olsun. Eğer herhangi iki düğüm arasında yol varsa, bu çizge bağlı çizgedir veya başka bir deyişle her düğüm çifti arasında en az bir yol bulunan çizgelere bağlı çizge denir.

Bağlı olmayan çizgelere parçalı çizge denir. G gibi bir çizgenin parça sayısı $\kappa(G)$ ile gösterilir. Bir bağlı çizge için $\kappa(G)=1$ olur. Bir çizgenin kendi arasında bağlı olan ve olabildiğince çok sayıda ayrıtı içeren alt çizgelerinden her birine parça denir.

$G=(V,E)$ ve $G_2=(V_2,E_2)$ birer yönlendirilmemiş çizge olsunlar. Eğer $V_2 \subseteq V$, $E_2 \subseteq E$ ve $\forall (v_i,v_{i+1}) \in E_2$ ancak ve ancak $v_i, v_{i+1} \in V_2$ ise, G_2 çizgesine G çizgesinin indirgenmiş (düğüm-indirgenmiş) alt çizgesi denir. Diğer bir deyişle, eğer $\forall v_i, v_{i+1} \in V_2$ ise E' nin elemanı olan ve uç noktaları v_i ve v_{i+1} olan bütün ayrıtlar aynı zamanda E2' nin de elemanıdırlar.

Eğer G_2 çizgesi G çizgesinin bir alt çizgesi ise (G çizgesinde tek düğüm yok) V_2 içindeki bütün düğümler E_2' nin elemanlarının uç noktalarıdırlar. Bu durumda, E_2 tek olarak V_2' yi belirtebilir ve aynı zamanda G_2' yi de belirtebilir. Bu tip çizgelere ayrıt-indirgenmiş alt çizge denir. Aksi belirtilmediği sürece, indirgenmiş alt çizge terimi düğüm-indirgenmiş alt çizge teriminin yerine kullanılacaktır.

$K_n=(V,E_n)$ çizgesi bir tam çizge olsun. Eğer $(v_i,v_{i+1}) \in E_n$ ve $(v_i,v_{i+1}) \in E_2$ ancak ve ancak $(v_i,v_{i+1}) \notin E_1$ ise, $G_2=(V,E_2)$ çizgesine $G_1=(V,E_1)$ çizgesinin tamlayanı (tümleyeni) denir ve $E_1 \cup E_2 = E_n$. Diğer bir deyişle, G_2 çizgesindeki iki tane düğüm birbirine bitişiktir ancak ve ancak bu iki düğüm G çizgesinde birbirine bitişik değildir.

2.3. Çizgeler Üzerinde Tanımlanan İşlemler

$G_1=(V_1,E_1)$ ve $G_2=(V_2,E_2)$ birer çizge olsunlar. Bu çizgeler üzerinde tanımlanabilen işlemler birleşim, kesişim, fark, çembersel toplam, düğüm silme, ayrıt silme, büzüştürme gibi işlemlerdir.

İki çizgenin birleşimi

$G=G_1 \cup G_2=(V_1 \cup V_2, E_1 \cup E_2)$

İki çizgenin kesişimi

$G=G_1 \cap G_2=(V_1 \cap V_2, E_1 \cap E_2)$

İki çizgenin farkı

$G=G_1-G_2=(V_1-V_2, E_1-E_2)$

G çizgesi tekdüğüm içermez ve G' nin ayrıt kümesinin elemanları ya E_1' in elemanıdırlar ya da E_2' nin elemanladırlar, aynı anda her ikisinin elemanı olamazlar.

Yukarıda tanımlanan işlemlerden fark hariç diğerlerinin değişim özelliği vardır.

$G_1 \cup G_2 = G_2 \cup G_1$

$G_1 \cap G_2 = G_2 \cap G_1$

Düğüm silme : Eğer v_i G çizgesinin bir düğümü ise, $G-v_i$ çizgesi G çizgesinin indirgenmiş alt çizgesidir ve düğümler kümesi $V-v_i$' dir. $G-v_i$ çizgesi G çizgesinden v_i düğümü ve bu düğüme çakışık olan bütün ayrıtların silinmesi ile elde edilmiştir.

Ayrıt silme : Eğer e_i G çizgesinin bir ayrıtı ise, G-e çizgesi G çizgesinin ayrıt-indirgenmiş alt çizgesidir ve ayrıtlar kümesi $E-e_i$' dir. $G-e_i$ çizgesi G çizgesinden e_i ayrıtın silinmesi ile elde edilir. e_i ayrıtın çakışık olduğu düğümler silinmez.

Kısa devre yapma : v_i ve v_j gibi bir düğüm çiftinin yerine yeni tek bir düğüm koyup ve bu iki düğüme çakışık olan bütün ayrıtlar bu yeni düğüme çakışık hale getirilmesi işlemine kısa devre yapma denir.

Büzüştürme : e gibi bir ayrıtın silinip uç noktalarının kısa devre yapılmasıdır.

2.4. Çizgelerin Bilgisayar Ortamında İfade Edilmesi

Çizgeler çok değişik usullerle ifade edilebilirler. Her ifade etme tarzının avantajları ve dezavantajları vardır. Bi

Yukarıda tanımlanan iki matris dışında kullanılan başka matrislerde tanımlanabilmektedir. Her düğümün derecesini gösteren derece matrisi (bu matriste ise ana diyagonal dışındaki bütün elemanlar sıfır ve boyutları ise |V|x|V|)

$$D=[d_{ij}]=\begin{cases}d_i \text{ Eğer } d(v_i)=d_i \text{ ve } i=j \text{ ise}\\ 0 \text{ Eğer } i\neq j\end{cases} \qquad (2.7)$$

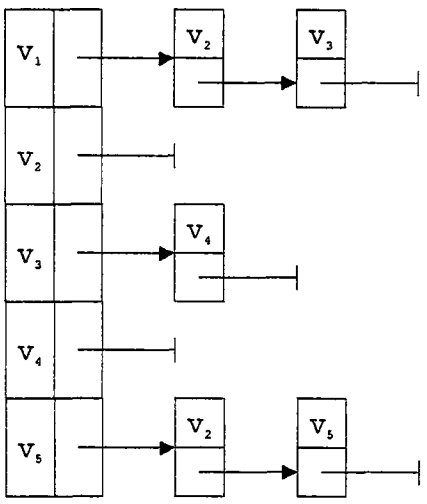
2.4.2. Çizgelerin dinamik olarak ifade edilmesi

Çizgelerin ayrıtlarının ifade edilmesinde kullanılan bitişiklik listesi kullanım olarak çok kompleks olmasına rağmen, hafıza kullanımı bakımından ve çalışma anında çizgenin ayrıt sayısının ve/veya düğüm sayısının değiştirilmesi imkanı sağlaması bakımından matris yöntemine göre daha iyidir [MCHUGH, 1990]. Çünkü bu yöntemde var olan düğümler ve ayrıtlar için hafızada yer ayrılır. Bu yöntemde daha çok kullanılan veri yapısı, bir düğümler kümesi vardır ve bu düğümlerin bitişik olduğu ayrıtlar listesi veya ayrıtlar listesi şeklinde basitçe tanımlanabilir. Bu yöntem için genel olarak üç tip veri yapısı kullanılmaktadır.

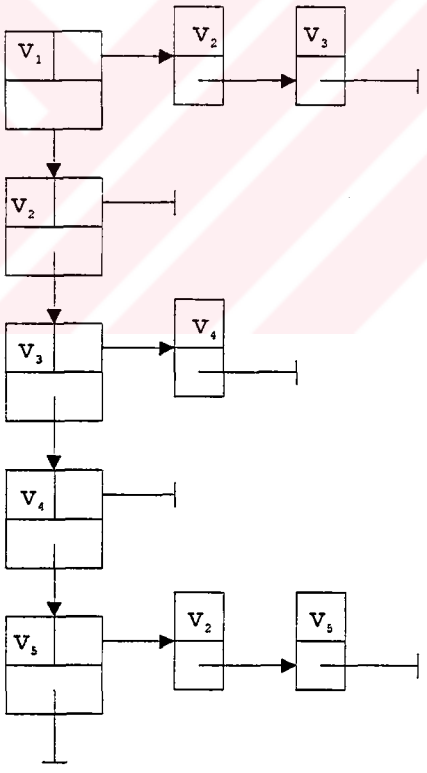
1. Lineer dizi.
2. Lineer bağlı liste.
3. Tamamen bağlı liste.

Her üç durumda da herhangi bir düğümün ADJ(v) kümesi için lineer bağlı liste kullanılır ve listenin başlık kısmı da v' yi temsil eden kısma kayıt edilir.

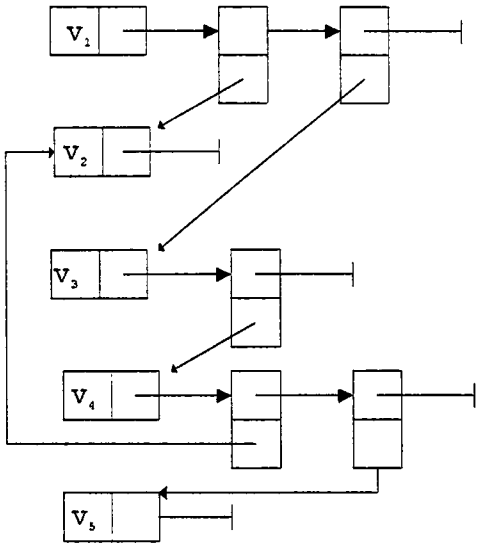
Lineer dizi için, her düğüm lineer dizinin bir elemanı olarak bağlı listenin başlık kısmını temsil eder (Şekil 2.2). Lineer bağlı liste organizasyonun da ise ADJ(v) kümesi v düğümünün başlık olduğu bağlı listeye kayıt edilir ve çizgenin düğümleri de bağlı bir listeye kayıt edilir (Şekil 2.3).



Şekil 2.2 : G çizgesi için lineer dizi



Şekil 2.3. G çizgesi için lineer bağlı liste.



Şekil 2.4. G çizgesi için tamamen bağlı liste.

Tamamen bağlı yapıda ise, çizgenin bir düğümü giriş işaretçisi olarak seçilir ve geriye kalan düğümler ise çizgenin ayrıtlarını taklit eder şekilde birbirine bağlanır. Eğer bir tane giriş işaretçisi yetersiz geliyorsa bu durumda birden fazla işaretçi kullanılabilir (Şekil 2.4).

Ayrıtların temsil edilmesi şu iki etkenden etkilenir.

1. Bitişiklik listesinin yönlendirilmiş çizgeyi mi yoksa yönlendirilmemiş çizgeyi mi temsil ettiği
2. Temsil edilen ayrıtların kendi uç noktaları tarafından paylaşılıp paylaşılmadığı

Yönlendirilmemiş çizge durumunda, $\{v_i, v_j\}$ ayrıtının uç noktaları simetrik bir yapının oluşmasına neden olurlar. Bundan dolayı her iki uç noktaya $ADJ(v_i)$ veya $ADJ(v_j)$ listesinden ulaşılabilir.

Doğal olarak, bitişiklik matrisinde olduğu gibi, her ayrıt için bağlı liste içinde yer ve işaretçi tutulur. Bu hafızanın kullanımını arttırır fakat işlem yapmada kolaylık sağlar.

Yönlendirilmiş çizgede ise (v_i, v_j) ayrıtı için $ADJ(v_i)$ bağlı listede tutulurken bu ayrıt için $ADJ(v_j)$ tutulmaz.

3. ÇİZGE ALGORİTMALARI

Çizgeler, bir çok pratik çalışmalarda kullanım alanı bulmaktadırlar. Bu tip çalışmaların ilk adımı, verilen problemin çizgesel özelliklerini ortaya çıkarmaktır. Genelde problemin çözümü için çizgenin ya özellikleri test edilir ya da çizgenin analizi yapılır. Çizgeler, günlük yaşamda ortaya çıkan karışık ve karmaşık problemlerin analizi sonucunda ortaya çıkabilirler. Bundan dolayı çizgelerin etkili olarak test edilmesi veya analiz edilmesi etkili algoritmaların oluşturulmasını gerektirmektedir.

3.1. En Kısa Yollar

G=(V,E) bağlı bir çizge olsun ve her ayrıtın ağırlığı da o yolun uzunluğunu gösteriyor olsun. v_i düğümünden v_j düğümüne doğru yönlendirilmiş ayrıtın uzunluğu $w(v_i,v_j)$ şeklinde gösterilsin. Eğer v_i düğümünden v_j düğümüne yol yoksa, $w(v_i,v_j)=\infty$ olur. Bir yolun uzunluğu, o yolu oluşturan ayrıtların uzunluklarının toplamına eşittir. İki düğüm arasında uzunluğu en az olan yola en kısa yol denir. Bir v_i-v_j yolunun uzunluğu v_i ile v_j düğümleri arasındaki uzaklığı verir ve $d(v_i,v_j)$ ile gösterilir.

Bu bölümde şu iki problem göz önüne alınacaktır;

1. G çizgesi içerisinde belli bir düğümden diğer bütün düğümlere olan uzaklıkların bulunması.
2. Sıralı düğüm ikilileri arasındaki en kısa yolun bulunması.

Bu iki problem bir çok optimizasyon probleminde ortaya çıkmaktadır.

3.1.1. Bir düğümden diğer düğümlere en kısa yollar

Yönlendirilmiş bir çizgede, belli bir düğümden diğer bütün düğümlere olan yolların en kısa şeklini bulan algoritmayı ilk olarak Dijkstra [BONDY, vd., 1976], [THULASIRAMAN, vd., 1992], [DEO, 1974], [CORMEN, vd., 1991], [GIBBONS, 1985] geliştirmişlerdir.

G=(V,E) çizgesi için, V düğümler kümesi ve E ayrıtlar kümesidir. S⊂V ve s∈S olsun ve $\overline{S}$ kümesi, S' nin V' ye göre tümleyeni olsun. Bu durumda $\overline{S} = V - S$ olur.

s' den $\overline{S}$' ye olan bütün yönlendirilmiş yollar arasında P:s,...,u,v uzunluğu minimum olan yol olsun. P yolunun uzunluğuna uzaklık denir ve $d(s,\overline{S})$ şeklinde gösterilir. Açıkça görülür ki $v \in \overline{S}$ ve u∈S' dir. Bundan dolayı P yolunun s-u bölümü S kümesinin bütün düğümleri için minimum olması gerekir ve

$$d(s,\overline{S}) = d(s,u) + w(u,v) \qquad (3.1)$$

olur. Buradan da görüldüğü gibi $d(s,\overline{S})$ uzaklığının hesaplanması için (3.2) bağıntısı kullanılabilir.

$$d(s,\overline{S}) = \min_{\substack{u\in S \\ v\in\overline{S}}}\{d(s,u) + w(u,v)\} \qquad (3.2)$$

Eğer bazı u∈S için $d(s,\overline{S}) = d(s,u) + w(u,v)$ ise, buradan açık olarak

$$d(s,v) = d(s,\overline{S}) \qquad (3.3)$$

olur. Dijkstra algoritması V' nin (|V|=n) alt kümeleri olan $S_0=\{s\}$, S_1, S_2, ... serisini oluşturur. Bu alt kümeler |V|=n için, şu şartları sağlarlar;

1. Eğer $s=u_0, u_1, u_2, \ldots, u_{n-1}$, V kümesinin elemanları ve $d(s,u_1)\le d(s,u_2)\le \ldots \le d(s,u_{n-1})$ ise i>0 için $S_i=\{s, u_1, u_2, \ldots, u_i\}$ olur.

2. S_i serisi belirlendikten sonra, s' den u_1, u_2,...,u_i düğümlerine olan en kısa yollar bulunmuş olur.

Eğer S_i kümeleri belirtildiği gibi tanımlanırlarsa, (3.3) bağıntısından görüleceği gibi

$$d(s, u_{i+1}) = d(s, \overline{S}_i) \tag{3.4}$$

olur. Bundan dolayı S_{i-1}' i belirlemek için $d(s, \overline{S}_i)$' nün hesaplanması gerekir. S_1, S_2, ..., S_{n-1} alt kümeleri şu şekilde oluşturulurlar.

$$d(s, \overline{S}_0) = \min_{\substack{u \in S_0 \\ v \in \overline{S}_0}} \{d(s, u) + w(u, v)\} = \min_{v \in \overline{S}_0} \{w(s, v)\} \tag{3.5}$$

(3.4) bağıntısından görüleceği gibi u_1 düğümü (3.6) bağıntısındaki özelliği sağlar.

$$d(s, u_1) = \min_{v \in \overline{S}_0} \{w(s, v)\} \tag{3.6}$$

Eğer P_i yolu, s düğümünden u_i düğümüne olan en kısa yol ise, P_1:s, u_1 olur.

S_0, S_1, S_2, ... alt kümeleri ve P_1, P_2, ..., P_i en kısa yolların belirlendiği varsayılsın. S_{i+1} alt kümesinin belirlenmesi için, ilk olarak $d(s, \overline{S}_i)$, (3.2) bağıntısı kullanılarak hesaplanır ve u_{i+1} bir düğüm olup, (3.4) bağıntısındaki özelliği taşır. (3.2) bağıntısından dolayı, öyle bir $u_j \in S_i$ düğümü vardır ki (3.7) bağıntısındaki özelliği taşır.

$$d(s, u_{i+1}) = d(s, u_j) + w(u_j, u_{i+1}) \tag{3.7}$$

P_j yoluna (u_j, u_{i+1}) ayrıtı eklenerek P_{i+1} yolu elde edilir.

Eğer sadece s ve t düğümleri arasındaki en kısa yol aranıyorsa, bu durumda t düğümünü içeren S_i alt kümesi oluşturulunca, algoritma sonlandırılır.

Bu algoritma, (3.2) bağıntısında görülen her seviyedeki minimum değerleri hesaplamayı gerektirir. Eğer S_i alt kümesinin S_{i-1} alt kümesinden elde edilmesi esnasında her seviyede minimum değerler hesaplanırsa, (i-1)x(n-i) tane toplama ve {i(n-i)-1}

tane de karşılaştırmaya ihtiyaç vardır. Toplamda, bu algoritmanın ihtiyaç duyduğu işlem sayısı (3.8) bağıntısı ile hesaplanır.

$$\sum_{i=1}^{n-1} \{ (2i - 1)(n - i) - 1 \} \qquad (3.8)$$

Sonuç olarak, bu algoritmanın mertebesi $O(n^3)$ olur. Bununla birlikte bu algoritmada yapılan birçok toplama ve karşılaştırma işlemleri gereksiz yere yapılabilir. Bu gereksiz toplama ve karşılaştırma işlemlerinden kurtulmak için her seviyede yapılan hesaplamalar bir sonraki seviyeye aktarmak için saklanır. Bu işlem, bütün düğümlere etiket verilerek yapılır ve toplamda algoritmanın mertebesi $O(n^2)$ olur. Düğümleri etiketlemek için, $i=0,1,2,\ldots,n$ için düğümlerin etiketleri $l_i(v)$ gösterilsin ve bu etiketler (3.9) bağıntısındaki şartları sağlarlar.

1. $\forall v \neq s$, $l_0(s)=0$ ve $l_0(v)=\infty$.

2. $i \geq 1$ için, $\forall v \in S_{i-1}$ $l_i(v)=d(v,s)$

$$l_i(v) = \min_{u \in S_{i-1}} \{ d(s,u) + w(u,v) \} \qquad \forall v \in \overline{S}_{i-1} \qquad (3.9)$$

Açık olarak görüldüğü gibi, u_i düğümü (3.10) bağıntısındaki özelliği sağlar.

$$d(s,u_i) = d(s,\overline{S}_{i-1}) = \min_{v \in \overline{S}_{i-1}} \{ l_i(v) \} \qquad (3.10)$$

Buradan $l_{i+1}(v)$ etiket değeri $l_i(v)$ etiket değeri yardımıyla (3.11) bağıntısında görüldüğü gibi hesaplanır.

1. $v \in S_i$, $l_{i+1}(v)=l_i(v)=d(s,v)$

2. $v \in \overline{S}_i$,

$$\begin{aligned} l_{i+1}(v) &= \min_{u \in S_i} \{ d(s,u) + w(u,v) \} \\ &= \min \{ l_i(v), d(s,u_i) + w(u_i,v) \} \\ &= \min \{ l_i(v), l_i(u_i) + w(u_i,v) \} \end{aligned} \qquad (3.11)$$

u_{i+1} düğümü öyle seçilir ki,

$$d(s, u_{i+1}) = d(s, \overline{S}_i) = \min_{v \in \overline{S}_i}\{l_{i+1}(v)\} \qquad (3.12)$$

olur. S_i kümesi belirlendikten sonra u_i düğümünün etiketi değişmez.

Bundan dolayı Dijkstra algoritması başlangıçta $l_0(s)=0$ ve $l_0(v)=\infty$, $\forall v \neq s$ şartlarını sağlar. Algoritma işlemleri devam ettikçe, etiket değerleri, (3.11) bağıntısına göre değişirler. $l_{n-1}(v)$ etiketi d(s,v) uzaklığını verir.

u_{i+1} düğümünü belirlemek için, $\forall v \in \overline{S}_i$ için $l_{i+1}(v)$ hesaplanması gerekmektedir ve bu etiketlerin içinde minimum olanı seçilir. $i \geq 1$ için, (3.11) bağıntısı (n-i-1) adet toplama ve (n-i-1) adet karşılaştırma yapar. Halbuki (3.12) bağıntısı (n-i-2) adet karşılaştırma yapar. Buradan da Dijkstra algoritmasının mertebesinin $O(n^2)$ olduğu görülür.

Dijkstra algoritması, anlatılmadan önce bazı veri yapılarının ne amaçla kullanıldıklarının belirtilmesi gerekir. LABEL dizisi, o anki düğümlerin etiket değerlerini tutar ve PERM dizisi de hangi düğümlerin alabileceği son etiket değerine ulaştığını saklar. Eğer PERM(v)=1 ise, v düğümü alabileceği son etiket değerine ulaşmıştır. Görüleceği gibi (başlangıçta PERM(s)=1 ve PERM(v)=0, $\forall v \neq s$) düğümlerin etiketleri d(s,v) olur. PRED dizisi ise s-v yolu üzerinde her düğümün kendisinden önceki düğümünü tutar. Bu algoritma bilindiği gibi belli bir düğümden diğer düğümlere olan en kısa yolları bulmaktadır. Eğer bu algoritma çizgenin mertebesi kadar tekrar edilirse bütün düğümler arasındaki en kısa yollar bulunmuş olur. Dijkstra algoritması Algoritma 3.1' de verilmiştir.

Algoritma 3.1. Dijkstra tarafından ilk olarak uygulanan en kısa yol algoritması.

yordam EnKısaYol(Dijkstra)

1- LABEL(s)=0, PERM(s)=1, PRED(s)=s

$\forall v \neq s$, LABEL(v)=∞, PERM(v)=0, PRED(v)=v

2- u=s, i=0

3- LABEL(v)=hesaplanır, i=i+1, nihayi etiketine ulaşmayan bütün düğümler için şunlar yapılır

3.1- M=min{LABEL(v), LABEL(u)+w(u,v)}

3.2- Eğer M<LABEL(v) ise

LABEL(v)=M

PRED(v)=u

4- Nihai etiketine ulaşamayan düğümler arasında etiket değeri minimum olan düğüm seçilir (w).

PERM(w)=1

u=w

5- Eğer i<n-1 ise, 3. adıma git, aksi halde bitir.

yordam sonu

Bilgisayarda ∞ ifade edilemeyeceğine göre düğümler arasındaki mesafenin sayısal değer olarak ulaşması pek mümkün olmayan büyük sayı seçilmelidir. Sonuç olarak, algoritma uygulandıktan sonra hala etiket değeri ∞ olan düğümler için, s' den bu düğümlere yol olmadığı anlamına gelir.

Bu algoritma için, bütün ayrıt değerleri pozitif olarak alınmalıdır, çünkü negatif değerler için Dijkstra algoritması doğru çalışmaz.

Ford tarafından geliştirilen en kısa yol algoritması ise, negatif değerlere izin vermektedir, fakat negatif değerli devrelere izin vermemektedir. Çünkü bu durumda algoritma sonsuz döngüye girecektir. Bu algoritmada LABEL(v) yerine $\lambda(v)$ kullanılmaktadır.

Algoritma 3.2. Bellman-Ford tarafından geliştirilen en kısa yol algoritması.

yordam EnKısaYol(Bellman-Ford)

1- $\lambda(s)=0$ ve PRED(s)=s ve $\forall v \neq s$, $\lambda(v)=\infty$, PRED(v)=v

2- Eğer $\lambda(v_j) > \lambda(v_i)+w(e)$ değerine sahip olan $e=(v_i,v_j)$ ayrıtı bulunamıyorsa, programı bitir.

3- $e=(v_i,v_j)$ ayrıtı için $\lambda(v_j) > \lambda(v_i)+w(e)$ şartı sağlanıyorsa, bu ayrıtı seç ve $\lambda(v_j)=\lambda(v_i)+w(e)$, $PRED(v_j)=v_i$ ve 2. adıma git.

yordam sonu

3.1.2. Bütün düğümler arasındaki en kısa yollar

n düğümlü bir çizgenin n(n-1) tane sıralı düğüm ikilileri arasındaki en kısa yolların bulunacağı kabul edilsin. İlk akla gelecek olan Dijkstra algoritmasının her düğüm için ayrı ayrı kullanılmasıdır. Fakat, bununla birlikte hesapsal yönü daha iyi olan algoritmalar vardır. Bu algoritmalar negatif ağırlıklı ayrıtları olan çizgeler için de doğru çalışırlar, fakat bütün ayrıt ağırlıkları negatif olan devrenin bu çizgelerde olmaması gerekmektedir. Bu algoritmalardan biri, ilk olarak Floyd tarafından oluşturulmuştur.

Ayrıtları ağırlıklandırılmış n düğümlü bir yönlendirilmiş çizge düşünülsün. Bu çizgenin düğümleri 1, 2, ..., n ile gösterilsin ve bu çizgede ayrıtları negatif ağırlıklandırılmış devrenin olmadığı kabul edilsin. $W=[w_{ij}]$ matrisi, nxn boyutlarında bir kare matris olup, matrisin w_{ij} elemanı $e=(v_i,v_j)$ ayrıtının ağırlığını göstersin. Eğer $e=(v_i,v_j)$ gibi bir ayrıt yoksa, $w_{ij}=\infty$ olur ve $w_{ii}=0'$ dır.

Floyd algoritması başlangıçta $W^{(0)}=W$ matrisinden başlayarak, $W^{(0)}$, $W^{(1)}$, ..., $W^{(n)}$ gibi nxn boyutlarında kare matrisler serisi oluşturur. Böylece $W^{(n)}$ matrisinin $w_{ij}^{(n)}$ elemanı v^i ile v^j düğümleri arasındaki en kısa yolun uzunluğunu verir. $W^{(k)} = \left[w_{ij}^{(k)}\right]$ matrisi $W^{(k-1)} = \left[w_{ij}^{(k-1)}\right]$ matrisinden (3.13) bağıntısına göre oluşturulur.

$$w_{ij}^{(k)} = \min\{w_{ij}^{(k-1)}, w_{ik}^{(k-1)} + w_{kj}^{(k-1)}\} \qquad (3.13)$$

$P_{ij}^{(k)}$ yolu {1, 2, ..., k} düğümlerinden başka düğüm içermeyen ve uzunluğu minimum olan yoldur.

Ek olarak, en kısa yolların yanında birde bu yolların hangi düğümlerden geçtiğinin bulunması lazımdır. Bu algoritmada PRED en kısa yol üzerindeki bir önceki düğümü tutan bir dizidir. $W^{(0)}$, $W^{(1)}$, ..., $W^{(n)}$ serisi oluşturulurken, aynı zamanda $Z^{(0)}$, $Z^{(1)}$, ..., $Z^{(n)}$ serisi oluşturulur ve $z_{ij}^{(k)}$ elemanı $P_{ij}^{(k)}$ yolunda v_i düğümünden sonra gelen düğümü tutar. Bu durum (3.14)' de ifade edilmiştir.

$$z_{ij}^{(0)} = \begin{cases} j & \text{eğer } w_{ij} \neq \infty \\ 0 & \text{eğer } w_{ij} = \infty \end{cases} \qquad (3.14)$$

$Z^{(k)} = \left[z_{ij}^{(k)}\right]$ matrisi $Z^{(k-1)} = \left[z_{ij}^{(k-1)}\right]$ matris yardımıyla (3.15)

bağıntısına göre hesaplanır.

$$M = \min\{w_{ij}^{(k-1)}, w_{ik}^{(k-1)} + w_{kj}^{(k-1)}\}$$

$$z_{ij}^{(k)} = \begin{cases} z_{ij}^{(k-1)} & \text{eğer } M = w_{ij}^{(k-1)} \\ z_{ik}^{(k-1)} & \text{eğer } M < w_{ij}^{(k-1)} \end{cases} \qquad (3.15)$$

Eğer $M = w_{ij}^{(k-1)}$ ise $P_{ij}^{(k)}$ yolunun uzunluğu $P_{ij}^{(k-1)}$ uzunluğuna eşit olur. Bundan dolayı $z_{ij}^{(k)}$ değeri $z_{ij}^{(k-1)}$ değerine eşit olur. Diğer bir yandan, eğer $M < w_{ij}^{(k-1)}$ ise, $P_{ij}^{(k)}$ yolu $P_{ik}^{(k-1)}$ yolu ile $P_{kj}^{(k-1)}$ yolunun birbirine eklenmesine eşittir.

v_i-v_j en kısa yolu $v_i, v_{i_1}, \ldots, v_{i_p}, v_j$ düğümlerinden oluşsun. Buradan

$$i_1 = z_{ij}^{(n)},\ i_2 = z_{i_1 j}^{(n)}, \cdots, j = z_{i_p j}^{(n)} \qquad (3.16)$$

olarak verilebilir. (3.13) bağıntısında, eğer $w_{ik}^{(k-1)} = \infty$ veya $w_{kj}^{(k-1)} = \infty$ ise $w_{ij}^{(k)} = w_{ij}^{(k-1)}$ olur.

Algoritma 3.3. Floyd algoritması. Verilen bir çizgede bütün düğümler arasındaki en kısa yolu bulan algoritmadır.

yordam EnKısaYol(Floyd)

1- $W=[w_{ij}]$= ayrıt ağırlıkları, $w_{ii}=0$, $\forall i=1, 2, \ldots, n$
   $Z=[z_{ij}]$ nxn boyutlarında matris olup

$$z_{ij}^{(k)} = \begin{cases} z_{ij}^{(k-1)} & \text{ğer } M = w_{ij}^{(k-1)} \\ z_{ik}^{(k-1)} & \text{ğer } M < w_{ij}^{(k-1)} \end{cases}$$

2- $k=0$

3- $k=k+1$, Eğer $i\neq k$ ve $w_{ik}\neq\infty$, $j\neq k$ ve $w_{kj}\neq\infty$ ise, $M=\min\{w_{ij}$, $w_{ik}+w_{kj}$ Eğer $M<w_{ij}$ ise, $z_{ij}=z_{ik}$ ve $w_{ij}=M$

4-

i- Eğer $w_{ii}<0$ ise, G çizgesinde ayrıt uzunlukları negatif olan bir devre vardır ve bu durumda programı bitir.

ii- Eğer bütün $w_{ii}\geq 0$ ve $k=n$ ise, $[w_{ij}]$ v_i-v_j yolunun en kısa uzunluğunu verir ve $[z_{ij}]$ değeri v_i-v_j yolu içinde v_i düğümünden sonra gelen düğümü verir. Programı bitir.

iii- Eğer $w_{ii}\geq 0$ ve $k<n$ ise, 3. adıma git.

yordam sonu

3.2. Minimum Ağırlıklı Kapsar Ağaç

Devre içermeyen çizgelere devresiz çizge denir. Ağaç, bağlı ve devresiz olan bir çizgedir. Bir çizgenin kapsar ağacı, o çizgenin bütün düğümlerini içeren ağaçtır.

Çizgelerin özel bir durumu olan ağaçlar için şunlar söylenebilir.

G=(V,E) çizgesi için, |V|=n ve |E|=m olsun. Bu çizge için aşağıdaki cümlelerin hepsi birbirlerine denktirler.

1.G bir ağaçtır.
2.G çizgesinin herhangi iki düğümü arasında tek bir yol vardır.
3.G bağlı bir çizge ve m=n-1' dir.
4.G devresiz olan bir çizge ve m=n-1' dir.
5.G devresiz olan bir çizge ve eğer bitişik olmayan iki düğümü bir ayrıt ile birleştirilirse, elde edilen çizge tam olarak bir tane devre içerir.

n-düğümlü bir G çizgesi için, G_1 çizgesi G çizgesinin alt çizgesi olsun. $G_1=(V_1,E_1)$ ve $V_1=V$, $|V_1|=n$ ve $|E_1|=m_1$' dir. Aşağıdaki cümleler birbirlerine denktirler;

1.G_1 çizgesi, G çizgesinin kapsar ağacıdır.
2.G_1 çizgesinin herhangi iki düğümü arasında tek bir yol vardır.
3.G_1 çizgesi bağlı ve $m_1=n-1$' dir.

4. G_1 çizgesi devresiz bir çizge ve $m_1=n-1'$ dir.
5. G_1 devresiz bir çizge ve eğer bitişik olmayan iki düğümü bir ayrıt ile birleştirilirse, G_1 çizgesi tam olarak bir tane devre içerir.

Eğer G_1 çizgesi, G çizgesinin bir alt çizgesi olup, $V_1=V$, G_1 devresiz, bağlı ve n-1 tane ayrıtı varsa, G_1 çizgesi, G çizgesinin kapsar ağacıdır. Aşağıdaki dört özellikten en az üçünü taşıyan çizge, bir kapsar ağaç olur.

1. G_1 çizgesinin n tane düğümü vardır.
2. G_1 çizgesi bağlı bir çizgedir.
3. G_1 çizgesinin n-1 tane ayrıtı vardır.
4. G_1 çizgesi, devresiz bir çizgedir.

Eğer bir G çizgesinin kapsar ağacı varsa, o çizge bağlı bir çizgedir. Çünkü, kapsar ağaç, G çizgesinin bir alt çizgesi olup, bağlı bir çizgedir. Bunun tersi için, eğer G çizgesi bağlı bir çizge ise, bunun en az bir tane kapsar ağacı vardır.

Eğer bağlı bir çizge devresiz ise, kendisi bir kapsar ağaçtır. Eğer G çizgesi, devresiz değilse, e_1 ayrıtı bir devrenin ayrıtı olsun. $G_1=G-e_1$ bağlı ve devresiz olan bir alt çizgedir ve bütün düğümleri içerir. Eğer G1 çizgesi devresiz değilse, bir önceki adım tekrar edilir. Bu işlem bütün düğümleri içeren devresiz bir alt çizge elde edilinceye kadar devam ettirilir.

Bir çizgenin minimum kapsar ağacı, o çizgenin bütün düğümlerini içeren bir alt çizge olup, ayrıt ağırlıkları toplamı minimumdur. Bağlı ve bütün ayrıtlarına negatif olmayan ağırlıklar verilmiş bir çizge düşünülsün. Alt çizgenin ağırlığı, bu alt çizgenin ayrıt ağırlıklarının toplamına eşittir. Amaç, verilen çizgeden, ayrıt ağırlıkları toplamı minimum olan ve bütün düğümleri içeren bir ağaç oluşturmaktır. İlk olarak Kruskal [THULASIRAMAN, v.d., 1992], [BONDY, v.d., 1976], [MCHUGH, 1990], [GIBBONS, 1985] verilen çizgenin bu tip ağacını bulan algoritmayı oluşturmuştur ve aynı işi farklı bir yöntemle yapan Prim algoritması vardır [THULASIRAMAN, v.d., 1992], [BONDY, v.d., 1976], [MCHUGH, 1990], [GIBBONS, 1985].

Teorem 3.1. v ağırlıklandırılmış, bağlı ve yönlendirilmemiş bir çizgenin düğümü olsun. v düğümü ile çakışan ayrıtlar arasında e ayrıtının ağırlığı minimum olsun. G çizgesi minimum ağırlıklı kapsar öyle bir ağaca sahiptir ki, bu ağaç e ayrıtını içerir.

İspat : T_{min}, G çizgesinin minimum ağırlıklı kapsar ağacı olsun. Eğer T_{min} e ayrıtını içermiyorsa, e ayrıtına göre T_{min} ağacının temel devresi C düşünülsün. e_1 ayrıtı C devresinde e ayrıtına bitişik olan ayrıt olsun. Açık olarak $e_1 \in T_{min}$. $T_1 = T_{min} - e + e_1$ ağacı, G çizgesinin bir minimum ağırlıklı kapsar ağacıdır. e ve e_1 ayrıtları v ayrıtına çakışık olduklarından, $w(e_1) \geq w(e)$ ifadesine ulaşırız. Fakat $w(e) \geq w(e_1)$' dır. Çünkü $w(T_1) = w(T_{min}) - w(e_1) + w(e) \geq w(T_{min})$' dir. Böylece $w(e) = w(e_1)$ ve $w(T_1) = w(T_{min})$ olur. Böylece T_1 ağacı G çizgesinin minimum ağırlıklı kapsar ağacıdır■

Teorem 3.2. Ağırlıklandırılmış bağlı G çizgesinin devre içermeyen alt çizgesi T olsun. G çizgesinin minimum ağırlıklı kapsar ağacı, T alt çizgesini içerir.

Eğer G_1 çizgesi T alt çizgesinin ayrıtlarının büzüştürülmesi ile elde edildiyse ve T_{min_1} ağacı G_1 çizgesinin minimum ağırlıklı kapsar ağacı ise, $T_{min_1} \cup T$ ağacı, G çizgesinin minimum ağırlıklı kapsar ağacıdır.

İspat : G çizgesinin minimum ağırlıklı kapsar ağacı T_{min} olsun ve bu ağaç T alt çizgesini içersin. Eğer $T_{min} = T \cup T_{min_2}$ ise, T_{min_2} ağacı G_1 çizgesinin minimum ağırlıklı kapsar ağacıdır. Böylece;

$$w(T_{min_2}) \geq w(T_{min_1}) \tag{3.17}$$

olur. $T_{min_1} \cup T$ sonucun, G çizgesinin minimum ağırlıklı kapsar ağacı olduğu kolayca görülür. Böylece;

$$w(T_{min_1} \cup T) \geq w(T_{min}) = w(T) + w(T_{min_2}) \tag{3.18}$$

olur. Buradan (3.19) eşitsizliğine ulaşılabilir.

$w(T_{min_1}) \geq w(T_{min_2})$ (2.19)

(3.17) ile (3.19) eşitsizlikleri birleştirildiğinde, $w(T_{min_1}) = w(T_{min_2})$ sonucuna ulaşılır ve $w(T_{min_1} \cup T) = w(T_{min_2} \cup T) = w(T_{min})$ olur. Böylece $T_{min_1} \cup T$ ağacı G çizgesinin minimum ağırlıklı kapsar ağacıdır ■

Algoritma 3.4. Kruskal algoritması.

yordam MinimumAğırlıklıKapsarAğacı

1- G ağırlıklandırılmış bağlı bir çizge olsun.

2- i=1 ve $E_0=\varnothing$

3- Minimum ağırlıklı e_i ayrıtı seçilir ve $e_i \in E_{i-1}$ ve G çizgesinin ayrıt indirgenmiş alt çizgesi $<E_{i-1} \cup \{e_i\}>$ devre içermez, $T_i=<E_{i-1} \cup \{e_i\}>$ ve $E_i=E_{i-1} \cup \{e_i\}$ tanımlansın. Eğer böyle ayrıt yoksa, $T_{min}=T_{i-1}$ ve bitir.

4- i=i+1 ve 3. adıma git.

yordam sonu

Kruskal algoritmasının bir değişik şekli de Prim algoritmasıdır. Prim algoritması, bir düğümden başlayarak, ağacı bütün düğümleri içerecek şekilde genişletirken, o ana kadar ağaca dahil edilmiş düğümlerin, ağacın düğümler kümesinde olmayan komşu düğümleri göz önüne alır. Ağaçtaki herhangi bir düğümü, bu ağaçta olmayan düğümlerden biri ile minimum ayrıtla bağlanıyorsa, bu düğüm ağacın düğümler kümesine ve ayrıt da ağacın ayrıtlar kümesine eklenir.

Algoritma 3.5. Prim algoritması.

yordam MinimumAğırlıklıKapsarAğacı

1- G ağırlıklandırılmış bağlı bir çizge olsun.

2- i=1 ve $E_0=\varnothing$. v gibi bir ayrıt seçilsin ve $V_i=\{v\}$.

3- Öyle bir ayrıt $e_i=(p,q)$ seçilsin ki ağırlığı minimum olsun ve e_i ayrıtın uçların bir tanesinin V_i elemaı olsun, $p \in V_i$. $V_{i+1}=V_i \cup \{q\}$, $E_i=E_{i-1} \cup \{e_i\}$ ve $Ti=<E_{i-1} \cup e_i>$ tanımlansın.

4- Eğer $i<n-1$ ise, $i=i+1$ ve 3. adıma git. Aksi halde $T_{min}=T_{i-1}$ ve bitir.

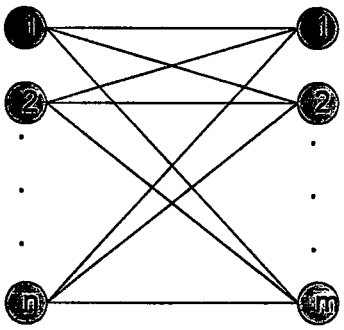
yordam sonu

3.3. İki Parçalılık

Bir çizgenin iki parçalılığı, bu çizgenin düğümler kümesinin iki ayrık alt kümeye parçalanabilmesi ve bu alt kümelerin barındırdığı düğümlerin kendi aralarında ayrıtları olmamasıdır.

Bu kavramı anlatmak için şu örnek verilebilir. Bir fabrikadaki üretim için kullanılacak olan cihazlar ile işçiler bir çizgenin düğümler kümesini oluştursunlar. İşçilerin cihazları kullanabilme yetenekleri de ayrıtlar kümesini oluştursunlar. Bu durumda, cihazlar kümesi elemanları arasında ve işçiler kümesi elemanları arasında ayrıtlar olmaz. V_1=işçiler kümesi olsun ve V_2=cihazlar kümesi olsun. $V=V_1 \cup V_2$ için, $G=(V,E)$ çizgesi bu fabrikada işçiler ile cihazlar arasındaki ilişkiyi modelleyen bir çizgedir. Bu çizge iki parçalı bir çizgedir. İki parçalı bir çizge için Şekil 3.1' de görüldüğü gibi, $V_1 \cap V_2=\varnothing$ olur.

Bir çizgenin iki parçalı olup olmadığını test etmek için, bu çizgenin bir düğümü alınır. Bu düğüm bir kümeye atılır ve bu düğümün komşuları diğer kümeye atılır. Komşuları henüz test edilmemiş düğümler işaretlenmez, kendisi ve komşuları ayrı ayrı kümelere atılan düğümler işaretlenir. Bütün düğümler işaretleninceye kadar devam edilir. Eğer daha işaretlenmemiş düğüm var ve herhangi bir düğümün kendisi ve komşularından en az biri aynı kümeye atılmışsa, bu çizge iki parçalı değildir. Aksi halde, bu çizge iki parçalıdır. Bu işlemi yapan algoritma Algoritma 3.6' da görülmektedir.



Şekil 3.1. İki parçalı bir çizge.

Algoritma 3.6. Bir çizgenin iki parçalı olup olmadığını test eden program

```
yordam İkiParçalılık(G)
      1- İkiParçalı=1
      2- ∀v∈V için P[v]=0
      3- işaretlenmemiş u∈V düğüm olduğu sürece işleme devam et
            Parça=Parça∪{u}
            P[u]=1
            Parça<>∅ olduğu sürece devam et
                  Parça=Parça-{v}
                  Bütün v' nin w komşuları için eğer P[w]=0 ise
                        Parça=Parça∪[w}
                        P[w]=3-P[w]
                  Eğer P[v]=P[w] ise
                        İkiParçalı=0 ve Bitir
yordam sonu
```

3.4. Çizgelerin Bağlılığı

G=(V,E) çizgesi yönlendirilmemiş bir çizge olsun. $v_i, v_j \in V$ için, eğer en az bir tane v_i-v_j yolu varsa, v_i ve v_j düğümleri bağlıdır denir. Her düğüm, doğal olarak kendisine bağlıdır. Eğer

bir çizgenin her düğüm çifti için, en az bir tane yol varsa, bu çizge bağlı çizgedir. Eğer yönlendirilmiş bir G=(V,E) çizgesinin bütün düğüm çiftleri için, v_i-v_j yolu var ve aynı zamanda v_j-v_i yolu da varsa, bu çizgeye kuvvetli bağlı çizge denir.

G=(V,E) çizgesi bağlı olmayan bir çizge olarak düşünülsün. Bu durumda, V kümesi birden fazla alt kümeye $V_1,...,V_p$ bölünebilir. Bu alt kümelerin düğümlerinin kendi içinde bitişiklikleri söz konusudur. Fakat bu alt kümeler arası düğümlerin bitişikliği söz konusu değildir. (3.20) bağıntısında G çizgesinin alt çizgelerinin özelliği verilmiştir. Bu alt çizgelere G çizgesinin parçaları denir.

$$\{v_i,v_j\}\in E \Leftrightarrow v_i,v_j\in V_k \text{ ve } v_i,v_j\notin V_r,\ 1\le k,r\le p \text{ ve } r\neq k \qquad (3.20)$$

Yönlendirilmemiş bir çizgenin bağlı olup olmadığını test etmek, yönlendirilmiş bir çizgenin bağlı olup olmadığını test etme işlemine göre kolaydır. Eğer yönlendirilmiş bir çizgenin sadece bağlılığı test edilecekse, bu işlem bu çizgenin kuvvetli bağlı olup olmadığını test etmeye göre daha kolaydır. Çünkü yönlendirilmiş bir çizgenin kuvvetli bağlı olması için, eğer v_i-v_j gibi bir yönlendirilmiş yol varsa, v_j-v_i gibi bir yönlendirilmiş yol olmak zorundadır. Bu zorunluluk bütün düğüm çiftleri için geçerlidir. Bu testi yapabilmek için yapay zekada kullanılan arama algoritmalarından ilk-önce-derinlik veya ilk-önce-genişlik algoritmalarından birinin kullanılması gerekir. Burada bu yapay zeka algoritmaları verilmeyecektir.

Yönlendirilmemiş çizgelerde ise, çizgenin bağlı olması kuvvetli bağlı anlamına da gelir. Bundan dolayı sadece çizgenin bağlılığı test edilir. Algoritma 3.7, yönlendirilmemiş bir çizgenin bağlılığını test eder. Algoritmada verilen B dizisinin her elemanı çizgenin bir parçasının düğümlerini içerir.

Algoritma 3.7. Yönlendirilmemiş bir çizgenin kendi içinde parçalarını bulur.

yordam Bağlı

1- G=(V,E) yönlendirilmemiş bir çizge ve B her elemanı bir küme olan bir dizidir. A nxn boyutlarında bitişiklik matrisi

2- i=1,|V|=n, B bütün düğümler test edilinceye kadar devam et

3- Eğer $\forall v_i \in V$ ise, $C=v_i$ düğümünün komşu düğümleri kümesini oluştur.

4- B[i]=B[i]+C ve V=V-C

5- Eğer B[i] eklenen yeni düğümlerin komşularından en az birinin B[i]' de olmaması durumunda 3. adıma git

6- Eğer i<n ve V≠{} ise, i=i+1, 3. adıma git

yordam sonu

4. ÇİZGE BÖLMELEME

Çizgelerle ifade edilebilecek bilimsel hesaplamalarda ortaya çıkan kombinatorik yapıya sahip olan problemler vardır. Bu problemlerin çözümünün daha az zaman alması için problem veri kümesinin ayrık olan alt kümelere parçalanması gerekir. Bunu gerçekleştirmek için problemi temsil eden çizgenin bölmelenmesi gerekmektedir. Çizge bölmeleme, araştırma alanlarında ve paralel hesaplamalarda çok bilinen bir problemdir. VLSI devre tasarımında, yük dengelemede, seyrek matrislerin düzenlenerek mühendislik ve optimizasyon uygulamalarında ortaya çıkan sistemlerin doğrudan çözümünün elde edilmesinde kullanılır. Sonuç olarak, çizge bölmeleme, bilimsel hesaplama, optimizasyon, VLSI tasarımı, yük dengeleme gibi alanlarda uygulama alanı bulan önemli bir problemdir.

Çizgeler üzerinde çeşitli işlemler yapılabilmektedir. Bu işlemlerden biri de çizge bölmelemedir. Bazı bilimsel ve mühendislik problemlerinin çözümünde veriler işlemcilere öyle bir şekilde dağıtılır ki işlemcilere düşen yük dengelidir ve işlemciler arası iletişim en aza indirgenmiştir. Bu, bir çizge bölmeleme problemidir. Çizgedeki düğümler yapılacak işi ve düğümler arası ayrıtlar da bu işlerin atandığı işlemciler arasındaki iletişimi gösterir. Optimal bölmeleme, işlemcilere düşen işin hemen-hemen (mümkünse) dengeli ve işlemciler arasındaki iletişimin de minimuma indirgenmesidir, bu da iletişim ve işlerin yapılması için harcanacak zamanın minimuma indirgenmesidir.

Problemi oluşturan veriler, işlem safhalarında değişmiyorlarsa, bu durumda bölmeleme işlemine başlamadan önce çizge oluşturulur ve bu çizgenin bölmeleme işlemi çeşitli algoritmalara göre bir kez yapılır.

Eğer veriler işlem safhasında değişiyorlarsa, bu durumda bölmeleme işlemi ilk verilere göre yapılır. Bundan sonra yeni veriler eklenince bölmeleme yapılan çizgede bu eklemeye göre düzenleme yapılır veya veri silinmesi söz konusu ise benzer

işlem bunun için de tekrar edilir [OU, v.d., 1997]. Bölmeleme işlemi, verilerin durumuna göre süreklilik arz eden bir oluştur [OU, v.d., 1997].

Çizge bölmeleme işlemi NP-tamam (NP-complete) bir problemdir. Bundan dolayı bölmeleme işlemi yapan algoritmanın matematiksel karmaşıklığı (complexity) bir polinom değildir, eksponansiyel bir bağıntıdır. Günümüze kadar bulunan algoritmalardan, hiç biri verilen çizgenin optimum bölmeleme işlemini yaptığını garanti edemiyor. Buna rağmen geliştirilen algoritmaların çoğu iyiye yakın veya iyi sonuçlar vermektedir.

Probleme göre verilen çizgenin düğümlerinin koordinatları olabilir veya olmayabilir. Eğer koordinatları varsa, bu durumda bölmeleme işlemi biraz daha kolaylaşmaktadır [LASZEWSKI, 1993]. Çünkü düğümlerin eksen koordinatlarına göre ilk önce bir sıralaması yapılır, ondan sonra bölmeleme işlemi yapılır (kullanılan yöntem böl ve yönet). Elde edilen parça sayısı yetersiz ise bu parçalar yine aynı yöntem ile diğer eksene (üzerinde sıralama işlemi yapılmamış veya en az yapılmış eksen) göre sıralanır ve bu sıralama işleminden sonra bölmeleme işlemi uygulanır. Genelde her parça kendi içinde sıralanır ve ondan sonra o parça ikiye bölmelenir.

Eğer düğümlerin koordinatları yoksa, bu durumda çizgenin bölmeleme işlemi için genel olarak üç değişik yöntem uygulanır. Bunlardan biri gelişi güzel başlangıç bölmeleme işleminin yapılması, ikincisi çizgenin öz değerleri ve bu öz değerlere denk düşen öz değer vektörlerini bularak bölmeleme işleminin yapılması [KARYPIS, v.d., 1995], [ZHOU, 1993], [KIM, v.d., 1993], üçüncü olarak da eğer çizgenin düğüm sayısı fazla ise ilk önce çizge üzerinde gelişi güzel eşleştirme yapılır veya maksimum-ayrıt-önce metodu ile eşleştirme yapılır ve eşleşen düğümler bir düğüme indirgenir (coarsening)[ZHOU, 1993], [KARYPIS, v.d., 1996], [KIM, v.d., 1993], [KARYPIS, v.d., 1997], [HENDRICKSON, v.D., 1995], [KARYPIS, v.d., 1995]. Bu şekilde istenen sayıda düğüm elde edildikten sonra yukarıda belirtilen yöntemlerden biri kullanılarak bölmeleme işlemi yapılır ve ondan sonra çizge geri açılarak (uncoarsening) orijinal çizgenin

bölmelenmiş hali elde edilir (çok seviyeli çizge bölmeleme metodu). Başlangıç parçalar elde edildikten sonra bu parçalara iyileştirme metotları uygulanarak hata oranı en aza indirgenmeye çalışılır (Kernighan-Lin).

Eğer çizge homojen bir yapıya sahipse (bütün düğüm ağırlıkları eşit ve bütün ayrıt ağırlıkları eşit), bu durumda çoğu zaman bölmeleme işleminden sonra iyileştirme işlemine ihtiyaç duyulmaz, eğer çizge homojen değilse, bu durumda da çoğu zaman iyileştirme işlemine ihtiyaç duyulur.

Çizge bölmeleme işlemlerinde kullanılan yöntemler genel olarak iki bölümden oluşmaktadır. Bunlar, oluşturma ve iyileştirme algoritmalarıdır. Bu algoritmalardan şu anda kullanılanların bir listesi verilirse,

Bölmeleme (oluşturma) algoritmaları

1. Gelişi güzel bölmeleme yapan algoritmalar
a) Lineer
b) Gelişi güzel
c) Serpme
2. Spektral Çizge Bölmeleme
3. Özyinelemeli Eş İki Parçaya Bölmeleme Algoritması
4. Eylemsizlik
5. Basit Çizge İndirgeme
6. Çok Seviyeli Çizge Bölmeleme

İyileştirme Algoritmaları

1. İki optimal iyileştirme algoritması
2. Kernighan-Lin iyileştirme algoritması
3. Isı düşüşünün taklidi (Simulated annealing - SA)
4. Enerji durum taklidi ve Uyumlu ısı düşüşü taklidi (Simulated Tempering and Adaptive Simulated Tempering - AST)

4.1. Oluşturma Algoritmaları

Oluşturma algoritmaları çizgenin başlangıç bölmeleme işlemini yapan algoritmalardır. Bu algoritmalardan sonra iyileştirme algoritmaları uygulanır.

4.1.1. Gelişi güzel bölmeleme yapan algoritmalar

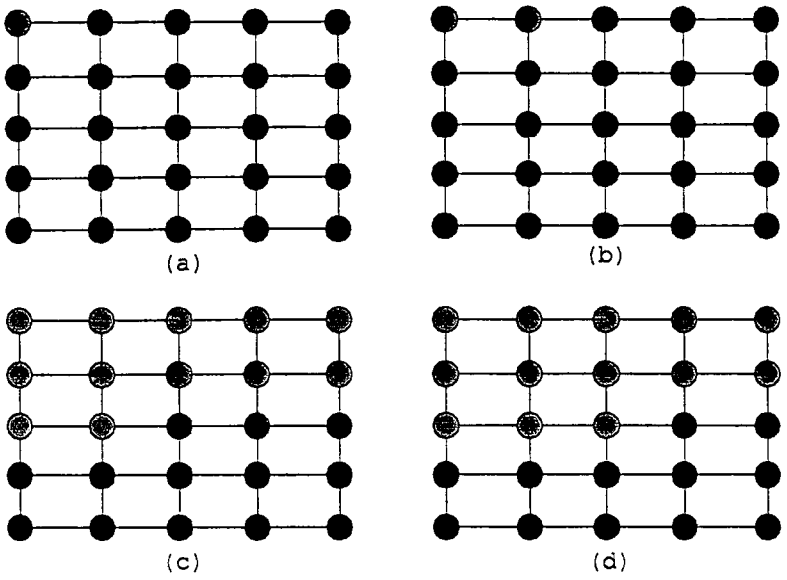
Gelişi güzel bölmeleme algoritmalarının her birinin belli bir algoritmik yapısı vardır. Gelişi güzel denilmesinin sebebi yük dengeleme şartını ve minimum kesim ayrıt boyutunu göz önüne almamalarından kaynaklanır.

4.1.1.1. Lineer algoritma

Bu algoritmaya göre çizge bir parça olarak alınır ve bu parçadan ilk önce bir düğüm alınır ve ikinci parça olarak kabul edilir. Eğer parçalardaki düğüm ağırlıkları eşit veya birbirine çok yakın değilse toplam ağırlığı fazla olandan bir düğüm seçilir ve diğer parçaya eklenir, tekrar parçaların ağırlıkları kontrol edilir. Seçilecek olan düğümün en fazla bir tane komşu düğümünün diğer parça içinde olması gerekir. Elde edilen parçalara iyileştirme işlemi uygulanır. Lineer bir algoritmanın çalışma şekli, Şekil 4.1' de görülmektedir.

4.1.1.2. Gelişigüzel algoritma

Gelişigüzel çizge bölmeleme algoritmasında ise çizge gelişi güzel iki parçaya bölünür ve bu parçalara iyileştirme algoritmaları uygulanır. Bu yöntemin iyileştirme kısmı çok yavaş olabilir. Çünkü çizge gelişigüzel iki parçaya bölünmektedir ve bu parçalar bir birinden çok farklı parçalar olabilirler.



Şekil 4.1. Lineer çizge bölmeleme algoritması (düğüm ve ayrıt ağırlıkları aynı olan 5x5 ağ çizgesi), (a) bir parçada bir düğüm, diğer parçada ise geriye kalan bütün düğümler, (b) bir parçada iki düğüm, diğer parçada ise geriye kalan bütün düğümler, (c) birinci parçada $\lfloor n/2 \rfloor$ düğüm diğer parçada ise $\lfloor n/2 \rfloor+1$ düğüm var, (d) birinci parçada $\lfloor n/2 \rfloor+1$ düğüm varken, diğer parçada ise $\lfloor n/2 \rfloor$ düğüm vardır.

4.1.1.3 Serpme algoritması

Serpme algoritmasında ise, başlangıçta çizgenin her iki parçasını içerecek kümeler boş durumdadır. Bölmeleme işlemi yapılırken, bir düğüm seçilir ve bu seçilen düğüm ağırlık toplamı az olan kümeye atılır ve eğer her iki kümenin ağırlık toplamları eşit ise bu durumda seçilen düğümün hangi kümede komşu düğüm sayısı fazla ise, o kümeye katılır ve eğer hem kümelerin ağırlık toplamları ve hem de seçilen düğümün kümelerde ki komşu ayrıt sayısı eşitse, bu durumda, düğüm gelişi güzel bir kümeye kayıt edilir. Bu işlemlerden sonra eğer her iki parça arasındaki fark büyükse iyileştirme işlemi uygulanabilir. Eğer her iki kümenin düğümlerin ağırlıkları toplamı birbirine denk

ise ve $W(E_{kesim})$ değeri kabul edilecek kadar küçükse, bu bölmelemenin sonucuna iyileştirme işleminin uygulanmasına gerek kalmaz.

4.1.2. Spektral çizge bölmeleme

Spektral çizge bölmeleme yönteminde, numerik metotlar kullanılmaktadır [HENDRICKSON, v.d., 1995], [SPIELMAN, v.d., 1996], [LASZEWSKI, 1993], [HENDRICKSON, v.d., 1993], [BARNARD, v.d., 1993], [HENDRICKSON, v.d., 1992], [LELAND, v.d., 1994], [SUARIS, vd., 1988]. Bunlardan biri de reel-simetrik matrislerin öz değerleri ve öz değer vektörlerinin bulunmasıdır. Bir çizgenin Laplace matrisi reel ve simetrik olan bir matristir. Bu matrisin öz değer ve öz değer vektörleri bulunur. Bu matrisin ikinci en küçük öz değere denk düşen öz değer vektörü çizgenin iki parçaya bölmelenmesinde kullanılır. Bu yöntemi ilk olarak Fiedler kullanmıştır [HENDRICKSONN, v.d., 1993]. Bu ikinci en küçük öz değerin öz değer vektörünÜ çizgenin cebirsel bağlılığı olarak adlandırmıştır. Fiedler' in bu çalışmasından dolayı, bu öz değere matrisin Fiedler değeri ve öz değer vektörüne de Fiedler vektörü denilmiştir. Bu vektör çizgenin en küçük ayıracını verir.

Bu yöntem, bir çok numerik algoritma için algoritmanın önemli bir parçasını oluşturmaktadır. Spektral bölümleme yöntemi, çizge ve matris bölmelemede oldukça fazla kullanılmakla birlikte devre dizaynı ve simülasyonlarında da kullanılmaktadır.

G=(V,E) bağlı ve yönlendirilmemiş bir çizge olsun. Çizgenin bölmelenmesi, verilen çizgenin düğümlerinin ayrık birden fazla alt kümeye bölmelenmesidir. Bu alt kümeler A ve $\overline{A}$ olsun ($A \cap \overline{A} = \varnothing$ ve $A, \overline{A} \subset V$). Başlangıçta $|A| \leq |\overline{A}|$ kabul edilmesi yöntemin genelliliğini bozmaz. $E(A,\overline{A})$ çizgenin ayıracı olsun. $(A, \overline{A})$ bölmelemenin kesme boyutu $|E(A,\overline{A})|$ olur ve buna çizgenin ayıracının boyutu denir ve genelde bölmeleme işlemlerinde bunun minimum olması istenir. Kesme oranı $\phi(A,\overline{A})$ aşağıdaki eşitlik ile hesaplanır.

$$\phi(A,\overline{A}) = \frac{|E(A,\overline{A})|}{\min(|A|,|\overline{A}|)} \qquad (4.1)$$

Bir çizgenin beklenen en iyi kesme oranına o çizgenin izoperimetrik sayısı denir ve aşağıdaki eşitlik ile hesaplanır.

$$\phi(G) = \min_{|A|\le n/2} \frac{|E(A,\overline{A})|}{|A|} \qquad (4.2)$$

Bir çizgenin bölümleri arasında en fazla 1 düğüm fark varsa bu bölümlere o çizgenin eş iki parçaları denir. Bir çizgenin alt çizgeleri ve izoperimetrik değeri bulunduktan sonra o çizgenin eş iki parça boyutunun $O(\phi n)$ olduğu görülür.

Öneri 4.1 : k-düğüm bir alt çizgenin kesme oranını en fazla $\phi(k)$ olarak bulan bir algoritmanın verildiği kabul edilsin ($\phi(k)$ fonksiyonu monoton azalan bir fonksiyondur). Tekrarlı bir şekilde bu algoritma kullanılarak verilen çizgenin eş iki parça boyutu en fazla şu integralin sonucu olarak bulunur.

$$\int_{x=1}^{n} \phi(x)dx \qquad (4.3)$$

İspat : Aşağıdaki algoritma verilen bir çizgenin eş iki parçasını bulur.

1. Başlangıçta, $D^{(0)}=G$, A ve B boş kümeler olsunlar, i=0
2. Eğer $D^{(i)}$ boş bir küme ise, A ve B' yi çizgenin eş iki parçaları olarak geriye döndür, aksi halde işleme devam et
   - $D^{(i)}$' yi F(i) ve $\overline{F^{(i)}}$ gibi iki parçaya bölen ve boyutu en fazla $\phi(|D^{(i)}|)$ olan bir kesme oranı bulun. $|F^{(i)}| \le |\overline{F^{(i)}}|$ olduğu kabul edilir.
   - Eğer $|A| \le |B|$ ise, $A=A \cup F^{(i)}$ aksi halde $B=B \cup F^{(i)}$.
   - $D^{(i+1)} = \overline{F^{(i+1)}}$ ve i=i+1, (a) adımına geri dön.

Bu algoritmanın çalışmasının t iterasyon sonucunda durduğu kabul edilsin. Bu algoritmanın eş iki parçayı bulduğunu

göstermek için, $\forall i, 0 \le i < t$ için $\min(|A|,|B|)+|F(i)| \le n/2$ olur. Çünkü

$|F(i)| \le |\overline{F^{(i)}}|$ ve

$\min(|A|,|B|)+|F^{(i)}| \le (|A|+|B|+|F^{(i)}|+|\overline{F^{(i)}}|)/2 = n/2'$ dir. (4.4)

Bu şekilde toplam olarak kesmenin boyutu analiz edilebilir. Çünkü i. iterasyonda algoritma, kesme oranını en fazla $\phi(|D^{(i)}|)$ olarak bulur. $F^{(i)}$' yi bölmek için kesilecek ayrıt sayısı

$$\begin{aligned} \phi(|D^{(i)}|)\ |F^{(i)}| &= \sum_{j=1}^{|F^{(i)}|} \phi(|D^{(i)}|) \\ &= \sum_{j=|D^{(i)}|}^{|D^{(i)}|-|F^{(i)}|+1} \phi(|D^{(i)}|) \\ &\le \sum_{j=|D^{(i)}|}^{|D^{(i)}|-|F^{(i)}|+1} \phi(j) \end{aligned} \qquad (4.5)$$

ϕ fonksiyonu monoton olarak azalan bir fonksiyon olduğundan dolayı yukarıdaki işlemin sonucu eşitsizlik olmaktadır. Bu algoritma kullanılarak en fazla kesilecek ayrıt sayısı

$$\sum_{i=0}^{t-1} \phi(|D^{(i)}|)|F^{(i)}| \le \sum_{i=0}^{t-1} \left(\sum_{j=|D^{(i)}|}^{|D^{(i)}|-|F^{(i)}|+1} \phi(j) \right)$$

$$\begin{aligned} &= \sum_{j=1}^{n} \phi(j) \\ &\le \int_{1}^{n} \phi(x)dx \end{aligned} \qquad (4.6)$$

ϕ fonksiyonu monoton azalan bir fonksiyon olduğundan yukarıdaki işlemin son adımı eşitsizlik olarak verilmiştir.

Teorem 4.1 : Eğer $\phi(x)=x^{-1/d}$ ise

$$\int_{1}^{n} \phi(x)dx = \frac{d}{d-1}\ (n^{1-1/d}-1) \qquad (4.7)$$

4.1.2.1. Laplace matrisi ve Fiedler vektörleri

Bir çizgenin bitişiklik matrisi |V|=n için, nxn boyutunda olan bir matristir. Eğer $(v_i,v_j)\in E$ ise matrisin (i,j)-elemanı 1 diğer durumlarda sıfırdır. Bir çizgenin derece matrisi de nxn boyutunda bir kare matristir. (i,i)-elemanı v_i düğümünün derecesine eşit, diğer elemanlar ise sıfırdır. Çizgenin bitişiklik matrisi ile derece matrisinin farkı alınarak Laplace matrisi elde edilir. Laplace matrisi $L(G)=(l_{ij})$, i,j=1,...,n' dir

$$l_{ij} = \begin{cases} 1 & \text{eğer } \{v_i, v_j\} \in E \\ -\deg(v) & \text{eğer } i = j \\ 0 & \text{diğer durumda} \end{cases} \qquad (4.8)$$

Laplace matris öz değerlerinden biri 0 ve bu öz değere karşılık bulunan öz değer vektörü ise bütün elemanları 1 olan bir vektördür. Çünkü Laplace matrisi pozitif yarı tanımlı bir matristir ve bundan dolayı diğer bütün öz değerleri negatif olmayan değerlerdir. Bundan sonraki işlemlerde Laplace matrisinin ikinci en küçük öz değere denk düşen öz değer vektörü bulunur. Daha öncede belirtildiği gibi bu öz değere Fiedler değeri ve bu öz değere karşılık bulunan öz değer vektörüne de Fiedler vektörü denir.

Her hangi bir $\vec{x} \in R^n$ vektörü için

$$\vec{x}^T L(G)\vec{x} = \sum_{(i,j)\in E} (x_i - x_j)^2 \qquad (4.9)$$

ve bununla birlikte, G çizgesinin Fiedler değeri de

$$\lambda_2 = \min_{\vec{x}\perp(1,1,\ldots,1)} \frac{\vec{x}^T L(G)\vec{x}}{\vec{x}^T\vec{x}} \qquad (4.10)$$

olur. Eğer $\vec{x}$ Fiedler vektörü ise yukarıdaki oran minimum olacaktır. Genel olarak

$$\frac{\vec{x}^T L(G)\vec{x}}{\vec{x}^T\vec{x}} \qquad (4.11)$$

oranına $\vec{x}$ vektörünün Rayleigh oranı denir. N.Alon, A.J. Sinclair ve M.R. Jerrum küçük Fiedler değerli çizgelerin iyi bir kesme oranına sahip olduğunu ispatlamışlardır (Alon küçük düğüm ayıracının varlığını ispatlamıştır) [SPIELMAN, v.d., 1996]. M. Mihail çalışmaları sonucunda, bütün elemanları 1 olan vektöre dik olan ve Rayleigh oranı küçük olan her vektör kullanılarak iyi bir kesme oranı elde edilebilir. M. Mihail' e ait olan teorem şu şekildedir.

Teorem 4.2: G=(V,E) n tane düğümü ve düğümlerin maksimum derecesi Δ olan bir çizge olsun ve L(G) bu çizgenin Laplace matrisi olsun, ve ϕ bu çizgenin izoperimetrik sayısı olsun. Öyle her hangi bir $\vec{x} \in R^n$ vektörü vardır ki $\sum_{i=1}^{n} x_i = 0$,

$$\frac{\vec{x}^T L(G)\vec{x}}{\vec{x}^T\vec{x}} \geq \frac{\phi^2}{2\Delta} \qquad (4.12)$$

Bundan başka, öyle bir s (kesme ayıracı) vardır ki

$$(\{i:x_i \leq s\},\{i:x_i > s\}) \qquad (4.13)$$

kesmesi en fazla $\phi^2/(2\Delta)$ oranına sahiptir.

4.1.2.2. Spektral bölmeleme yöntemleri

G çizgesinin Laplace matrisinin Fiedler vektörü $\vec{u}=(u_1,\ldots,u_n)$ olsun. Spektral çizge bölmelemenin ana fikri, bir tane bölmeleme değeri s bulunup, Fiedler vektörü iki parçaya bölünür, öyle ki bir parça için $u_i > s$ ve diğer parça için $s \leq u_i$ olur. Bu tip bir çizge kesmesine Fiedler kesmesi denir. Genel olarak Fiedler kesmesini bulmak için, $(u_1,\ldots,u_n)$' nin medyanı s olarak alınır ve bu şekilde seçilen s kesme ayıracına eş iki parça kesmesi denir. s' nin en iyi kesme oranını verdiği durumda ise, buna kesme oranı, s' nin değerinin 0 olması durumunda s' ye işaret kesmesi, s sıralanmış Fiedler vektörü elemanları içinde

en uzun aralık içinde bir değere sahipse buna da aralık kesmesi denir.

4.1.2.3. Matrislerin öz değeri ve öz değer vektörünün hesabı

Verilen bir denklem sisteminin abes olmayan bir çözümünün olabilmesi için, bu denklem sisteminin katsayılar matrisi elde edildikten sonra bundan elde edilen vektörlerin bağımsız olması gerekmektedir. $U_1,...,U_n$ vektörler olsunlar [MATHEWS, 1992], [DAHQUIST, v.d., 1974]. Eğer bu vektörler lineer bağımsız iseler

$$c_1U_1+...+c_nU_n=0 \qquad (4.14)$$

(4.14) eşitliğindeki katsayılar için, $c_1=c_2=...=c_n=0$ eşitliği sağlar. Eğer en az bir tane c_i (i=1,2,...,n) sıfırdan farklı ise bu vektörler lineer bağımsız değildirler. X bir vektör olsun ve A katsayılar matrisi olsun

$$AX=\lambda X \qquad (4.15)$$

eşitliğini sağlayan λ, A matrisinin öz değeridir ve X' te λ' ya karşılık gelen öz değer vektörüdür. (4.15) eşitliği kullanılarak

$$(A-\lambda I)X=0 \qquad (4.16)$$

(4.16) eşitliği elde edilir. Bu eşitliğin determinantı alınınca A matrisinin karakteristik polinomu elde edilir. Bu polinomun kökleri A matrisinin öz değerleridir. Bu öz değerler kullanılarak öz değer vektörleri elde edilebilir. Bilgisayar ortamında bunun çözümü biraz daha farklı olmaktadır. Bunun için geliştirilmiş yöntemler vardır. Bu yöntemler Kuvvet yöntemi, Jacobi yöntemi, Householder yöntemi gibi yöntemlerdir. Kuvvet yöntemi en küçük öz değere karşılık gelen öz değer vektörünü bulmaktadır. Jacobi yöntemi bütün öz değerleri ve bunlara karşılık düşen öz değer vektörlerini bulmaktadır. Householder yöntemi ise reel simetrik olan bir matrisi üç-diyagonal matrisine dönüştürmektedir.

Jacobi yönteminde bütün öz değerler ve öz değer vektörleri bulunabildiğinden, bu çalışmada, bu yöntem kullanılmıştır ve burada bu yöntemden söz edilecektir. Jacobi yöntemi simetrik matrislerin öz değerlerini ve öz değer vektörlerini bulmada kullanılmaktadır. Bir çizgenin Laplace matrisi de simetrik bir matris olduğundan, bu yöntem kullanılabilir. Bu yöntem bütün simetrik reel matrislerin öz değer ve öz değer vektörler probleminin çözümünü garanti etmektedir.

4.1.2.3.1. Düzlem döndürme

Esas algoritmaya geçilmeden önce bazı geometrik bilgilerin verilmesi gerekmektedir. X n-boyutlu bir vektör olsun ve (4.17) bağıntısındaki lineer dönüşümün yapıldığı varsayılsın.

$$Y=RX \tag{4.17}$$

$$R=\begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & \cos\phi & \cdots & \sin\phi & \cdots & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & -\sin\phi & \cdots & \cos\phi & \cdots & 0 \\ \vdots & & & & & & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \begin{matrix} \\ \\ \leftarrow p \\ \\ \leftarrow q \\ \\ \\ \uparrow \quad \uparrow \\ p \quad q \end{matrix} \tag{4.18}$$

R, nxn boyutunda diyagonal elemanları 1 ve diyagonal dışındaki elemanları sıfır olan bir matristir. Fakat bir fark vardır, p ve q sütun ve satırına denk gelen diyagonal elemanları $\cos\phi$, $r_{pq}=-\sin\phi$ ve $r_{qp}=\sin\phi$. (4.17) bağıntısındaki lineer dönüşümü (4.19) bağıntısındaki gibi olur.

$$\begin{aligned} y_j &= x_j \text{ eğer } j\neq p \text{ ve } j\neq q \\ y_p &= x_p\cos\phi + x_q\sin\phi \\ y_q &= -x_p\sin\phi + x_q\cos\phi \end{aligned} \tag{4.19}$$

Bu dönüşüm n-boyutlu bir uzayın $x_p x_q$-düzleminde ϕ açısı ile döndürülmesidir. Bunun ters dönüşümü ise aynı uzayın aynı düzlemde $-\phi$ açısı ile döndürülmesidir ve ters dönüşüm $X=R^{-1}Y$. R matrisi ortogonal bir matris olduğundan

$$R^{-1}=R^T \text{ veya } R^TR=I \qquad (4.20)$$

olur.

4.1.2.3.2. Benzerlik ve ortogonal dönüşümler

$$AX=\lambda X \qquad (4.21)$$

Bu öz değer problemi düşünülecek olursa, K matrisi singular olmayan bir matris olsun ve B (4.22) bağıntısına göre tanımlanmış kabul edilerek,

$$B=K^{-1}AK \qquad (4.22)$$

(4.22) bağıntısının her iki tarafı $K^{-1}X$ çarpılarak olursa

$$BK^{-1}X=K^{-1}AKK^{-1}X=K^{-1}AX=K^{-1}\lambda X=\lambda K^{-1}X \qquad (4.23)$$

elde edilir. Değişken değişimi aşağıdaki gibi yapılırsa

$$Y=K^{-1}X \text{ veya } X=KY \qquad (4.24)$$

elde edilir. Eğer (4.24) bağıntıları (4.23)' de yerine yazılırlarsa

$$BY=\lambda Y \qquad (4.25)$$

yeni bir öz değer problemi elde edilmiş olur. (4.21) ve (4.25) bağıntıları karşılaştırılırlarsa, öz değerleri aynı ve öz değer vektörleri aynı olmak zorunda olmayan iki tane matrisin elde edildiği görülür (A ve B matrisleri). Bu da benzerlik dönüşümünün öz değerleri koruduğunu gösterir.

R matrisinin ortogonal olduğu kabul edilsin ve D matrisi şu şekilde tanımlansın.

$$D=R^TAR \tag{4.26}$$

Bu bağıntının her iki tarafı R^TX ile çarpılırsa

$$DR^TX=R^TARR^TX=R^TAX=R^T\lambda X=\lambda R^TX \tag{4.27}$$

elde edilir. Aynı şekilde değişken değişimi şu şekilde yapılabilir.

$$Y=R^TX \text{ veya } X=RY \tag{4.28}$$

(4.27) ve (4.28) beraber kullanılırsa, yeni bir öz değer problemi elde edilir.

$$DY=\lambda Y \tag{4.29}$$

$R^{-1}=R^T$ olduğundan X' ten Y' nin ve Y' den X' in elde edilmesi kolaydır. (4.21) bağıntısı ile (4.29) bağıntısı karşılaştırılırsa, öz değer probleminde öz değerlerin değişmediği ve sadece yeni bir matrisin elde edildiği görülür.

Eğer A matrisi simetrik bir matris ise

$$D^T=(R^TAR)^T=R^TA(R^T)^T=R^TAR=D \tag{4.30}$$

olur. Böylece D matrisi de simetrik bir matristir. Böylece eğer A simetrik ve R ortogonal matrisler ise, A matrisinden D matrisine olan dönüşüm hem öz değerleri ve hem de simetrikliği korumaktadır. Öz değer vektörleri arasındaki ilişki ise (4.28)' de verilmiştir.

4.1.2.3.3 Dönüşümlerin Jacobi serileri

Dönüşümlere, reel ve simetrik olan A matrisi ile başlanır ve ortogonal olan R_1, R_2, ..., R_n matrisleri şu şekilde elde edilirler.

$$\begin{aligned} D_0 &= A \\ D_j &= R_j^T D_{j-1} R_j \quad j=1,2,\ldots \end{aligned} \tag{4.31}$$

Pratikte D matrisinin diyagonal üzerinde olmayan elemanları sıfıra yaklaşınca işlem durdurulur ve

$$\lim_{j \to \infty} D_j = D = \mathrm{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n) \qquad (4.32)$$
$$D_n = D$$

olur. Buradan

$$D_n = R_n^T R_{n-1}^T \cdots R_1^T A R_1 R_2 \cdots R_{n-1} R_n \qquad (4.33)$$

olur. Eğer R matrisi için şu tanımlama yapılırsa

$$R = R_1 R_2 \ldots R_{n-1} R_n \qquad (4.34)$$

olur ve bu tanımlamadan yola çıkılarak, $R^{-1}AR=D$ aşağıdaki sonucu verir.

$$AR = RD = R\ \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) \qquad (4.35)$$

R matrisinin sütunları X_1, X_2, ..., X_n vektörleri ile gösterilirse, $R=[X_1, X_2, .., X_n]$ olur. Buradan

$$AR = [AX_1, AX_2, \ldots, AX_n] = [\lambda_1 X_1, \lambda_2 X_2, \ldots, \lambda_n X_n] \qquad (4.36)$$

Görüldüğü gibi R matrisinin sütunları öz değer vektörlerini göstermektedir.

Jacobi yönteminde, her adımda A matrisinin a_{pq} ve a_{qp} elemanları sıfır yapılır. R1 ilk ortogonal matris ve $D_1=R^T_1 A R_1$ olsun. R_1 matrisi (4.18)' de görülen matrisin aynısıdır ve tek fark olarak $c=\cos\phi$ ve $s=\sin\phi$ olarak alınacaktır. Her adımda değişikliklerin p ve q satırları ile p ve q sütunlarında yapıldığının ispatlanması lazımdır. $B=AR_1$ olsun.

$$B = \begin{bmatrix} a_{11} & \cdots & a_{1p} & \cdots & a_{1q} & \cdots & a_{1n} \\ a_{p1} & \cdots & a_{pp} & \cdots & a_{pq} & \cdots & a_{pn} \\ a_{q1} & \cdots & a_{qp} & \cdots & a_{qq} & \cdots & a_{qn} \\ a_{n1} & \cdots & a_{np} & \cdots & a_{nq} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \qquad (4.37)$$

Bağıntı (4.37)' deki çarpım yapılacak olursa

$$\begin{aligned} b_{jk} &= a_{jk} && \text{eğer } k\neq p \text{ ve } k\neq q \\ b_{jp} &= ca_{jp} - sa_{jq} && j=1,2,\ldots,n \\ b_{jq} &= sa_{jp} + ca_{jq} && j=1,2,\ldots,n \end{aligned} \qquad (4.38)$$

olur. $D_1=R^T_1AR_1$ dönüşümü A matrisinin sadece p ve q satırları ile p ve q sütunlarının değişimine sebep olacaktır. A matrisinin diğer elemanları değişmeden D matrisine geçeceklerdir.

$$\begin{aligned} d_{jp} &= ca_{jp} - sa_{jq} && \text{eğer } j\neq p \text{ ve } j\neq q \\ d_{jq} &= sa_{jp} + ca_{jq} && \text{eğer } j\neq p \text{ ve } j\neq q \\ d_{pp} &= c^2a_{pp} + s^2a_{qq} - 2csa_{pq} \\ d_{qq} &= s^2a_{pp} + c^2a_{qq} + 2csa_{pq} \\ d_{pq} &= (c^2-s^2)a_{pq} + cs(a_{pp}-a_{qq}) \end{aligned} \qquad (4.39)$$

D_1 matrisinin diğer elemanları simetri ile bulunurlar. Burada önemli olan c ve s değişkenlerinin değerlerinin nasıl bulunacağıdır.

$$\theta = \cot 2\phi = \frac{c^2 - s^2}{2cs} \qquad (4.40)$$

Bu eşitlikten (4.39) eşitliğinin son denklemi kullanılarak

$$0=(c^2-s^2)a_{pq}+cs(a_{pp}-a_{qq}) \qquad (4.41)$$

elde edilir. Bu eşitlik düzenlenerek aşağıdaki biçime getirilir.

$$\theta=\frac{a_{qq}-a_{pp}}{2a_{pq}} \qquad (4.42)$$

$t=s/c=\tan\phi$ olur ve (4.42) eşitliği kullanılarak aşağıdaki denklem elde edilir.

$$t^2+2t\theta-1=0 \qquad (4.43)$$

Buradan

$$t=-\theta\pm(\theta^2+1)^{1/2}=\mathrm{sign}(\theta)/(|\theta|+\theta^2+1)^{1/2} \qquad (4.44)$$

olur ve buradan c ve s hesaplanabilir.

$$c=1/(t^2+1)^{1/2} \qquad (4.45)$$
$$s=ct$$

Bir çizgenin Laplace matrisinin öz değer vektörlerin bulunması ile nasıl bölmelenebileceğini ilk olarak Fiedler göstermiştir. Bunu çizgenin iki parçası arasında kalan ayrıtların ağırlık toplamının minimum olması gerektiğini kabul ederek ve çizgenin iki parçaya bölündüğünü düşünerek, bir parçadaki düğümlere +1 değeri atanmış ise diğerine -1 değeri atanarak, aşağıdaki fonksiyonun bu iki parça arasında kalan ayrıt sayısını verdiği ortaya çıkacaktır (4.46). Bu fonksiyonu kullanarak numerik yöntemler ile çizgenin nasıl bölmeleneceğinin teorisi şu şekilde olur.

$$f=\frac{1}{4}\sum_{E_{ij}}(x_i-x_j)^2 \qquad (4.46)$$

Çizgenin derece matrisi D ve bitişiklik matrisi de A olsun.

$$\sum_{E_{ij}}(x_i-x_j)^2=\sum_{E_{ij}}(x_i+x_j)^2-\sum_{E_{ij}}2x_ix_j$$
$$\Rightarrow \sum_{E_{ij}}2x_ix_j=\sum_{V_i}\sum_{V_j}x_iA_{ij}x_j=\sum_{V_i}x_i\sum_{V_j}A_{ij}x_j=x^TAx$$
$$\Rightarrow \sum_{E_{ij}}(x_i+x_j)^2=\sum_{E_{ij}}2=2|E|=\sum_{v_i}x_i^2d_i=x^TDx \qquad (4.47)$$

Buradan

$$\sum_{E_{ij}}(x_i-x_j)^2=x^TDx-x^TAx=x^T(D-A)x \qquad (4.48)$$

D-A matrisi bilindiği gibi Laplace matrisidir.

4.1.3. Özyinelemeli eş iki parçaya bölmeleme algoritması

Özyinelemeli (eş iki parça) iki eşit parçaya bölmeleme algoritması çok hızlı bir stratejidir. İki boyutlu bir düzlemdeki noktaların birden fazla kümeye, mümkünse eşit noktalar kümesine bölmelemektir ve bu bölümler yaklaşık olarak eşit sayıda noktalar içerirler. Bir noktanın, bir parçaya olan üyeliği, o noktanın iki boyutlu düzlemdeki yeri ile belirlenir. Bu metot iki yolla genellenebilir.

1. Bu metot daha fazla boyutlu düzlemlere genişletilebilir.
2. Elemanlar arasında lineer düzenlemesi tanımlı olan her düzlemde bu algoritma kullanılabilir.

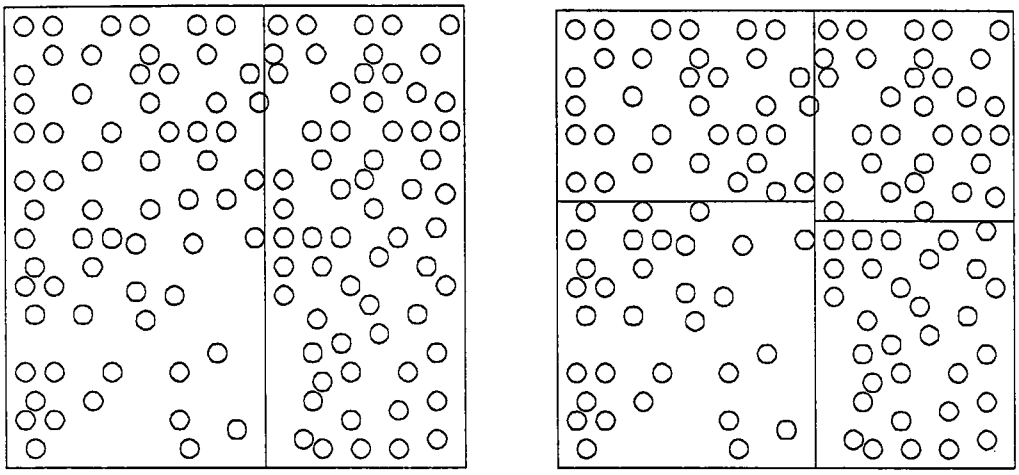
VLSI devrelerinin simülasyonlarında, transistörlerin simülasyonu bütün işlemlerin ağırlıklı olarak çekirdeğini oluşturur, bundan dolayı bu simülasyon hesaplamaları paralel ortamlarda bölmelenip işlemcilere paylaştırılması lazımdır, fakat bu yüklerin dengeli bir şekilde işlemcilere dağıtılmasına dikkat edilmelidir.

Şekil 4.2' de görüldüğü gibi sıralama, paralel olarak iki eşit parçaya bölme algoritmasının bloklarını oluşturmaktadır.

Özyinelemeli eşit parçalara bölme algoritmasının açıklanabilmesi için aşağıdaki örnek algoritmanın tekniğini göstermektedir.

İki boyutlu ve sonlu olan bir düzlemde, gelişigüzel noktalar olsun. Problem, verilen işlemcilere eşit sayıda nokta dağıtmaktır. Şekil 4.2' de 124 noktalı bir düzlem görülmektedir. İlk olarak, algoritma noktaları x-koordinatlarına göre sıralar ve iki eşit parçaya böler. Ondan sonra bu parçaları kendi içlerinde y-koordinatlarına göre sıralar ve her parçayı tekrar iki parçaya böler. Bu işlem istenilen parça sayısı elde edilene kadar devam ettirilir.

Hızlı bir sıralama algoritması, özyinelemeli eşit parçalara bölmeleme stratejisinin temelini oluşturur. Sıralama algoritması üzerinde işlem yapılan elemanların topolojisine bağlıdır. Genel olarak iki yönlü dizi topolojisi düşünülmektedir. Sıralama esnasında sadece hesapsal işlemler yapan birimler çalışacak



Şekil 4.2. 124 noktalı gelişigüzel bir çizgenin 2 ve 4 parçaya bölmelenmesi.

diğer birimler atıl kalacaktır. Bu da aynı anda birden fazla birimin sıralama algoritmaları için kullanılabileceğini gösterir. Paralel sıralama algoritması kendi içinde ardışıl hızlı sıralama (quick-sort) algoritmasını içerir. Paralel sıralama için paralel birleştirme algoritması (parallel mergesort) kullanılır.

Boyutu büyük olan sıralamalarda, mertebesi (O(nlogn))iyi olan hızlı sıralama algoritması kullanılır. Verimin iyi olmasına sebep olan uzak mesafeli yer değişimler hızlı sıralama algoritmasında mevcuttur.

İlk olarak dizinin medyanı bulunur. Bu diziyi iki alt diziye parçalar. Alt dizilerin içindeki elemanlar ya medyandan küçük ya da medyandan büyüktürler. Bu işlem böl ve yönet işleminin devam edebileceği noktaya kadar devam eder. Algoritma 4.1' de bu işlemi yapan sıralama algoritması görülmektedir.

Algoritma 4.1 : Ardışıl hızlı sıralama algoritması

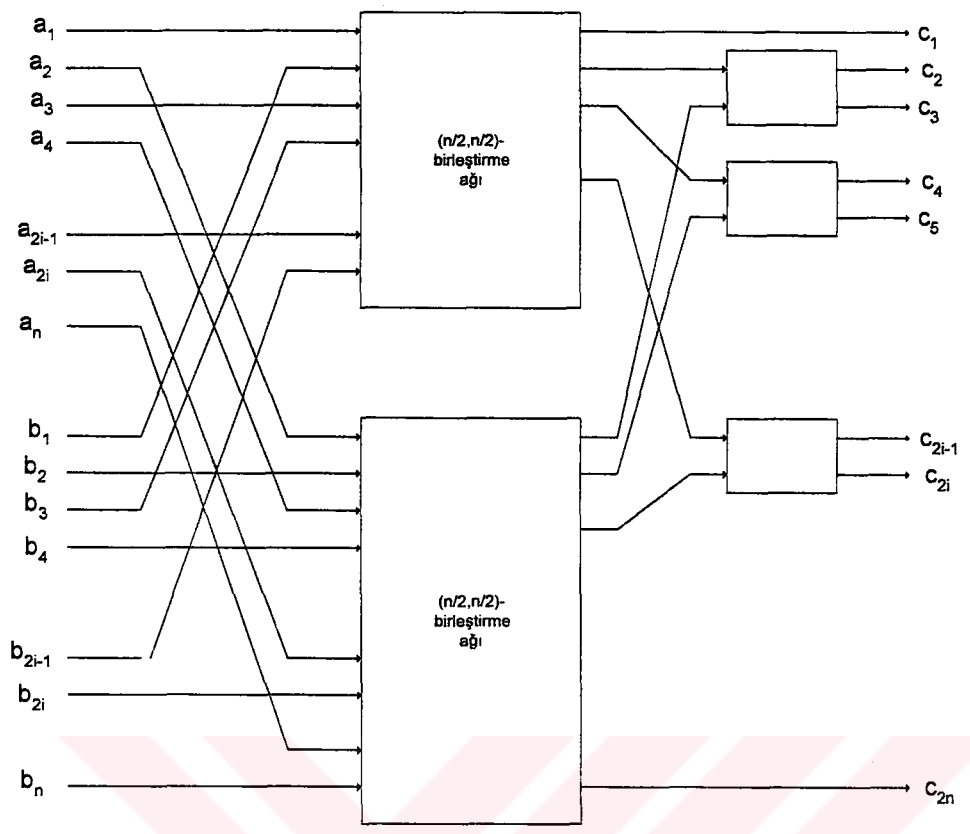
yordam hızlısırala(int l, int r, dizi a)

1. i=l; j=r; $x = \frac{a(l + r)}{2}$;

4. (i>j) olana kadar tekrarla

```
        4.1. (karşılaştır(a_i,x)=daha_küçük) olduğu sürece
             i=i+1;
        4.2.(karşılaştır(x,a_j)=daha_küçük) olduğu sürece
             j=j-1;
        4.3. eğer (i<=j) ise
             ElemanDeğiştir(a_i,a_j);
             i=i+1;
             j=j-1;
    5. eğer (l<j) ise hızlısırala(l,j,a);
    6. eğer (i<r) ise hızlısırala(i,r,a);
yordam sonu
```

Paralel sıralama algoritması Merge-Splitting metodunu temel alır [AKL, 1989], [DIMOV, v.d., 1994], [HULL, v.d., 1994], [ZOMAYA, 1996], [HOCKEY, v.d., 1988]. Başka paralel sıralama algoritmaları da kullanılabilir. Bu algoritma Tek-Çift Transpozisyon (Şekil 4.3) sıralama tekniğini temel olarak almıştır. Bunun sebebi ÇKÇV (çoklu komut çoklu veri - MIMD) makinelerde sıralanacak dizinin elemanlarının yarısı kadar işlemci içerirler. Eğer mimari değiştirilirse, bu durumda farklı sayıda işlemci içeren paralel ortamlar kullanılabilir. Bu algoritmanın temelinde çift sayıların sağa ve tek sayılar da sola gönderilmesi yatar. İşlem esnasında her işlemci iki tane veri dizisi içerirler, biri kendine gelen dizi, diğeri de kendi komşusu olan işlemciden gelen dizidir. Her işlemci üzerinde, çift sayıların en küçüğü ve tek sayıların da en büyüğü belirlenir. İşlemcilerde saklı olan veriler yer değiştirilirler. Bundan sonraki adım önceki adıma benzer fakat bir fark vardır. Tek indeksli işlemciler verisini sola gönderirler ve çift indeksli işlemciler verilerini sağa gönderirler. Lineer dizinin uçlarında yer alan işlemciler kendi verilerine herhangi bir tarafa göndermezler. Tek indeksli işlemciler üzerinde küçük sayılar ve çift indeksli işlemcilerde ise büyük sayılar saklanır. Eğer işlemci sayısı p ise, bir önceki sayfadaki adımlar p/2 kez tekrarlanırlar ki dizinin sıralanmış olması garanti edilmiş olsun.



Şekil 4.3. Tek-Çift ağı

Algoritma 4.2. Paralel birleştir-sırala algoritması

```
yordam BirleştirmeSıralamaParalel
     1.t=0;
          1.1. eğer (çift(t)) ise
               1.1.1. eğer (çift(m)) ise
                    SendRecv(sağa);
                    BirleştirDüşük(boyut);
                1.1.2. eğer (tek(m)) ise
                    SendRecv(sola);
                    BirleştirYüksek(boyut);
          1.2.
               1.2.1. ((m=İlkİşlemci) veya (m=SonİşlemciI)) ise
                    yer değişimi yok
               1.2.2.
                    1.2.2.1. eğer (çift(m)) ise
```

```
                    SendRecv(sola);
                    BirleştirYüksek(boyut);
            1.2.2.2. eğer (tek(m)) ise
                    SendRecv(sağa);
                    BirleştirYüksek(boyut);
    2. eğer t<DizidekiİşlemciSayısı ise,
        t=t+1 ve 1.1. adıma git.
yordam sonu
```

SendRecv(indeks) : kendi verisini indeks numaralı işlemciye gönderir ve kendi komşusunun verisini alır.

Algoritma 4.1 ve Algoritma 4.2' de görülen iki algoritmayı kullanarak özyinelemeli eşit parçalara bölmeleme algoritması Algoritma 4.3' te verilmiştir ve bu yordamın çağırılması

```
EşitParçaAlgo(0, p-1, seviye, n/p);
```

şeklinde olur ve seviye değişkeni dizinin 2^{seviye} parçaya bölüneceğini göstermektedir.

Algoritma 4.3 : Eşit parçalara bölme algoritması

```
yordam EşitParçaAlgo(İlkİşlemci, SonİşlemciSayısı, Boyut, Seviye,
                ElemanSayısı)
    hızlısırala(0,ElemanSayısı,Elemanlar, Boyut);
    OrtaNokta=(SonİşlemciSayısı-İlkİşlemci)/2;
    BirleştirmeSıralamaParalel(İlkİşlemci,SonİşlemciSayısı,Boyut);
    eğer seviye>1 ise
        SonrakiBoyut=(Boyut+1)%MaksBoyut;
        EşitParçaAlgo(İlkİşlemci,OrtaNokta,SonrakiBoyut,
                Seviye-1,ElemanSayısı);
        EşitParçaAlgo(OrtaNokta+1,SonİşlemciSayısı,SonrakiBoyut,
                Seviye-1,ElemanSayısı);
yordam sonu
```

Eşit parçalara bölmeleme algoritması hızlı olan stratejilerden biridir. Bu strateji çok boyutlu çizgelere uygulanabilir ve lineer mertebesi tanımlı olan gelişi güzel tanım kümelerinde de uygulanabilir.

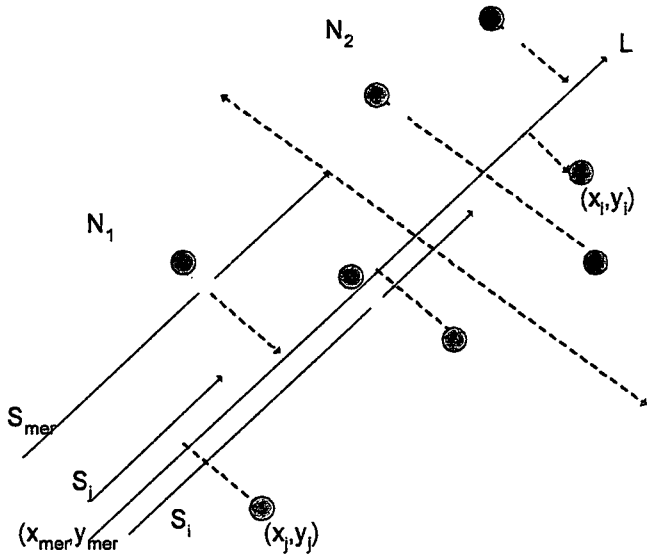
4.1.4. Eylemsizlik algoritması

Bu yöntem, geometrik bilgileri kullanan, basit ve hızlı olan bir algoritmadır. Bu yöntemde çizgenin yanında bu çizgenin düğümlerinin bir, iki veya üç boyutlu uzayda koordinat bilgisinin olması gerekmektedir. Düğümleri kütlesel nokta olarak kabul eden bu yöntem, 2D uzayda noktaları bir doğru ile iki parçaya böler, 3D uzayda ise bir düzlem ile noktaları iki parçaya böler. Bu işlem aşağıdaki algoritmaya göre yapılır.

1. L gibi bir doğru seçilsin ve bu doğrunun denklemi $a(x-x_{mer})+b(y-y_{mer})=0$. Bu doğru (x_{mer},y_{mer}) noktasından geçer ve $-a/b$ eğimine sahiptir. $a^2+b^2=1$ kabul edilmesi genelliliği bozmaz.
2. Her $n_i=(x_i,y_i)$ düğümü için $S_i=-b(x_i-x_{mer})+a(y_i-y_{mer})$ skalar çarpımı yapılır. S_i, (x_{mer},y_{mer}) noktasından (x_i,y_i) noktasının L doğrusuna iz düşümünün yapılmasında elde edilen noktaya olan uzaklıktır.
3. S_i' lerin S_{mer} medyanı bulunur.
4. (x_i,y_i) düğümü $S_i \leq S_{mer}$ şartını sağlayıp ve N_1 parçası içinde olsun ve $S_i > S_{mer}$ olan düğümler de N_2 parçasında olsun.

Bu anlatılan yöntem Şekil 4.4' da görülmektedir. Bu çizgede ayrıtlar gösterilmemiştir çünkü algoritmada çizgenin ayrıtları kullanılmamaktadır. Bunun yerine çizgedeki düğümlerin koordinat sistemindeki birbirlerine olan uzaklıkları önemlidir. Eğer çizgenin ayrıtları düğümler arasındaki bu uzaklıkla uyumlu ise bu durumda çizge bölmelendikten sonra ara kesitte kalan ayrıt sayısı az olacaktır.

Burada önemli olan L doğrusunun yönünün seçilmesidir. Eğer Şekil 4.4' da görülen çizge gibi düğümler dar ve uzun bir bölgede yer alıyorlarsa bu durumda L doğrusu bu alanın boyu yönündeki eksen yönünde olacaktır. Öyle bir (x,y) nokta seçilir ki bütün noktaların bir bütün olarak minimum eylemsizlik momentinin ekseni olmalıdır. Bundan dolayı bu metoda eylemsizlik yöntemi denilmektedir. a, b, x_{mer}, y_{mer} değerleri öyle seçilmelidir ki $a^2+b^2=1$ ve aşağıdaki değer minimum olmalıdır.



Şekil 4.4. Eylemsizlik yöntemi ile 2D' de çizge bölmeleme.

$$X_2 = \sum_{i=1}^{|N|} (x_i - x_{mer}),\ Y_2 = \sum_{i=1}^{|N|} (y_i - y_{mer}) \text{ ve}$$

$$XY = \sum_{i=1}^{|N|} (x_i - x_{mer})(y_i - y_{mer})$$

$$\sum_{i=1}^{|N|} (\text{nokta nokta olan doğruların uzunluğu})^2$$

$$= \sum_{i=1}^{|N|} ((x_i - x_{mer})^2 + (y_i - y_{mer})^2 - (-b(x_i - x_{mer}) + a(y_i - y_{mer}))^2)$$

$$= a^2\left(\sum_{i=1}^{|N|} (x_i - x_{mer})^2\right) + b\left(\sum_{i=1}^{|N|} (y_i - y_{mer})^2\right) + 2ab\left(\sum_{i=1}^{|N|} (x_i - x_{mer})(y_i - y_{mer})\right)$$

$$= a^2X_2 + b^2Y_2 + 2abXY$$

$$= \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} X_2 & XY \\ XY & Y_2 \end{bmatrix} \begin{bmatrix} a \\ a \end{bmatrix} \qquad (4.49)$$

$$= \begin{bmatrix} a & b \end{bmatrix} M \begin{bmatrix} a \\ a \end{bmatrix}$$

x_{mer} ve y_{mer} değerlerinin şu olduğu gösterilebilir.

$$x_{mer} = \sum_{i=1}^{|N|} \frac{x_i}{|N|} \text{ ve } y_{mer} = \sum_{i=1}^{|N|} \frac{y_i}{|N|} \qquad (4.50)$$

(x_{mer}, y_{mer}) noktası kütlenin ağırlık merkezidir ve [a b], M matrisinin en küçük öz değerine denk düşen bir birim öz değer vektörüdür.

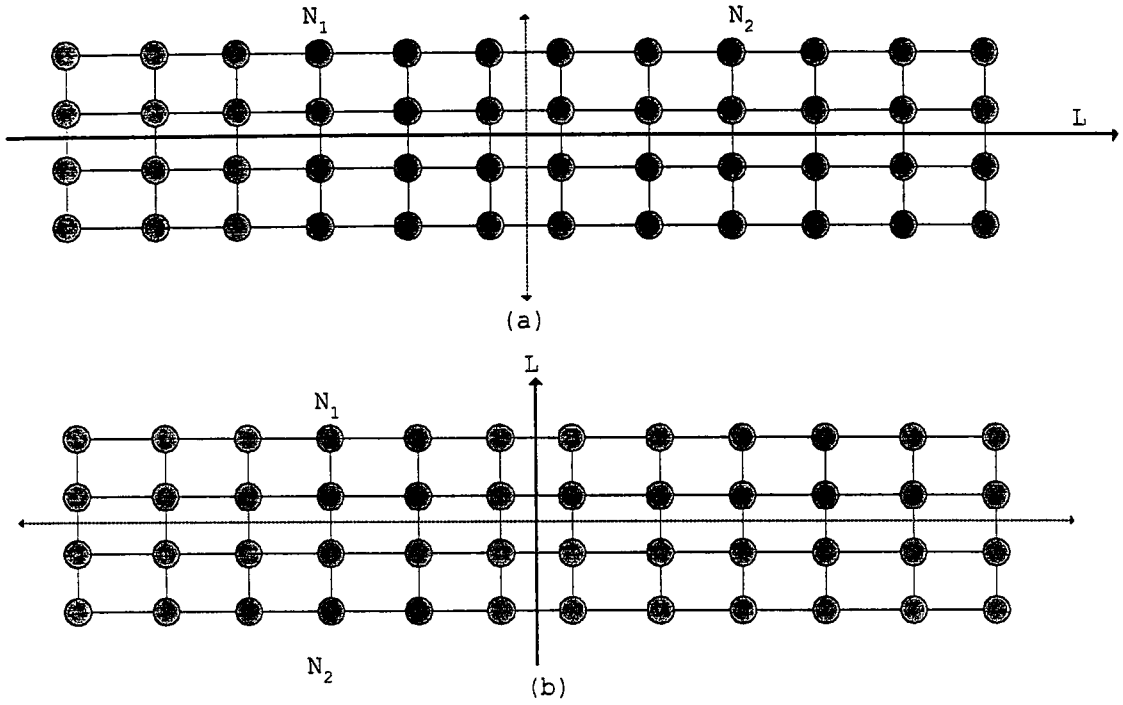
(x,y) düğüm seçiminin iyi yapılması gerekmektedir. Eğer bu düğüm seçimi iyi yapılmaz ise Şekil 4.5-b' de görüldüğü gibi hiç de iyi olmayan bir sonuç elde edilir. Eğer bu seçim iyi yapılırsa, Şekil 4.5-a' da görüldüğü gibi daha iyi bir sonuç elde edilir.

3D ve daha yüksek boyutlu uzaylarda yöntem aynı şekilde işlemektedir, fakat 3D için 3x3 boyutlarında bir matris elde edilir ve bu matrisin en küçük öz değerine denk düşen birim öz değer vektörü bulunur. nD için ise nxn boyutlarında bir M matrisi bulunması gerekir ve aynı zamanda bu matrisin en küçük öz değerinin birim öz değer vektörü bulunmalıdır.

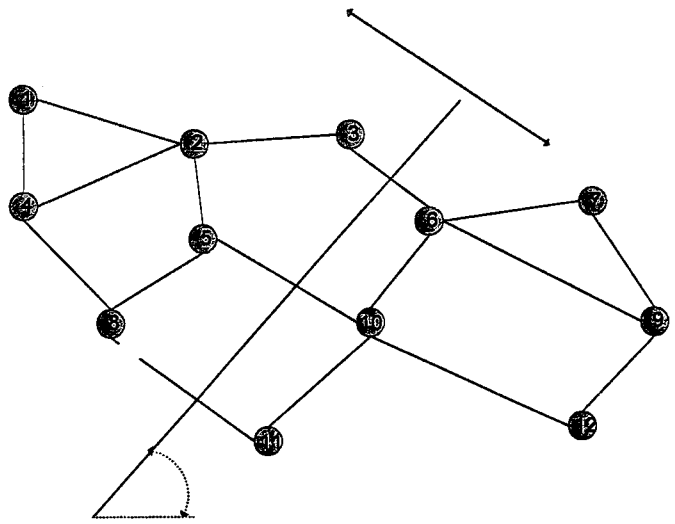
Özellikle spektral yöntem düşünüldüğü zaman bu yöntemin kalitesi düşük bölmeleme yaptığı görülmektedir. Bu yöntemden sonra iyileştirme algoritmalarından biri uygulandığı zaman daha kaliteli bölmeleme sonucu elde edilebilmektedir. Eğer geliştirme algoritmalarından Kernighan-Lin uygulanırsa bunun sonucunda elde edilen bölmeleme sadece spektral bölmeleme ile elde edilen sonuçtan daha iyidir. Fakat Spektral algoritmadan sonra iyileştirme algoritması uygulandığı zaman daha iyi sonuç elde edilmektedir.

Bir doğru ile noktaları iki eş parçaya bölmelemek için başka bir yöntem de orjinden geçen bir doğru seçmektir. Bu doğrunun belirlenen bir eksen ile yaptığı açı, yük dengeleme şartına göre değiştirilir veya değiştirilmez. Bu yöntemde, özyinelemeli eş iki parçaya bölme stratejisi kullanılabilir ve bu strateji hızlı bir stratejidir (LASZEWSKI, 1993). Bir noktanın (düğümün) bir düzleme olan üyeliği o noktanın iki boyutlu olan bu düzlemdeki yeri ile belirlenir.

Şekil 4.6 bu algoritmanın nasıl çalıştığını çok iyi olarak göstermektedir. İlk olarak seçilen bir eksene göre bir açı değeri seçilir ve bu açı değeri ile düzlemde bir doğru çizilir. Eğer bu doğru noktaları iki eş kümeye bölebiliyorsa işlem durdurulur, aksi halde eleman sayısı fazla olan kümeye doğru



Şekil 4.5. Eylemsizlik yöntemine göre aynı ağın iki farklı şekilde bölmelenmesi, (a) iyi (x,y) seçimi, bölmeleme kesmesinin boyutu 4' tür. (b) Kötü (x,y) seçimi, bölmeleme kesmesinin boyutu 12' dir.



Şekil 4.6. Bir doğru ile noktaları iki eş parçaya bölme.

gidecek şekilde açı değeri değiştirilir. Tekrar eleman sayıları kontrol edilir. Bu işlem en iyi çözüm elde edilinceye kadar devam ettirilir. i düğümünün düzlemdeki yeri $(x_i, y_i)'$ dir ve her noktaya verilen değer (4.51) eşitliği ile hesaplanır.

$$x_i' = \cos(\alpha)x_i - \sin(\alpha)y_i \qquad (4.51)$$

(4.51) eşitliği x ekseni etrafında α açısı kadar döndürme yapar.

4.1.5. Basit çizge indirgeme

Çizge bölmeleme işleminin hızlandırılması için düğüm sayısı çok olan çizgeler için düğüm sayısı daha az olan ve bu çizgeden elde edilen bir çizgenin elde edilmesi ile işleme başlanır. Basit çizge indirgeme algoritması, çizgedeki düğümlerin komşuluk ilişkisinin kullanılmasını sağlar. Basit çizge indirgeme algoritmasının amacı, çizgedeki düğüm sayısını azaltarak daha önce sözü geçen algoritmalardan birinin kullanılmasıdır. İki strateji mümkündür:

1. İlk strateji orijinal çizgenin bir alt çizgesini alarak bu alt çizge üzerinde iyileştirme algoritmalarını kullanarak çizgenin bölmelenmesini sağlamaktır.
2. İkinci strateji olarak, çizgenin bazı düğümleri hiperdüğüm denilen düğümlere dönüştürülür. Hiperdüğümler birden fazla düğümün birleştirilmesinden elde edilirler. Çizge indirgendikten sonra başlangıç bölmeleme işlemi yapılır ve ondan sonra iyileştirme algoritmaları bu parçalara uygulanır.

Çizgenin indirgenmesinde (4.52) bağıntı kullanılır.

$$R:(V_1, E_1, w_1) \rightarrow (V_2, E_2, w_2) \qquad (4.52)$$

(4.52) bağıntısındaki düğümler ve ayrıtlar kümeleri $|V_2|<|V_1|$ ve $|E_2|<|E_1|$ özelliklerini sağlarlar. R bağıntısının derecesi bir hiperdüğümde en fazla kaç tane düğüm birleştirileceğini göstermektedir.

N(v) kümesi, v düğümünün komşu düğümlerini içeren bir küme olsun ve $N_r(v)$ kümesi ise v düğümünden en fazla uzunluğu r yollarla v' ye bağlı olan düğümlerin kümesi olsun. Bu komşuluk kümeleri (4.53) bağıntısındaki öz yinelemeli yöntem ile elde edilebilirler.

$$N_r(v) = \begin{cases} N(v) & \text{eğer } r = 1 \text{ ise} \\ N_{r-1}(v) & \text{eğer } r > 1 \text{ ise} \end{cases} \qquad (4.53)$$

Gelişi güzel bir düğüm seçilir ve bu düğümün $N_r(v)$ kümesi gelişi güzel ve sabit r değeri için oluşturulur öyle ki $|N_r(v)| \geq R-1$ olsun. Seçilen r-1 düğüm ile birlikte v düğümü bir hiperdüğüm oluşturur. Hiperdüğüm içinde birleştirilen düğümler ve bu düğümlere bağlı olan ayrıtlar orijinal çizgeden $G=(V_1,E_1,w_1)$ silinirler. Bundan sonra geriye kalan orijinal çizgenin alt çizgesinden bir düğüm seçilir ve yukarıda açıklanan işlem tekrar edilir. Bu işlemlere çizgede düğüm sayısı istenilen seviyeye indirgenene kadar devam edilir.

Şekil 4.7' da görüldüğü gibi, bu elde edilen ve hiperdüğümlerden oluşan çizgenin ayrıtları güncelleştirilir. Hiperdüğümler oluşturulurken yeni çizgede paralel ayrıtlar ve tek çevre ayrıtlar oluşabilir. Tek çevre olan ayrıtlar doğrudan silinir ve paralel ayrıtlardan biri bırakılır diğerleri silinir. Diğerleri silinmeden önce bu paralel ayrıtların hepsinin ağırlıkları toplanır ve kalan ayrıta atanır. Elde edilen çizge basit çizgedir. Bu basit ve düğüm sayısı az olan çizgenin bölmelenmesi basit olur ve iyileştirme algoritmalardan biri bu sonuca uygulanır.

Bu yöntemde her hiperdüğümde eşit sayıda düğüm olacaktır diye bir garanti yoktur. Yük dengeleme şartının sağlanması için başka yollara baş vurulması gerekir. Bu problem şu işlemlerle çözülebilir.

1. Basit bir iyileştirme (greedy algoritması) algoritması elde edilen sonuca uygulanır.
2. Gelişi güzel bir çizge indirgeme algoritması uygulanır.
   a) Çizge indirgeme işleminden sonra, parçalar arasında düğümlerin transferi ile yük dengeleme şartı sağlanır.

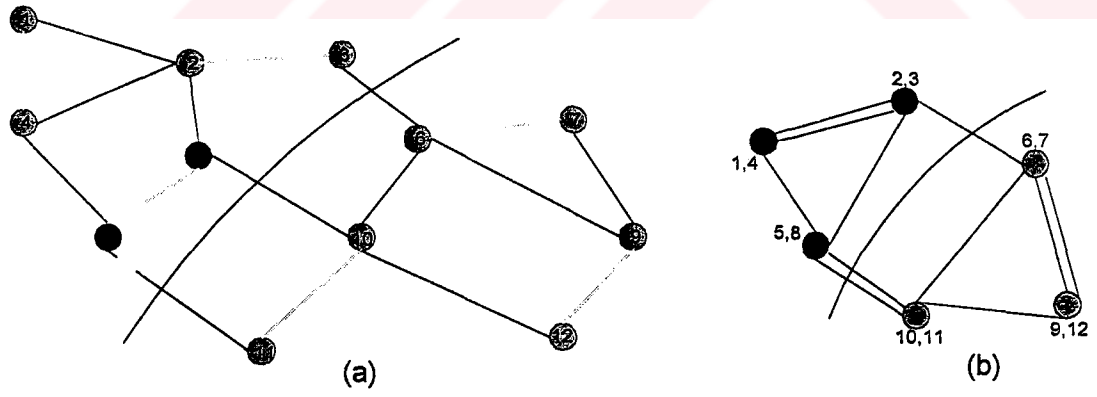
b) Bu işlemden sonra sonuç optimal olmayabilir. Bundan sonra iyileştirme algoritması uygulanır.

Bu yöntemde, çok dikkat edilmesi gereken bir nokta vardır; o da hiperdüğüm olarak birleştirilen düğümlerin çizge bölmelendikten sonra aynı parçaya ait olmasıdır. Bundan dolayı bütün düğümlere iyileştirme algoritmasının uygulanmasına gerek yoktur. İyileştirme algoritmasının kesim düğümlerine uygulanması yeterlidir. Kesim düğümleri kümesi V_{kesim} ve kesim ayrıtları kümesi E_{kesim}; M_p bağıntısı hangi düğümün hangi parçaya ait olduğunu belirlesin ve çizge m tane parçaya bölmelenmiş olsun ve (4.54) bağıntısı hangi düğümün hangi parçaya ait olduğunu vermektedir. (4.55) bağıntısında kesim ayrıtlar kümesini vermektedir ve (4.56) bağıntısı ise kesim düğümler kümesini vermektedir.

$$M_p: V\{P_a \mid a \in [1,m]\} \text{ ile } M_p(v)=P_a \Leftrightarrow v \in P_a \qquad (4.54)$$

$$E_{kesim}(P)=\{(v_i,v_j) \mid (v_i,v_j) \in E \text{ ve } M_p(v_i) \neq M_p(v_j)\} \qquad (4.55)$$

ve



Şekil 4.7. Basit çizge indirgeme, R=2, (a) orijinal çizgenin kendisi ve hiperdüğümlerin belirlenmesi, (b) bir seviye çizgenin indirgenmiş ve ayrıtların güncelleştirilmiş hali, paralel ve tek çevre ayrıtlar silinmeden önceki hali.

$$V_{kesim}(P)=\{v_i \mid v_j \in V \text{ ile } (v_i,v_j)\in E_{kesim}(P)\} \quad (4.56)$$

4.1.6 Çok seviyeli çizge bölmeleme

p-yollu çizge bölmeleme problemi şu şekilde tanımlanabilir. G=(V,E), |V|=n olan bir yönlendirilmemiş çizge olsun. Bu çizgenin düğümleri öyle p tane V_1, V_2, ..., V_p kümeye ayrılır ki

$$\forall i,j \; V_i \cap V_j=\varnothing,\; i\neq j,\; |V_i| \cong n/p \text{ ve } \bigcup V_i=V \quad (4.57)$$

ve uç düğümleri farklı parçalar içinde olan ayrıt sayısı da minimize edilir [ZECEVIC, vd., 1994], [SANCHIS, 1989], [ZHOU, 1993], [KARYPIS, vd., 1996], [LASZEWSKI, 1991], [SCHLOEGEL, vd., 1997], [HENDRICKSON, vd., 1995]. (4.54) bağıntısı bu problem için, (4.58) bağıntısındaki gibi değiştirilir.

$$M_p:V\{P_a \mid a\in[1,p]\} \text{ ile } M_p(v)=P_a \Leftrightarrow v\in P_a \quad (4.58)$$

P, n tane elemanı olan bir vektördür ve P[v], 1 ile p arasında bir sayı olur. Uç noktaları farklı parçalarda olan ayrıtların sayısına kesim-ayrıt denir (KARYPIS, vd. 1996).

Özyinelemeli iki eş parçaya bölemeleme algoritması p-yollu çizge bölmeleme probleminde kullanılabilir. İlk etapta çizge 2 parçaya bölmelenir, bundan sonra her parça kendi içinde iki parçaya bölmelenir ve bu işlem log(p) adım tekrarlanınca p tane parça elde edilir. Böylece p-yollu çizge bölmeleme işlemi 2-yollu çizge bölmeleme işleminin bir seri olarak tekrarından ibarettir. Bu yöntem her zaman iyi sonuç verdiğinden dolayı değil, basit olduğundan dolayı kullanılır. Çünkü bazı durumlarda bu yöntem iyi sonuç vermeyebilir.

Çok seviyeli çizge bölmeleme stratejisi çok basittir. İlk önce G=(V,E) çizgesi eğer çok sayıda düğüme sahipse, istenilen düğüm sayısı elde edilinceye kadar kabalaştırılır ve bu yeni çizgenin iki eş parçaya bölmeleme algoritması ile bölmelenmesi yapılır (başlangıç bölmeleme), daha sonra bu parçalar orijinal çizge elde edilinceye kadar açılır ve her safhada iyileştirme

algoritmaları uygulanır. Şekil 4.8' da bu yöntemin görsel şekli verilmiştir.

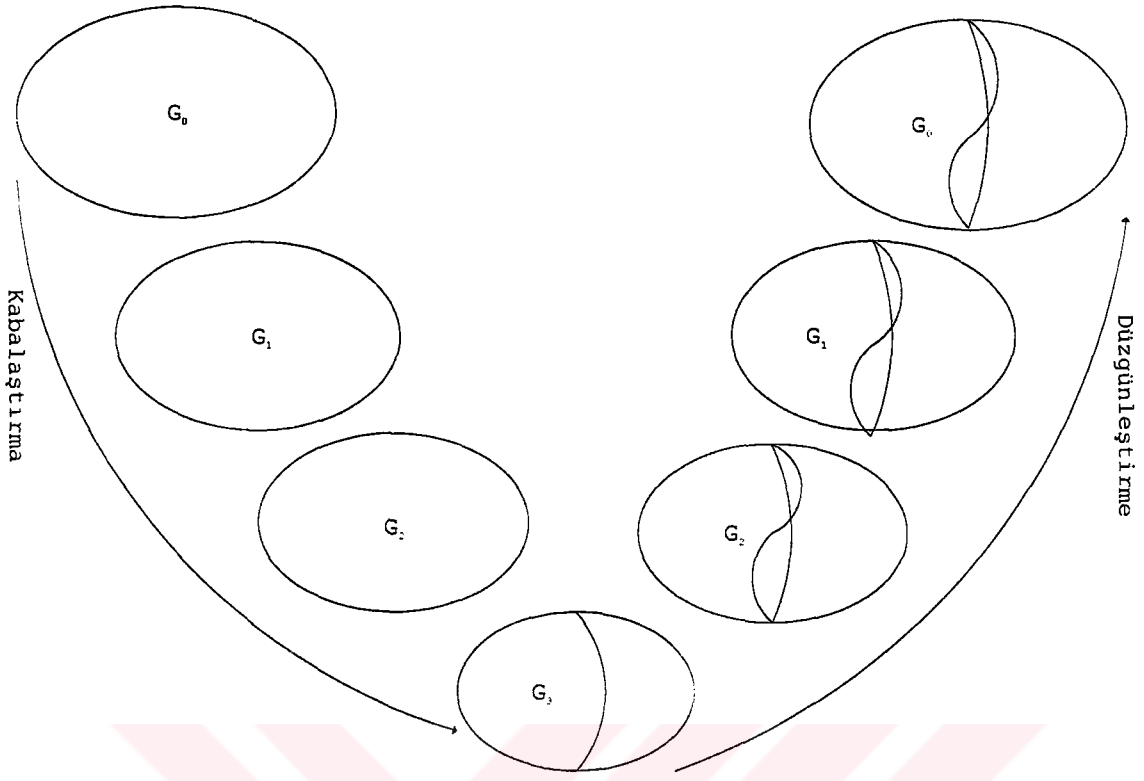
Kabalaştırma safhasında orijinal çizgeden $G_0=(V_0,E_0)$ daha kaba bir çizge $G_1=(V_1,E_1)$ elde edilir öyle ki $|V_0|>|V_1|$' dir. Bu işlem $|V_i|>|V_{i+1}|$ şartı göz önünde tutularak devam ettirilir. V_{i+1} kabalaştırılmış çizgenin G_{i+1} düğümler kümesi ve V_i kendisinden kabalaştırılmış çizge elde edilen G_i çizgenin düğümler kümesidir. G_{i+1} çizgesi G_i çizgesinde maksimum eşleştirme kümesi bulunarak, eşleşen düğümler bir hiperdüğüme indirgenerek elde edilir. Eşleştirme yöntemi hem bölmelemenin kalitesini ve hem de kabalaştırma safhasında geçecek zamanı etkiler. Bunlardan biri gelişi düzel eşleştirme algoritmasıdır. Bu algoritmada düğümler gelişi güzel olarak ele alınır ve eşleştirilmemiş düğümler kendisine komşu olan ve eşleştirilmemiş bir düğüm ile eşleştirilirler. Bu eşleştirmeden daha iyi olan diğer bir eşleştirme algoritması ise en ağır ayrıt eşleştirme algoritmasıdır. Düğümler gelişi güzel ele alınır ve bu düğüm kendisine komşu olan düğümlerden eşleşmemiş olan ve ayrıt ağırlığı en fazla olan düğüm ile eşleştirilirler. Bu işlem eşleştirilebilecek düğüm kalmayana kadar devam ettirilir.

Eğer $G_i=(V_i,E_i)$ çizgesinin maksimum eşleştirmesi M_i ve bu çizgenin eşleştirme ayrıtlarının büzüştürülmesi ile $G_{i+1}=(V_{i+1},E_{i+1})$ elde edildiyse,

$$W(E_{i+1})=W(E_i)-W(M_i) \qquad (4.59)$$

(4.59) bağıntısı elde edilir ve G_{i+1} çizgesinin toplam ayrıtlar ağırlığı G_i çizgesinin ayrıtların toplam ağırlığından toplam eşleştirme ağırlığının çıkarılması ile elde edilir. Eğer eşleştirme maksimum ise, bu durumda G_{i+1} çizgesinin toplam ayrıtlar ağırlığı daha az olacaktır, bu da demektir ki kesim ayrıtlarının ağırlığı az olacaktır.

Değiştirilmiş ağır ayrıt eşleştirme algoritması ise, kabalaştırılmış çizgelerin ortalama derecesinin daha az olmasını sağlamaya çalışır. Bu algoritmada düğümler gelişi güzel olarak ele alınır. v böyle bir düğüm olsun ve H eşleşmemiş v düğümünün en ağır ayrıtlarla bağlanmış olan komşularının kümesi olsun. Her



Şekil 4.8. Çok seviyeli çizge bölmeleme yöntemi.

$u \in H$, W_{v-u} u düğümünü v düğümünün komşularına bağlayan ayrıtların ağırlıkları toplamı olsun. Bu algoritmada v düğümü W_{v-u} ağırlığı en fazla olan düğüm ile eşleştirilir.

Bu algoritmanın sonucunda kabalaştırılmış çizgede daha az ayrıt ve ortalama ayrıt ağırlığı daha fazla olan bir kaba çizge elde edilir. Daha ileri ki seviyelerde eşleştirmede yer alan ayrıtların ağırlığı artar, bu da bu algoritmanın daha etkili olmasını sağlar. Eğer çizgenin ayrıtlarının hepsinin ağırlıkları aynı ise bu algoritma bir önceki algoritmadan daha etkilidir. Bu tip bir çizgenin ilk adımında gelişi güzel eşleştirme algoritmasının sonucu ile ağır ayrıt eşleştirme algoritmasının sonucu aynı olur, çünkü bütün ayrıtlar aynı ağırlıktadır. Fakat değiştirilmiş ağır ayrıt eşleştirmesi daha iyi sonuç verir.

Çizgenin kabalaştırılmış çizgesi elde edildikten sonra, spektral yöntemine göre başlangıç bölmeleme yapılır. Elde edilen kabalaştırılmış çizgenin düğüm sayısı orijinal çizgenin düğüm

sayısından az olduğundan bölmeleme işlemi orijinal çizgenin bölmelenmesine göre daha az zaman alır.

Eğer kabalaştırılmış çizge G_p ise düzgünleştirme işlemi G_p, G_{p-1}, ..., G_0 şeklinde devam eder. Her parçada bulunan hiperdüğümlerin içindeki düğümler de o parçaya ait olduğundan, her hiperdüğüm için o hiperdüğümü oluşturan düğümler yerine konur ve ayrıtlar tekrar düzenlenir. Her düzgünleştirme adımından sonra her parçaya iyileştirme algoritması uygulanır. İyileştirme algoritmalarının amacı, eğer bir düğümün bir parçadan diğer parçaya geçişi kesim ayrıt sayısını azaltıyor ve yük dengeleme şartını bozmuyorsa, bu düğüm o parçaya transfer edilir.

Çok seviyeli çizge bölmeleme algoritmalarında iki tip paralellik kullanılır. İlk paralel algoritma, problem çözümünün özyinelemeli doğal yapısından gelmektedir. İlk olarak bir işlemci orijinal çizgenin iki eş parçasını bulur, ondan sonra iki işlemci her biri bir parçanın iki eş parçalarını bulurlar ve bu işlem p tane parça elde edilinceye kadar devam ettirilir. Bu işlemde log(p) tane işlemciye ihtiyaç duyar ve toplam bölmeleme işlemini seri bölmeleme zamanının log(p) ' ye bölünmesi ile elde edilir.

İkinci paralel algoritma ise iki eş parçayı elde etme adımı safhasında kullanılır. Bu algoritmada orijinal çizgenin bölmelenmesi tek işlemci yerine birden fazla işlemci üzerinde paralel olarak yapılır. Bölmeleme adımı paralelleştirildiğinden dolayı, bu algoritmanın toplam zamanı birinci tipteki algoritmadan daha iyidir, çünkü bölmeleme algoritmalarında zamanın çoğu bölmeleme işlemi için harcanır.

Kabalaştırma esnasında birden fazla kaba çizge elde edilir. $G_{i+1}=(V_{i+1},E_{i+1})$ kaba çizgesi için, $G_i=(V_i,E_i)$ kaba çizgesinin maksimum eşleştirmesi M_i bulunur ve bu eşleştirmedeki ayrıtlara göre hiperdüğümler oluşturulur ve ayrıtlar düzenlenir. Bu adım, bu çok seviyeli algoritmanın en fazla zaman alan kısmıdır ve bu adımın paralelleştirilmesi gerekir.

Gelişi güzel seri eşleştirme algoritması basit ve etkilidir. Paralel olarak eşleştirmenin hesaplanması zordur (özellikle dağıtık hafızalı sistemlerde). Seri eşleştirme

algoritmasının paralelleştirilmesi, işlemciler arasında fazla iletişim gerektirmektedir. Örneğin gelişi güzel eşleştirme algoritması paralelleştirilecek olunursa, her işlemci orijinal çizgenin gelişi güzel bir alt çizgesini içerir. Her yerel v düğümü için her işlemci eşleştirmede olacak bir ayrıt (v,u) seçer. Burada yapılması gereken, yerel olarak v ve u düğümlerini barındıran işlemcilerin iletişimi sonucunda bu düğümlerin eşleşmenin içinde olup olmadığının kontrol edilmesinden sonra eğer eşleşmenin içinde değillerse eşleşmeye eklenirler. Burada dikkat edilmesi gereken bir nokta vardır, o da işlemciler arasında yarış şartının (race condition) ortaya çıkmaması gerekir. Bir işlemci (v,u) ayrıtını kontrol ederken, diğer bir işlemci (u,w) gibi bir ayrıtı kontrol etmek isteyebilir, ancak bu ayrıtlardan biri eşleşmenin içinde yer alabilir. Maksimum bağımsız küme bulunarak eşleştirme yapan algoritmalar da vardır ve bu tip bir algoritma da paralelleştirilebilir. Her durumda işlemciler kendi üzerinde bulunan yerel düğümün komşu düğümleri için diğer işlemcilerle iletişime geçmesi gerekmektedir. Bu da eşleştirme safhasında algoritmanın toplam zamanının önemli bir kısmı burada harcanmaktadır. Eşleştirme yapıldıktan sonra, eşleştirmede yer alan düğümlerin büzüştürülmesi gerekir. Eşleştirmede yer alan her ayrıtın uçları farklı işlemciler üzerinde ise ayrıt büzüştürmeden önce düğüm transferi yapılır. İşlemciler arası iletişim zamanının azaltılması için işlemcilere, işlemciler arasında kalan ayrıtların bir üst sınırı göz önünde tutularak düğümler dağıtılmalıdır. Bu yukarıda verilen algoritmalarda ki iletişim için harcanan zamanın azaltılması anlamına gelir. Bu da başlangıçta bölmeleme işleminin yapılması anlamına gelmektedir.

Eşleştirmeyi hesaplamak için başka bir algoritma, her işlemciye, düğümler kümesinin bir alt kümesi dağıtılır ve bu işlemciler kendi üzerinde bulunan düğümler arasında bir eşleştirme yaparlar. Bu algoritmanın avantajı, eşleştirme yapılırken işlemciler arasında iletişime başvurulmaz ve komşuluk listesinin işlemciler arasında transferine ihtiyaç duyulmaz. Bu algoritmanın problemi ise her işlemci üzerinde az sayıda düğüm olması ve bu işlemciler ile düğümler arasında bir eşleştirme

yapmalarıdır. Komşuluk listesinin transferine gerek olmadığı halde, her işlemci kendisi üzerinde olan düğümlerin komşu düğümlerinin eşleştirmede ki durumları hakkında bilgi sahibi olmaları kabalaştırma safhasının iyi yapılmasını sağlar. Bu da işlemciler arasında iletişime sebep olur.

Ayrıtların büzüştürülmelerinden sonra elde edilen kabalaştırılmış çizgenin başlangıç bölmelenmesi yapılır. Bölmeleme işlemi için daha önce verilen algoritmalardan biri kullanılabilir.

Düzgünleştirme esnasında, kabalaştırılmış çizgenin bölmelenmiş halinin orijinal çizge üzerine izdüşümü alınırken G_p, G_{p-1}, ..., G_0 safhalarından geçer. Her izdüşümü alma adımından sonra elde edilen parçalara iyileştirme algoritması uygulanır. Nasıl ki kabalaştırma safhasında paralel algoritmalar kullanılabildiyse, düzgünleştirme safhasında da paralel algoritmalar kullanılabilir.

4.2. İyileştirme Algoritmaları

İyileştirme algoritmaları, çizge bölmeleme işlemi yapıldıktan sonra daha iyi sonuç elde etmek için çözüme uygulanan algoritmalardır. Bu algoritmalar çözümü optimal çözüme yaklaştırırlar veya optimal çözümü bulurlar.

4.2.1. İki optimal iyileştirme algoritması

Bu algoritmada, ayrı iki parçada birer düğüm seçilir ve bu düğümlerden hangisinin diğer parçaya aktarılması yük dengeleme şartını bozmuyor ve kesim ayrıt ağırlığını düşürüyorsa, o düğüm belirlenen parçaya atanır [LASZEWSKI, 1993]. Bundan dolayı bu algoritmaya İki optimal iyileştirme algoritması denir. Bu algoritma Algoritma 4.4' de görülmektedir ve Algoritma 4.5' te ise çizge indirgeme ve iki-optimal algoritmasının greedy yöntemi ile yapılmış şekli görülmektedir. Bu iki algoritma da yerel

olarak çalışırlar ve kesim bölgesinden uzak olan başka bir noktada kesim yapılırsa, daha iyi sonuç elde edilebilir.

Algoritma 4.4 : Çizge indirgeme ve iki optimal algoritmalarının birlikte uygulanması.

yordam Indirgeme_2_Optimal
    $G \xrightarrow{R} G'$ indirgemesi yap
    Gelişi güzel olarak çizgeyi bölmele
    2_Optimal$(G'=(V',E',w'),P'_p)$ uygula
    P'_p P_p' ye genişlet
    $P_{a\epsilon[1,p]}$ arasında yük dengelemeyi geliştir
    İki Optimal$(G_{kesim}=(v_{kesim},E,w),P_p)$ uygula
Yordam Sonu

Algoritma 4.5: Çizge indirgeme ve iki optimal algoritmalarının Greedy yöntemi ile çözümü.

yordam İndirgeme-İki-Optimal
    ardışıl başla
    $G \xrightarrow{R} G'$ indirgemesi yap
    İndirgenmiş çizge için gelişi güzel bir bölmeleme
        yap.
    greedy$((G'=(V',E',w'),P'_p))$ uygula
    P'_p P_p' ye genişlet
    Greedy$(G_{kesim}=(V_{kesim},E,w),P_p)$ uygula
Yordam Sonu

4.2.2 Kernighan-Lin iyileştirme algoritması

Spektral çizge bölmeleme algoritmaları diğer algoritmalara göre daha iyi sonuç vermektedir. Bunun sebebi, bu algoritma, çizgede umut verici bir bölge bulmaktadır. Bu algoritmanın global bir güçlülüğü vardır. Bununla birlikte bu global güçlülük yerel zayıflık ile birleştirilir. Bundan dolayı bu tip bir

algoritmanın sonucuna yerel iyileştirme yapan bir algoritmanın uygulanması daha mantıklı ve doğru olur.

Bu tip iyileştirme algoritmalardan bir diğeri Kernighan ve Lin tarafından oluşturulan Kernighan-Lin algoritmasıdır. Bu algoritma, başlangıçta iyi bir bölmelemeyle başlatılmadıysa, yerel cevapları bulur, fakat yerel cevap optimal cevaptan çok uzak olabilir. Spektral bölmelemenin global güçlülüğü ve Kernighan-Lin (KL) algoritmasının yerel çözümlerde güçlülüğü, ikisinin bir arada kullanılması sonucunun herhangi birinin tek başına kullanılmasından daha iyi çözüm verir.

Algoritma 4.6' da görülen KL algoritmasının dış döngüsü geliştirilebilecek parça keşfedildiği sürece devam eder. Bu dış döngü başlarken her düğüm için q tercih değeri hesaplanır ve bu hesaplamanın nasıl yapıldığı (4.60) bağıntısında görülmektedir.

$$q(V_i) = \sum_{(V_i,V_j)\in E} \begin{cases} 1 & \text{eğer } P(V_i) \neq P(V_j) \\ -1 & \text{diğer durumda} \end{cases} \qquad (4.60)$$

$P(V_i)$: o anda incelemede olan düğümün parça numarasıdır. İçeride başka bir döngü vardır ve bu döngü iki tane düğümün parçalarını değiştirir ve tercih değerlerini tekrar hesaplar. Genelde tercih numarası en büyük olan düğümün parçası değiştirilir ve bir daha bu düğüm hesaba katılmaz. Her düğüm değişimi sonucunda parçalar değerlendirilir ve en iyisi tutulur.

Algoritma 4.6 : İki parçalı bölmeleme için KL iyileştirme algoritması.

```
Yordam KL
    Daha iyi bir bölmeleme bulunmayana kadar devam et
        İyiBölmeleme=ŞimdikiBölmeleme
        S1=Parça 1' deki düğümler
        S2=Parça 2' deki düğümler
        Bütün Vi için
            q(i) tercih değerlerini hesapla
        S1≠Ø ve S2≠Ø olduğu sürece
            Vi=S1' deki en büyük tercih değerli düğüm
            P(Vi)=2
```

$S_1=S_1\backslash V_i$

V_i düğümünün komşu düğümlerinin tercih
değerlerini hesapla

$V_j=S_2'$ deki en büyük tercih değerli düğüm

$P(V_j)=1$

$S_2=S_2\backslash V_j$

Vj düğümünün komşu düğümlerinin tercih
değerlerini hesapla

Eğer ŞimdikiBölmeleme EnİyiBölmeleme'den iyi ise
EnİyiBölmeleme=ŞimdikiBölmeleme

ŞimdikiBölmeleme=EnİyiBölmeleme

Yordam Sonu

Algoritma 4.6 eş iki parçalı bölmeleme algoritmasının sonucu için kullanılan bir iyileştirme algoritmasıdır. Eğer bölmeleme ikiden fazla parça meydana getiriyorsa, bunun için genelleştirilmiş KL algoritmasının kullanılması gerekir. Bu algoritma Algoritma 4.7' de görülmektedir.

Bu işlem için geliştirilmiş olan Genel Kernighan-Lin algoritmasıdır (GKL). Bu algoritma 4.6' da ki algoritmadan birkaç yönden farklıdır. İlk olarak, burada tek değer yerine, parçalardan birine transfer olan düğüm için $q^k(i)$ değerleri hesaplanır. İkinci olarak, düğümler iki tane parçalarının düğümlerini karşılıklı değiştirme yerine, tekli olarak düğümler değiştirilir. Üçüncü olarak, KL algoritmasında bulunmayan yük dengeli olmayan parçalar oluşturulabilir, düğüm transferi dengeli kümelerin oluşturulmasını sağlar. Ortalama değerin altında olan parçalar kadar transfer düşünülmelidir. Beşinci olarak, ara bölmelemelerde dengesiz parçalar oluşabilir, sadece dengeli bölmelemeler göz önüne alınır.

Algoritma 4.7 : η tane parçaya ayrılmış olan bir bölmelemenin sonucunun için Genelleştirilmiş Kernighan-Lin iyileştirme algoritması.

Yordam GKL

Daha iyi bir bölmeleme bulunmayana kadar

İyiBölmeleme=ŞimdikiBölmeleme
Bütün k∈{1,...,η}, S_k=k parçasındaki düğümler
Bütün V_i
η-1 tercih değerlerini $q^k(i)$ hesapla
İzin verilen düğüm transferi var olduğu sürece
V_i=en büyük tercih değerli düğüm
l=V_i düğümünün gideceği küme
$S_{P(V_i)} = S_{P(V_i)} \setminus V_i$
$P(V_i)$=l
V_i düğümünün komşu düğümlerinin tercih değerlerini hesapla
eğer ŞimdikiBölmeleme EnİyiBölmeleme' den iyi ise
EnİyiBölmeleme=ŞimdikiBölmeleme
ŞimdikiBölmeleme=EnİyiBölmeleme
Yordam Sonu

4.2.3 Isı düşüşünün taklidi (Simulated Annealing - SA)

Bu algoritma tahmini soğutma işlemlerinde bir optimizasyon tekniğidir [LASZEWSKI, 1993]. Bu algoritmanın temeli statik mekaniğin çok sayıda elemanın etkileşimi üzerinde yapılan çalışmalara dayanır. Önemli olan, bu tip sistemlerin düşük sıcaklıkta çalışmasının incelenmesidir. Böyle bir sistem oluşturmak için, sistem yavaş-yavaş soğutulur. Bu algoritma isminden de anlaşılacağı gibi bu tip bir sistemin taklit edilmesidir. Gelişi güzel bir sistem durumundan başlanır. Amaç gelişi güzel karışıklıklarla daha az enerjili bir duruma ulaşmaktır. Algoritmanın anlaşılabilmesi için, olasılık matrisi A=[a_{st}]' nin a_{st} elemanı s durumundan t durumuna geçebilme olasılığını verdiğinin bilinmesi gerekir. Bu olasılık matrisi simetrik olarak seçilir.

$$E(t)-E(s)<0 \qquad (4.61)$$

(E(t) : t durumunun toplam enerjisi) (4.61) eşitsizliğinin sağlanıp sağlanmadığı kontrol edilir. Eğer bu şart sağlanırsa, bu yeni durum kabul edilir. Çünkü daha az enerjiye sahiptir. Eğer

$$E(t)-E(s)>0 \qquad (4.62)$$

ise, bu yeni durum (4.63) bağıntısı ile orantılı olan olasılıkla kabul edilebilir.

$$e^{\frac{E(t)-E(s)}{T}} \qquad (4.63)$$

Daha yüksek enerjili bir durumun kabul edilmesi, yerel çözümlerden kaçıp genel çözüm bulmak için yapılır. Bu işlem uygun gelişmeler olduğu sürece tekrarlanır. Sıcaklık parametreleri soğutma planına göre verilir, öyle ki

$$T_1 \geq T_2 \geq \cdots \geq T_k \wedge \lim_{k \to \infty} T_k \qquad (4.63)$$

olması gerekir. Bu yöntem ile yazılmış olan algoritma Algoritma 4.8' de verilmiştir.

Algoritma 4.8 : Isı düşüşünün taklidi algoritması.

yordam SA

1. Soğutma stratejisi seç, T_1, T_2, ...
2. Bir başlangıç çözümü (s) oluştur ve k←1
2. $T_k \geq T_{min}$ veya zaman limiti ulaşılınca kadar tekrarla
   s çözümüne komşu olan t çözümünü oluştur
   ΔE←E(t)-E(s)
   Eğer ΔE>0 ise
      s←t
   Eğer $e^{\frac{E(t)-E(s)}{T}}$ <RANDOM[0,1] ise
      s←t
   k←k+1

Yordam sonu

Çizge ile bu ısı düşüşünün taklidi yöntemi arasındaki ilişki Tablo 4.1' de görülmektedir. Olasılık matrisi, şu şekilde seçilebilir; Birinci olarak, t durumu s durumunun komşu durumudur ancak ve ancak bir düğümün bir alt kümeden diğer bir alt kümeye taşınması ile elde edilir. İkinci olarak, aynı parça içindeki komşu durumların olasılıkları aynıdır.

4.2.4. Enerji durum taklidi ve uyumlu ısı düşüşü taklidi

Bu algoritma, sıfır olmayan serbest enerjili sistemlerin global optimizasyon yöntemidir. SA algoritmasının tersine, bu yöntemde sistem genelde dengede tutulur. Sıcaklığı $T_{k+1}=rT_k$ şeklinde değiştirme yerine soğutma işlemine sıcaklık değişimi eklenir. Sıcaklık değeri ise önceden stokastik olarak tanımlanmış kümeden seçilir.

Algoritma 4.9 : Enerji durum taklidi ve uyumlu ısı düşüşü taklidi algoritması.

Yordam AST

1. $T \leftarrow T_0$
2. Başlangıç çözümünü s oluştur
3. Donma olmayana kadar devam et
   Sıcaklıkları oluştur
   s←Bulunan minimum çözüm
   k←1
   3.1. Sıcaklıkları güncelleştir
      i←1
      3.1.1.s komşuluğundan yeni çözümü t oluştur
         (bir tane düğümün diğer parçalardan
         birine taşınması)
         $\Delta E \leftarrow E(t)-E(s)$
         Eğer $\Delta E<0$ ise
            s←t
         Eğer $e^{\frac{E(t)-E(s)}{T}}$ < RANDOM[0,1] ise

Tablo 4.1 : Isı düşüşünün taklidi ile çizge bölmeleme problemi arasındaki benzerlik

Fiziksel sistem	SA-Çizge Bölmeleme Problemi
Durum	Mümkün olan bölmeleme
Enerji	Çözümün maliyeti
Toprak durum	Optimal çözüm

s←t

zaman←zaman+1

3.1.2. Eğer i<=L*Düğüm Sayısı, 3.1.1. adıma git

3.2. Eğer k<=K ise, 3.1. adıma git

Yordam sonu

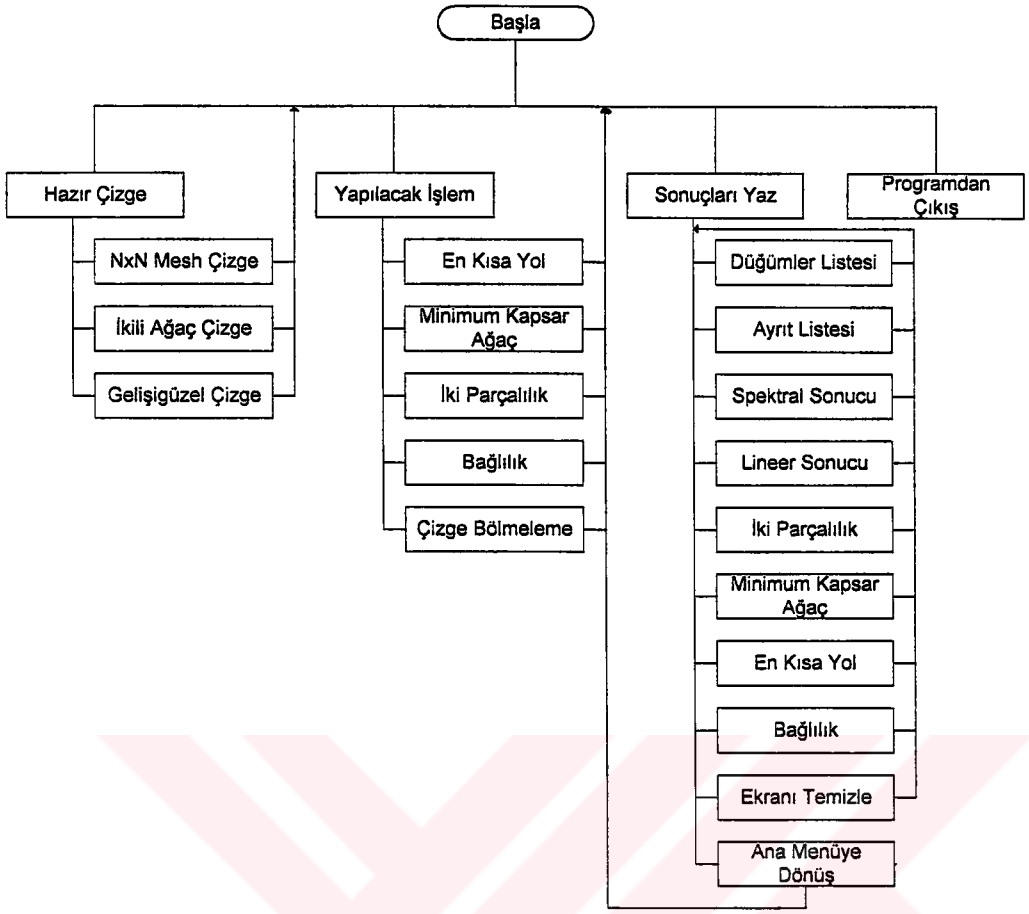
Sıcaklık değerlerinin medyanı başlangıç sıcaklığı olarak seçilir. Yüksek sıcaklıktan alçak sıcaklığa bir ölçeklendirmenin iyi bir performans sağlamadığı bilinmektedir. Çeşitli sıcaklık dağılımları seçilebilir. Dağılım şekli çözümü etkilemektedir. Bu yöntemde zor olanı ısının düşüşünün taklidinden çok sıcaklık dağılımının seçimidir. Bu metodun algoritması Algoritma 4.9' da görülmektedir.

5. YAPILAN UYGULAMA VE SONUÇLARI

Bu çalışmada, Delphi 2.0 derleyicisi kullanılarak, çizge bölmeleme uygulamasıyla bazı çizge algoritmalarının uygulamaları yapıldı. Çizge algoritmalarından En kısa yol, Minimum kapsar ağaç, İki parçalılık ve Çizge bağlılığı uygulamaları yapıldı. Bunun yanında çizge bölmelemede ise lineer ve spektral algoritmaları kullanılarak verilen çizgenin bölmelenmesi yapıldı. Yapılan bölmeleme sonuçlarının daha iyi olması için iyileştirme algoritmalarından Kernighan-Lin algoritmasının biraz değiştirilmiş şekli uygulandı.

Şekil 5.1' de ana programın akış diyagramı görülmektedir. Bu akış diyagramın sol sütununda görülen seçeneklerden biri seçilerek çizgenin oluşturulması sağlanır. İlk seçenek seçildiği zaman program iki boyutlu ağ (mesh) tipinde bir çizge oluşturmak için, ağın boyutunu kullanıcıdan ister. Bu bilgiyi aldıktan sonra düğüm ve ayrıt ağırlıkları gelişigüzel olan (negatif olmayan) bir kare ağ oluşturur. İkinci seçenek seçildiği zaman program çizgenin toplam ayrıt sayısı ister ve buna göre düğüm ve ayrıt ağırlıkları gelişigüzel olan (negatif olmayan) bir çizge oluşturur. Son seçenekte ise program toplam ayrıt sayısı ister ve kullanıcının düğüm numaralarını ve ağırlıklarını, ayrıt numaralarını ve ağırlıklarını girmesi için imkan tanınır. Bu şekilde çizge oluşturulur.

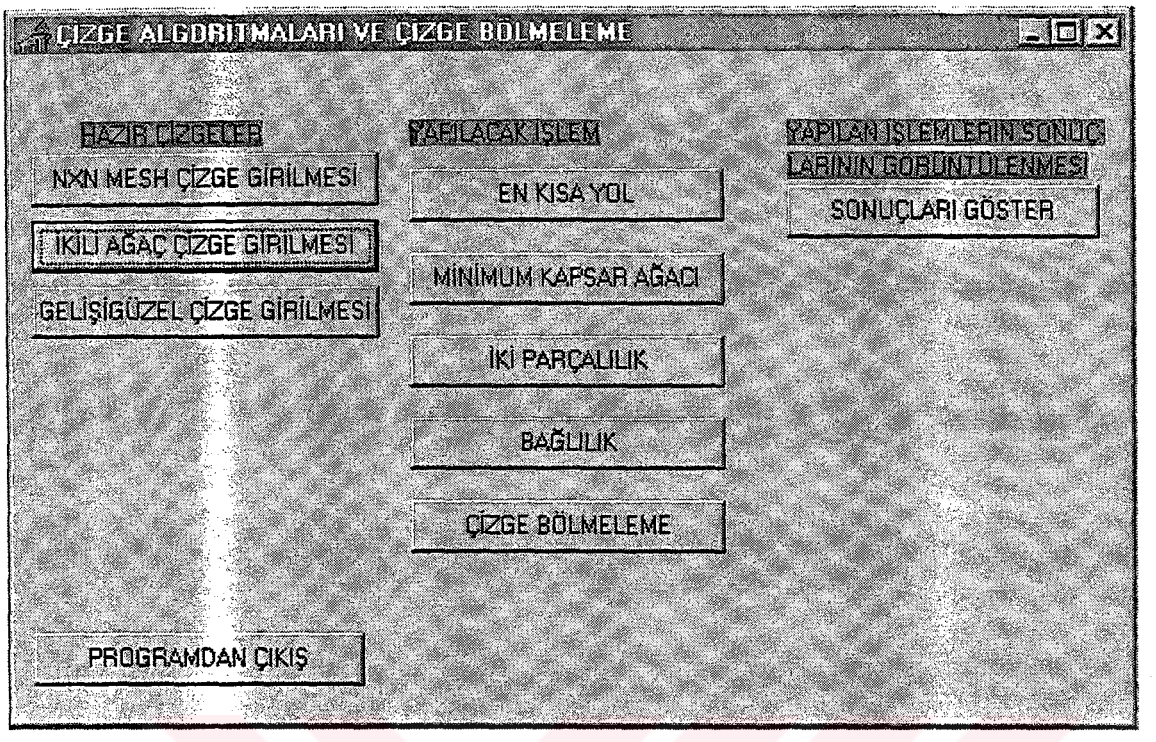
Ana mönünün ikinci sütununda ise girilen çizgenin üzerinde yapılacak işlemlerin seçenekleri sunulmuştur. İlk seçenek seçildiği zaman program en son girilen çizgenin bütün düğümleri arasındaki en kısa yolu ve bu yolun hangi düğümlerden geçtiğini bulur. İkinci seçenek seçildiği zaman çizgenin minimum ağırlıklı kapsar ağacı bulunur. Üçüncü seçenek seçildiği zaman çizgenin iki parçalı olup olmadığı test edilir. Dördüncü seçenek seçildiği zaman çizgenin bağlılığı test edilir. Son seçenekte ise, program girilen çizgeyi iki farklı yöntem (lineer ve spektral) ile bölmelenmesini yapar ve elde edilen bölmeleme sonuçlarına iyileştirme işlemini uygular. En sağdaki sütundaki tek seçenek seçildiği zaman, yeni bir mönü ekrana gelir.



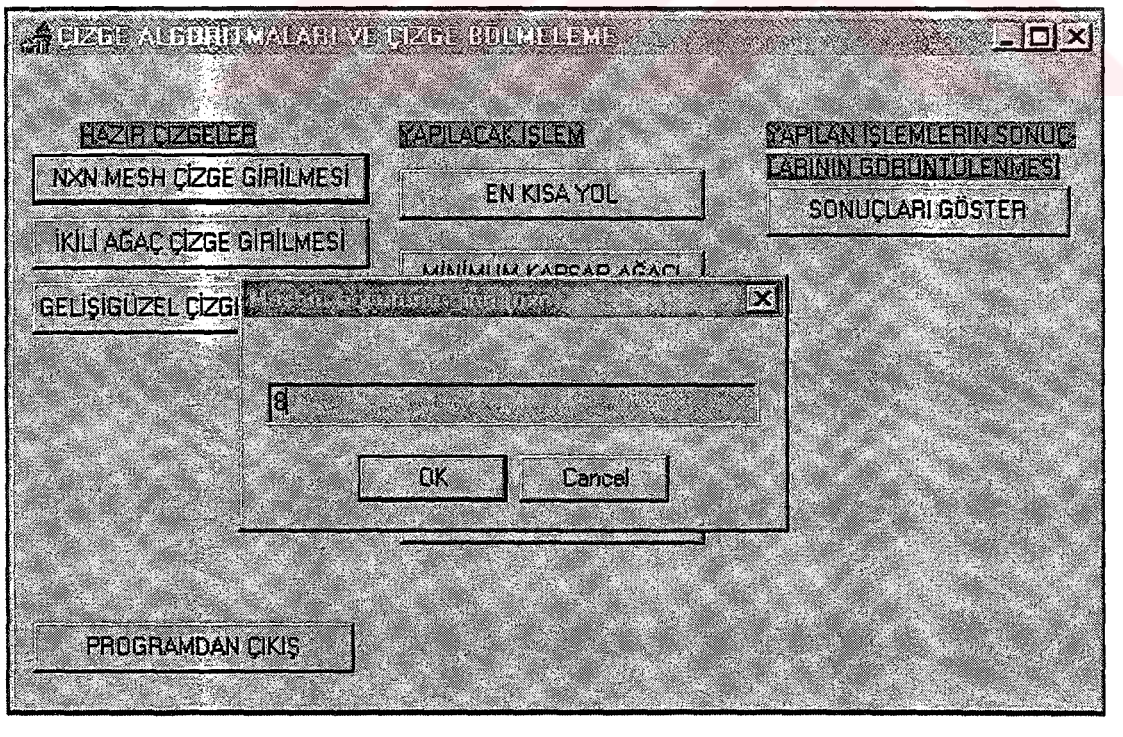
Şekil 5.1. Ana programın akış diyagramı.

Kullanıcı istediği işlemin sonucunun görüntülenmesini sağlayabilir.

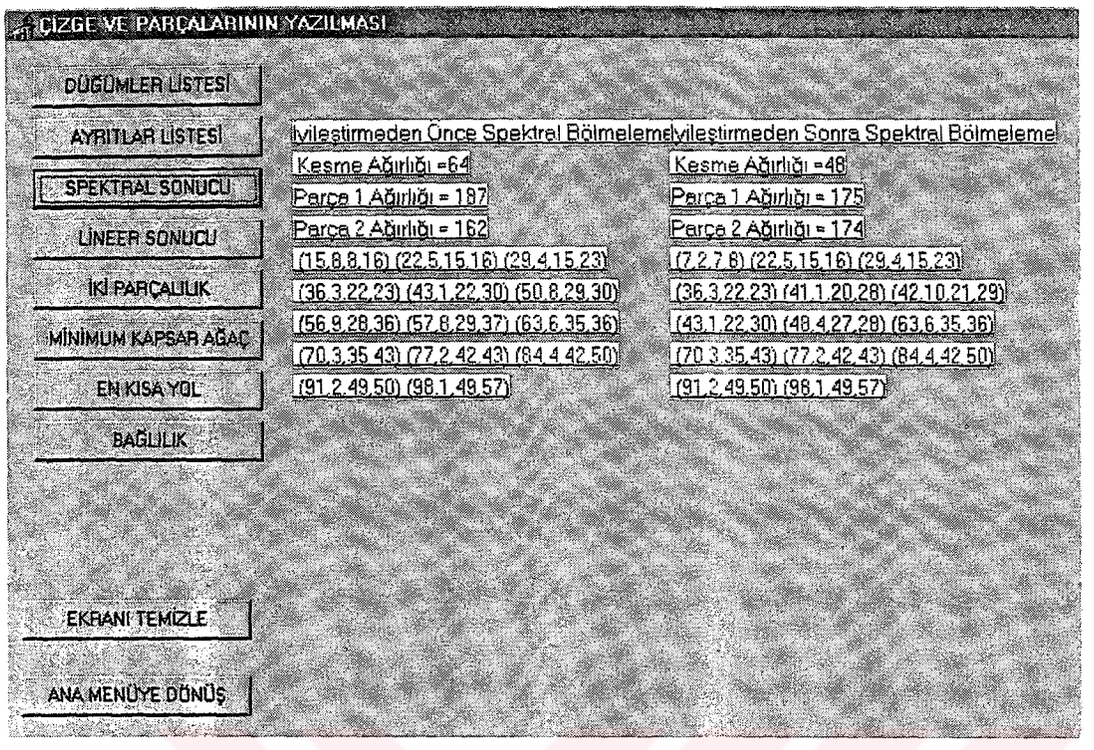
Program ilk çalıştırıldığı zaman Şekil 5.2' de görülen mönü kullanıcıya sunulur. Eğer yukarıda anlatıldığı gibi çizge girme işlemi yapılmadan önce işlemlerden biri seçilirse, kullanıcı uyarılır ve çizgeyi girdikten sonra işlem yapabileceği hatırlatılır. Şekil 5.3' te ağ tipinde bir çizgenin program tarafından hazırlanması için boyutu 8 olarak girilmektedir. Bu işlemin sonucunda kullanıcı istediği işlemin bu çizge üzerinde yapılmasını isteyebilir.



Şekil 5.2. Programın ana menüsü.



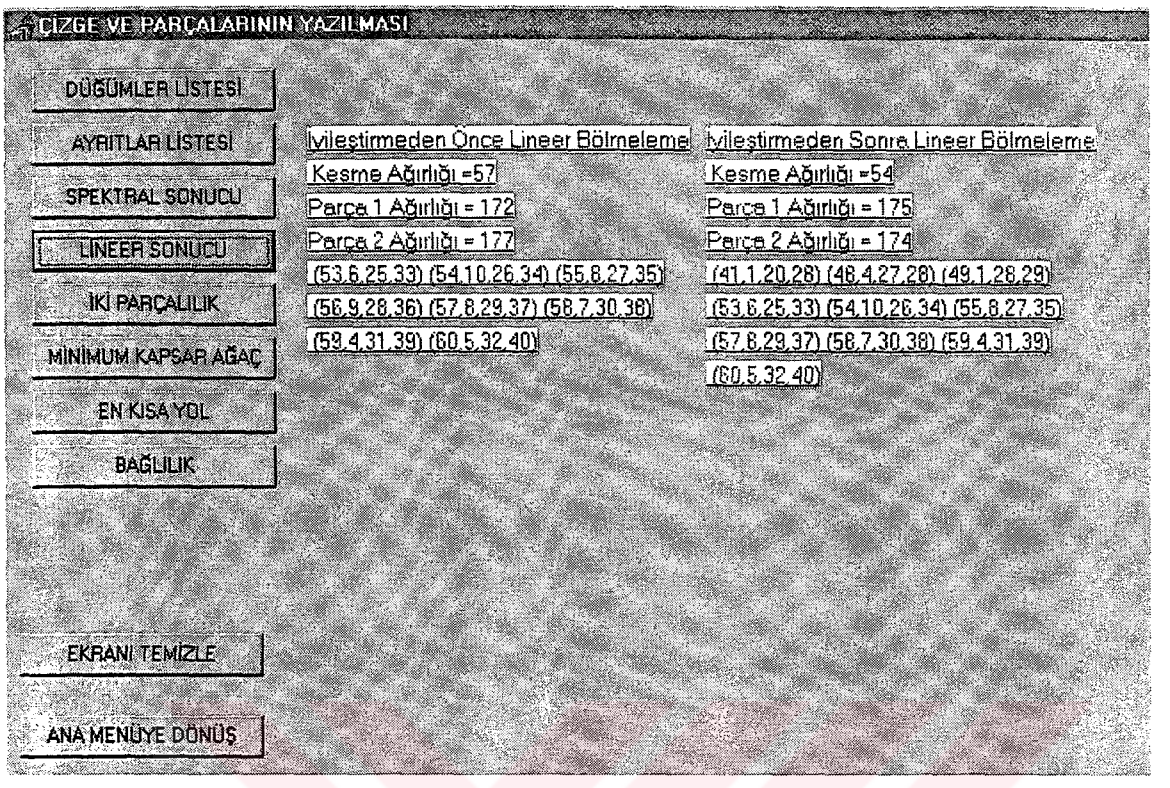
Şekil 5.3. Ağ türünden bir çizgenin oluşturulması.



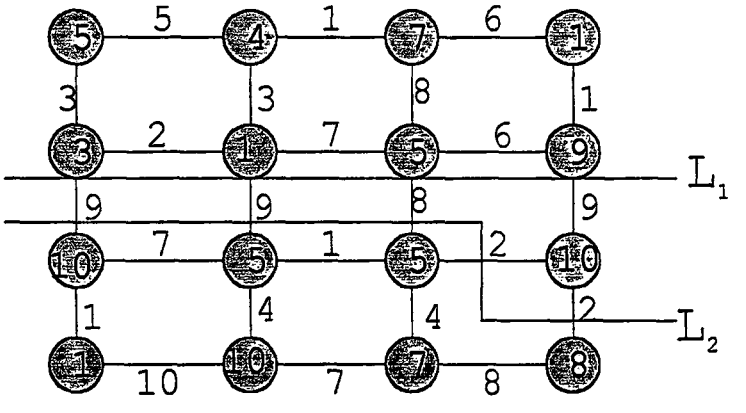
Şekil 5.4. Spektral yöntem ile 8x8' lik bir ağın bölmelenme sonucu.

Girilen 8x8 boyutlarında ağ üzerinde bölmeleme işlemi yapıldıktan sonra spektral bölmelemenin sonucu görüntülenmiş olarak Şekil 5.4' te görülmektedir. Şekil 5.5' te ise aynı çizgenin lineer algoritmaya göre bölmelenmesinin sonucu görülmektedir.

Tablo 5.1' de lineer algoritmanın çeşitli çizgelere göre ilk bölmeleme sonuçları ve iyileştirme algoritması uygulandıktan sonra elde edilen sonuçlar görülmektedir. İki boyutlu düzlemsel ağlarda (mesh) lineer algoritma, ikili ağaç çizgesindeki sonuçlardan daha iyi sonuçlar vermektedir. 4x4 boyutlarında bir iki boyutlu düzlemsel ağa uygulanan lineer algoritmasının sonucu Şekil 5.6' da görülmektedir. L_1 doğru parçası, çizgenin iyileştirme algoritmasının uygulanmadan önceki bölmelenmiş ayıracıdır ve L_2' de iyileştirme algoritmasının uygulandıktan sonra bölmelenmiş çizgenin ayıracıdır.



Şekil 5.5. Lineer çizge bölmeleme yöntemi ile 8x8' lik bir ağın bölmeleme sonucu.

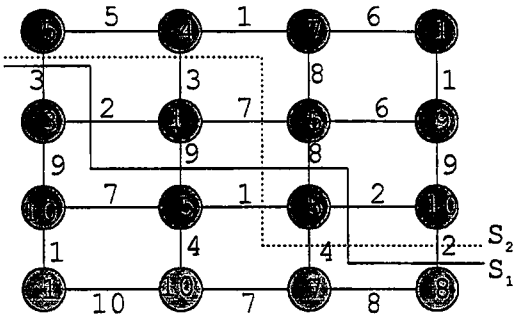


Şekil 5.6. Lineer algoritma kullanılarak iki boyutlu bir ağın bölmelenmesi.

Tablo 5.1. Lineer bölmeleme algoritmasının düzlemsel iki boyutlu ağ yapılı ve ikili ağaç yapıdaki çizgelere uygulanışının sonuçları

İki boyutlu ağ (mesh) tipindeki çizgelerde lineer algoritmanın sonuçları							
|V| ve |E|	İyileştirmeden önce			İyileştirmeden sonra			Gelişme yüzdesi
	W_{Ekesim}	W_{P1}	W_{P2}	W_{Ekesim}	W_{P1}	W_{P2}	
16 ve 24	35	56	35	30	46	45	%44,64
25 ve 40	32	71	72	22	76	67	%6,06
36 ve 60	37	112	94	36	110	96	%9,09
49 ve 84	55	110	162	31	134	138	%66
64 ve 112	57	172	177	54	175	174	%11,29
81 ve 144	37	215	223	32	222	216	%15,56
100 ve 180	64	235	262	51	251	246	%38,46
İkili ağaç yapısındaki çizgeler için lineer algoritmanın sonuçları							
|V| ve |E|	İyileştirmeden önce			İyileştirmeden sonra			Gelişme yüzdesi
	W_{Ekesim}	W_{P1}	W_{P2}	W_{Ekesim}	W_{P1}	W_{P2}	
7 ve 6	26	20	12	4	16	16	%88,24
15 ve 14	38	40	46	22	41	45	%40,91
31 ve 30	91	75	91	50	83	83	%53,27
63 ve 62	197	156	158	93	157	157	%53,27

İki boyutlu ağlar ve ikili ağaç yapıdaki çizgelerin bölmelenmesi spektral yöntemi kullanılarak yapıldı. Bu yapılan bölmelemelerin sonuçları Tablo 5.2' de görülmektedir. Bu çizgeler lineer bölmelemede kullanılan çizgelerin aynısıdır. Her iki tablonun incelenmesinde görüleceği gibi, spektral bölmelemenin sonucuna iyileştirme algoritmaları uygulanınca, daha büyük bir yüzde ile gelişme sağlanmakta ve spektral bölmeleme daha iyi sonuçlar vermektedir. Özellikle ikili ağaç yapıdaki çizgeler için spektral yöntemi çok iyi sonuç verdiği halde, lineer algoritma kötü sonuç vermektedir. Özellikle düğüm sayısı arttıkça lineer algoritma optimal çözümden çok



Şekil 5.7. Spektral yöntemi ile 4x4' lük bir ağın bölmelenmesi

uzaklaşmaktadır. İkili ağaç yapıda çizgeler için, spektral yönteminin gelişme yüzdeleri az görülebilir, fakat başlangıçta optimum çözüme yaklaştığından gelişme yüzdesi haliyle az olmaktadır.

Şekil 5.7' de spektral bölmeleme ile 4x4 boyutundaki bir ağın bölmelenmesi görülmektedir. Bu ağ spektral yöntem ile bölmelendikten sonra gelişme algoritması uygulanmıştır ve düğümlerin ve ayrıtların ağırlıklarından dolayı ilk çözüm en iyi çözüm olmadığından gelişme algoritması %23' lük bir iyileştirme sağlamıştır.

Bu tablolarda görülen değerler; ilk sütunda çizgenin düğüm sayısı ve ayrıt sayısı görülmektedir. Birinci üçlü sütundaki W_{Ekesim} değeri iki parça arasında kalan ayrıtların ağırlıkları toplamıdır. Aynı yerde W_{P1} birinci parçadaki düğümlerin ağırlıkları toplamıdır ve W_{P2}' de ikinci parçadaki düğümlerin ağırlıkları toplamıdır. Aynı değerler ikinci üçlü sütunda da vardır. Birinci üçlü sütundaki değerler iyileştirme işlemi uygulanmadan önceki değerlerdir ve ikinci üçlü sütundaki değerler de iyileştirme işlemi uygulandıktan sonraki değerlerdir.

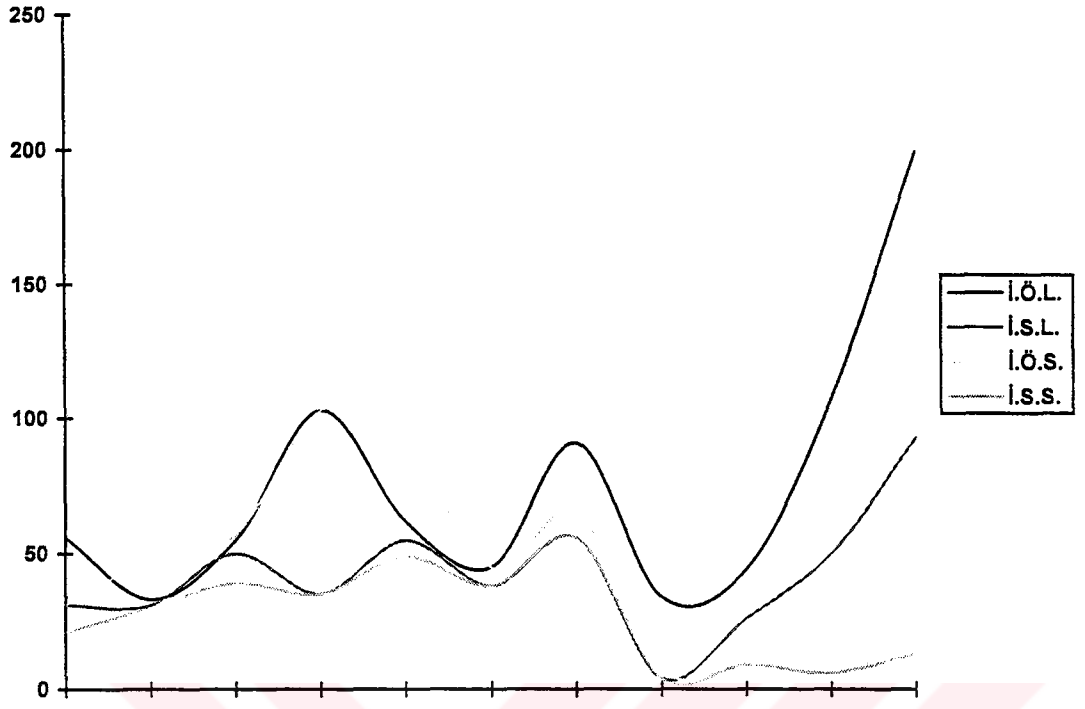
Her iki tablo dikkatlice incelendiği zaman spektral bölmelemenin lineer bölmelemeye göre çok daha iyi sonuçlar verdiği görülmektedir. Geliştirilen yazılımla bu iki algoritma aynı değerlere sahip 7, 15, 31, 63 düğümlü ikili ağaç şeklindeki çizgeler ve 16, 25, 36, 49, 64, 81, 100 düğümlü iki boyutlu

Tablo 5.2. Spektral bölmeleme algoritmasının düzlemsel iki boyutlu ağ yapılı ve ikili ağaç yapıdaki çizgelere uygulanışının sonuçları

İki boyutlu ağ (mesh) tipindeki çizgelerde spektral algoritmanın sonuçları							
|V| ve |E|	İyileştirmeden önce			İyileştirmeden sonra			Gelişme yüzdesi
	W_{ekesim}	W_{P1}	W_{P2}	W_{ekesim}	W_{P1}	W_{P2}	
16 ve 24	24	42	49	20	46	45	%36,36
25 ve 40	41	75	68	21	67	76	%37,5
36 ve 60	31	116	90	25	110	96	%31,58
49 ve 84	55	110	162	31	138	134	%66
64 ve 112	64	187	162	48	175	174	%44,94
81 ve 144	37	215	223	32	222	216	%15,56
100 ve 180	62	250	247	53	250	247	%13,85
İkili ağaç yapısındaki çizgeler için spektral algoritmanın sonuçları							
|V| ve |E|	İyileştirmeden önce			İyileştirmeden sonra			Gelişme yüzdesi
	W_{ekesim}	W_{P1}	W_{P2}	W_{ekesim}	W_{P1}	W_{P2}	
7 ve 6	3	18	14	4	16	16	%69,23
15 ve 14	1	39	47	1	39	47	%0
31 ve 30	6	83	83	6	83	83	%0
63 ve 62	6	152	162	3	162	152	%18,75

(mesh) ağ tipindeki çizgeler üzerinde denendi. İyileştirme işlemi uygulanmadan önce lineer algoritma optimum çözümden %14,8 saparken, spektral algoritma optimum çözümden %4,67 saptı. İyileştirme işleminden sonra lineer algoritma optimum çözümden %4,38 sapma gösterirken spektral algoritma optimum çözüme ulaştı.

Lineer çizge bölmeleme algoritmasının sonucu bazı durumlarda optimumdan çok uzak olabilir. Bu bölmelemede iyileştirme işlemi uygulanmadan önce bulunan sonuçtan daha uzakta optimum bir çözüm varsa ve bulunan çözüm ile bu optimum çözüm arasında kalan bölgelerde herhangi bir gelişme söz konusu değilse optimum çözüm bulunamaz.



Şekil 5.8. Lineer ve Spektral algoritmaların sonuçları (İ.Ö.L.:iyileştirmeden önce lineer, İ.S.L. : iyileştirmeden sonra lineer, İ.Ö.S. : iyileştirmeden önce spektral, İ.S.S. : iyileştirmeden sonra spektral).

Tablo 5.1 ve Tablo 5.2' verilen lineer ve spektral algoritmaların sonuçlarından, parçaların düğümlerinin ağırlıkları toplamının farkının ve kesim bölgesinde kalan ayrıtların ağırlıkları toplamı ile toplanması ile Şekil 5.8 elde edilmiştir. Şekil 5.8' de görüldüğü gibi, iyileştirme işlemi uygulandıktan sonra spektral algoritmanın sonucu her zaman için lineerden ya iyi veya eşit durumdadır.

Spektral çizge bölmeleme algoritması global olarak çok güçlü olduğundan, güçlü bir yerel iyileştirme algoritması uygulandığı zaman optimum veya optimuma çok yakın sonuçlar elde edilebilmektedir. Geliştirilen programın kaynak kodunun önemli kısmı EK-1' verilmiştir.

6. SONUÇ

Çizge algoritmaları, bir çok bilimsel problemin çözümünün kolaylaştırılması veya bu çözümün bulunması amacıyla yapılan çalışmalar sonucunda ortaya çıkmışlardır. Bundan dolayı, çizgelerin analizi ve testi için etkili algoritmaların geliştirilmesi çok önemlidir.

Günümüze kadar çizgeler üzerinde etkili çalışan çok sayıda algoritma geliştirilmiştir. Bunlar P sınıfı algoritmalardır. Bunun yanında hala yaklaşımla çözüm bulunan veya yaklaşık çözüm bulunan çizge problemleri de vardır. Bunlara örnek olarak verilen çizgenin bütün düğümlerini kapsayacak minimum ayrıt kümesinin bulunması, çizge bölmeleme v.b. problemlerdir.

Bu çalışmada bazı etkili çizge algoritmalarının uygulaması yapılmıştır. Fakat bu çalışmanın en önemli kısmı çizge bölmeleme algoritmalarının uygulanmasının yapılmış olmasıdır.

Çizge bölmeleme algoritmaları, temel olarak iki grupta toplanabilirler. Bir çizgenin düğümlerinin uzayda koordinatları belli ise, bu durumda çizge bölmeleme işlemi daha kolay olmaktadır. Ya sıralama algoritmaları kullanılarak bölmeleme yapılır ya da her düğüm bir kütle kabul edilerek bu kütlelerin ağırlık merkezinin bulunması yöntemi ile çizge bölmeleme işlemi yapılır. Eğer çizgenin düğümlerinin koordinatları yoksa, bu durumda işlemler biraz daha zorlaşmaktadır. Bu durumda, birden fazla algoritma kullanılabilir. Fakat bunların en iyi sonuç vereni spektral yöntemidir. Bu yöntemde, çizgenin Laplace matrisi oluşturulur ve numerik metotlar kullanılarak bu çizgenin öz değer vektörleri bulunur. Yapılacak olan bölmelemeye göre bu öz değer vektörlerinden biri kullanılır. Bu çalışmada ikinci en küçük öz değere karşılık gelen öz değer vektörü kullanılmıştır. Bu vektör ile çizge ikiye bölmelenmiştir. Bu çalışmada lineer yöntemi ile çizge bölmeleme uygulaması da yapılmıştır.

Çizge bölmeleme işlemi bir yaklaşım metodu olduğundan elde edilen sonuç optimum olmayabilir. Bundan dolayı elde edilen sonuçlara iyileştirme işlemleri uygulanır. Genel olarak iyileştirme işlemi uygulandıktan sonra spektral yöntemi ile

yapılan bölmeleme lineer bölmelemeye göre daha iyi sonuç vermektedir.

Düğüm koordinatları belli olmayan çizgelerin bölmelenmesi için kullanılan yöntemler içinde en iyisi spektral yöntemidir. Çünkü spektral yöntemin global olarak bir güçlülüğü vardır ve yerel olarak güçlü olan bir iyileştirme algoritması ile birleştirilince optimum ya da optimuma yakın sonuçlar elde edilebilmektedir. Bu yöntemde çizgenin şekli veya özelliği hiç önemli değildir. Çünkü çözüm cebirsel olarak yapılmaktadır.

Eğer çizgedeki düğüm sayısı çok fazla ise, bu durumda bilgisayarların hafıza sınırı olduğundan ve işlem zamanının önemli olmasından dolayı, bu çizgenin kabalaştırılmış hali elde edilir ve elde edilen kabalaştırılmış çizgenin bölmelenmesi yapılır. Kabalaştırma işlemi birden fazla seviyeden oluşabilir. Bundan dolayı bu yönteme çok seviyeli çizge bölmeleme yöntemi denir.

Yukarıda anlatılan yöntemlerin hepsinde işlem esnasında veya bölmeleme yapıldıktan uzun bir süre sonraya kadar çizgede herhangi bir değişikliğin olmayacağı varsayılmıştır. Eğer bu varsayım doğru değilse, bu durumda artımsal çizge bölmeleme yöntemi ile çizge bölmelenir. Bu yöntemin ana fikri, çizge anlatılan yöntemlerden biri ile bölmelenir ve çizgede bir değişiklik olursa, bu değişiklik için tekrar baştan bölmeleme yapılmaz, fakat bölmeleme sonucu üzerinde işlemler yapılarak yük dengeleme ve minimum kesme ayrıt sayısı şartları sağlanmaya çalışılır.

Bu çalışmada uygulamasını yaptığımız iki tip bölmeleme algoritması vardır. 5. bölümdeki değerlerin ışığı altında kıyas yapıldığı zaman spektral bölmelemenin lineere göre çok daha iyi olduğu görülmektedir. Bunun sebebi de spektral algoritmasının global bir güçlülüğü vardır ve bunun yanında yerel güçlülüğü iyi olan Kernighan-Lin algoritmasının uygulanması sonucu, bu çalışmada optimum çözümler elde edildi. Aynı iyileştirme algoritması lineer bölmelemenin de sonucuna uygulandı. Bazı durumlarda bu algoritmanın sonucu da optimum oldu fakat çoğunlukla optimumdan daha farklı sonuçlar elde edildi.

Uygulaması yapılan diğer algoritmalar P-tamam algoritmalar sınıfına girdiklerinden dolayı kesin sonucu bulacakları garanti demektir. Uygulamada da bunun böyle olduğunu gördük.

Çizge bölmeleme probleminin en büyük sıkıntısı düğümlerin ve ayrıtların ağırlıkları gelişigüzel olursa ve bir düğümün derecesi de gelişigüzel olursa, bu çizgelerin bölmelenmesi için iyi bir kriterin uygulanamamasından kaynaklanmaktadır. Kriter geliştirme yönünde Fiedler' in yaptığı bir çalışma var, fakat bu çalışmada çizgenin düğüm ve ayrıt ağırlıkları bir kabul edilerek yapılmıştır. Çizge ağırlıkları ile birlikte genel bir çizge değildir. Fiedler bu işi çizgelerin bir alt kümesi için yapmıştır. Şu ana kadar da bu bilim adamının bulduğu yöntemden daha iyi sonuç veren herhangi bir yöntem geliştirilmiş değil.

Fiedler, çizgeye sınırlama getirdiğinden dolayı bu yöntem ile elde edilen sonuçlara iyileştirme işlemi uygulanmaktadır.

Her türlü çizgeye uygulanabilecek ve optimum çözümü vereceğini garanti eden bir yöntem geliştirilirse, çizge bölmeleme problemi dışında bir çok problemin çözümünde kullanılacak ve optimum çözüm verecek olan algoritmalar geliştirilebilir. Bunun sebebi, çizge bölmeleme probleminin çözümü NP-zor olan bir problemdir, çözümü bu sınıfa giren bütün problemlere de çözüm bulunmuş olur.

KAYNAKLAR

AKL, S.G. (1989), The Design and Analysis of Parallel Algorithms, **Prentice-Hall, Englewood Cliffs**

BARNARD, S.T.; SIMON, H.D., (1993), A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems, **Proceedings of the 6th SIAM conference on Parallel Processing for Scientific Computing**

BONDY, J.A., MURTY, U.S.R.,(1976), Graph Theory with Applications, **MacMillan, London**

CORMEN, T.H., LEISERSON, C. E., RIVEST, R. L. ,(1991), Introduction to Algorithms, **The MIT Press**

DAHQUIST, G., BJÖRCK, A., ANDERSON, N. (1974), Numerical Methods, **Prentice-Hall, englewood Cliffs**

DEO, N. (1974), Graph Theory with Applications to Engineering and Computer Science, **Prentice-Hall, New Delhi**

DIMOV, I.; TONEV, O.(1994), Advances in Parallel Algorithms, **IOS Press, Amsterdam**

GIBBONS, A. (1985), Algorithmic Graph Theory, **Cambridge UniversityPress, Cambridge**

GUPTA, A.(1997), Fast and effective algorithms for graph partitioning and sparse-matrix ordering, **IBM Journal of Research and Development**, 41 (1/2)

HENDRICKSON, B. ve LELAND, R.(1995), The Chaco User's Guide

Version 2.0, **Technical report, Sandia National Laboratories**

HENDRICKSON, B. ve LELAND, R.(1993), An Improved spectral load balancing method, **6th SIAM Conf. Parallel Processing for Scientific Computing**, pp:953-961

HENDRICKSON, B. ve LELAND, R.(1992), An Improved spectral graph partitioning algorithm for mapping parallel computations, **Technical report, Sandia National Laboratories**

HENDRICKSON, B., LELAND, R. (1995), A Multilevel Algorithm for Partitioning Graph, **Proc. Supercomputing-95, ACM**

HOCKEY, R.W.; JESSHOPE, C.R., (1988), Parallel Computers 2, Architecture, Programming, and Algorithms, **IOP Publishing**

HULL, M.E.C ; CROOKES, D. ; SWEENEY, P.J, (1994), **Parallel Processing : The Transputer and its Applications**, Addison-Wesley

KARYPIS, G. ve KUMAR, V.(1997), Analysis of Multilevel Graph Partitioning, **Technical report, University of Minnesota**

KARYPIS, G. ve KUMAR, V. (1996), A Parallel Algorithm for Multilevel Graph Partitioning and Sparse-Matrix Ordering, **International Parallel Processing Symposium**

KARYPIS, G. ve KUMAR, V.(1996), Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs, **Technical report, University of Minnesota**

KARYPIS, G., KUMAR, V. (1996), Parallel Threshold-based ILU Factorization, **Technical report, University of Minnesota**

KARYPIS, G., KUMAR, V. (1996), Multilevel k-way Partitioning Scheme for Irregular Graphs, **Technical report, University of Minnesota**

KARYPIS, G. ; KUMAR, V., (1995), Analysis of Multilevel Graph Partitioning. **Technical Report, TR 95-037, University of Minnesota**

KARYPIS, G. ; KUMAR, V. (1995), A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. **Technical Report TR 95-035, University of Minnesota**

KIM, J-U, LEE, C-H ve KIM, M. (1993), Efficient multiple-way network-partitioning algorithm, **Computer-aided design**, 25 (5)

LASZEWSKI, G.V., (1993), A Collection of Graph Partitioning Algorithms, **Technical Report, Northeast Parallel Architectures Center at Syracuse University**

LASZEWSKI, G.V.(1993), An Object-Oriented Recursive Bisection Algorithm for CM5, **Technical Report, Northeast Parallel Architectures Center at Syracuse University**

LASZEWSKI, G.V.(1991), Intelligent Structural Operators for the k-way Graph Partitioning Problem, **Technical Report, Northeast Parallel Architectures Center at Syracuse University**

LELAND, R., HENDRICKSON, B. (1994), An Empirical Study of Static Load Balancing Algorithms, **Proc. Scalable High Perf. Comput. Conf., IEEE**, pp:682-685

MATHEWS, J.H. (1992), Numerical Methods for Mathematics, Science, and Engineering, **Prentice-Hall, Englewood Cliffs**

MAYEDA, W. (1972), Graph Theory, **Wiley-Interscience, New York**

MCHUGH, J. A. (1990),Algorithmic Graph Thoery, **Prentice-Hall, Englewood Cliffs**

OU, C.W. ve RANKA, S.(1997), Parallel Incremental Graph Partitioning, **IEEE Transactions on Parallel and Distributed Systems**, 8 (8)

SANCHIS, L.A. (1989), Multiple-way Network Partitioning, **IEEE Trans. on Computers**, 38 (1) pp:62-81

SCHLOEGEL, K., KARYPIS, G., KUMAR, V. (1997), Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes, **Technical report, University of Minnesota**

SCHLOEGEL, K., KARYPIS, G., KUMAR, V. (1997), Parallel Multilevel Diffusion Algorithms for Repartitioning of Adaptive Meshes, **Technical report, University of Minnesota**

SPIELMAN, D.A. ve TENG, S.H.(1996), Spectral Partitioning Work: Planar graphs and finite element meshes, **Technical Report, University of California, Berkeley**

SUARIS, P.R. ve KEDEM, G.(1988), An Algorithm for Quadrisection and Its Application to Standard Cell Placement, **IEEE Trans. on Circuits and Systems**, 35 (3)

THULASIRAMAN, K., SWAMY, M.N.S., (1992), Graphs : Theory and Algorithms, **A Wiley-Interscience Publication**

ZECEVIC, A.I., SILJAK, D.D.(1994), Balanced Decompositions of Sparse Systems for Multilevel Parallel Processing, **IEEE Trans. on Circuits and Systems**, 41 (3)

ZHOU, H.B., (1993) Two-stage m-way graph partitioning, **Parallel Computing**, 19, 1359-1373

ZOMAYA, A.Y., (1996), Parallel Computing : Paradigms and Applications, **International Thomson Computer Press**

EKLER

EK-1 : ÇİZGE ALGORİTMALARI VE ÇİZGE BÖLMELEME PROGRAMININ KAYNAK KODU

Burada verilen kodun çalışabilmesi için, burada verilen koda birde ana program kodunun eklenmesi gerekir. Bunun yanında sonuçları görmek için de ayrıca bir modülün yazılması gerekir. Veri yapıları ise koda bakıldığı zaman anlaşılmaktadır. Çizgeleri bilgisayar ortamında oluşturacak kodun da eklenmesi gerekir. Burada verilen kod sadece ana işlemleri yapan kısımlardır.

//Çizgenin iki parçalılığını test eden kaynak kod

```
unit bipartite;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun,graphun2;

  procedure GetInitialValue;
  procedure Bi_partite;
  function IsUnMarkedExist:boolean;

var
  Component : set of byte;
  Mark : array[1..MaxVertex] of
integer;

implementation

procedure GetInitialValue;
{ Çizgenin bitişiklik matrisinin bir
kopyasının çıkarılması}
var
  i, j : integer;
begin
  for i:=1 to MaxVertex do
    for j:=1 to MaxVertex do
      Matrix[i,j]:=AdMatrix[i,j];
  for i:=1 to MaxVertex do
    Mark[i]:=0;
  Component:=[];
end;

function IsUnMarkedExist:boolean;
{İşaretlenmemiş düğümün olup
olmadığını test eder}
var
  i : integer;
  flag2 : boolean;
begin
  flag2:=false;
  for i:=1 to Max do
    if Mark[i]=0 then
      flag2:=true;
  IsUnMarkedExist:=flag2;
end;

procedure Bi_partite;
{Verilen çizgenin iki parçalı olup
olmadığını test eder}
var
  flag : boolean;
  v,i : integer;
begin
  GetInitialValue;
  flag:=true;
  while ((flag) and (IsUnMarkedExist)) do
  begin
    i:=1;
    while i<=Max do
    begin
      if Mark[i]=0 then
      begin
        v:=i;
        i:=Max;
      end;
      i:=i+1;
    end;
    Component:=[v];
    Mark[v]:=1;
    PartSet[v]:=1;
    while ((flag) and (Component<>[])) do
    begin
      i:=1;
      while i<=Max do
      begin
        if i in Component then
        begin
          v:=i;
          i:=Max;
        end;
        i:=i+1;
      end;
      Component:=Component-[v];
      for i:=1 to Max do
        if ((v<>i) and (Matrix[v,i]=1))
then
        begin
          if PartSet[i]=0 then
          begin
            Component:=Component+[i];
            PartSet[i]:=3-PartSet[v];
            Mark[i]:=1;
          end
          else if PartSet[v]=PartSet[i]
then
            flag:=false;
        end;
    end;
  end;
  if flag then
    BiPart:=1
  else BiPart:=2;
end;
```

```
end.
```

//Çizgenin bağlılığını test eden kaynak kod

```
unit Connected;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun2;

  procedure Connectedness;
  procedure GetVertexSet;
  procedure GetComponentOfGraph(i2,k2
: integer);
  procedure GetVertexNeighbor(i2 :
integer);
  procedure GetAdjacencyMatrix;

var
  C_Set2,C_Set3 : set of byte;
  AdMatrix2 :
array[1..MaxVertex,1..MaxVertex] of
integer;

implementation


procedure GetVertexSet;
var
  i : integer;
begin
  C_Set2:=[];
  for i:=1 to Max do
    C_Set2:=C_Set2+[i];
end;

procedure GetAdjacencyMatrix;
var
  i,j : integer;
begin
  for i:=1 to Max do
    for j:=1 to Max do
      AdMatrix2[i,j]:=AdMatrix[i,j];
end;

procedure GetComponentOfGraph(i2,k2 :
integer);
var
  j,k : integer;
begin
  while C_Set3<>[] do
  begin
    for j:=1 to Max do
    begin
      if j in C_Set3 then
      begin
        for k:=1 to Max do
          if ((AdMatrix2[j,k]=1) and not(k
in C_Set3) and
              not(k in ConnectSet[k2])) then
            begin
              C_Set3:=C_Set3+[k];
              C_Set2:=C_Set2-[k];
            end;
          ConnectSet[k2]:=ConnectSet[k2]+[j];
          C_Set3:=C_Set3-[j];
        end;
      end;
    end;
  end;

procedure GetVertexNeighbor(i2 : integer);
var
  j : integer;
begin
  for j:=1 to Max do
    if ((AdMatrix2[i2,j]=1) and not(j in
C_Set3)) then
      begin
        C_Set3:=C_Set3+[j];
        C_Set2:=C_Set2-[j];
      end;
  C_Set3:=C_Set3+[i2];
  C_Set2:=C_Set2-[i2];
end;

procedure Connectedness;
var
  i,j,k : integer;
begin
  i:=1;
  k:=1;
  C_Set3:=[];
  GetVertexSet;
  GetAdjacencyMatrix;
  while ((C_Set2<>[]) and (i<=Max)) do
  begin
    if i in C_Set2 then
    begin
      GetVertexNeighbor(i);
      GetComponentOfGraph(i,k);
      k:=k+1;
    end;
    i:=i+1;
  end;
  if k=2 then
    Connect:=1
  else Connect:=2;
end;

end.
```

//Lineer çizge bölmeleme yönteminin kaynak kodu

```
unit LPartition;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun2;

  procedure LinearPartition;
  procedure GetLinearCutSet;
  procedure GetLinearPartitionSet;

var
  N:array[1..MaxVertex] of integer;

implementation

procedure GetLinearCutSet;
var
  i : integer;
begin
  i:=1;
  while AyritSet[i].Ayrit_No>0 do
  begin
```

```
Y_L_PartitionCutSet[i].Ayrit_No:=Ayri
tSet[i].Ayrit_No;

Y_L_PartitionCutSet[i].Ayrit_Ag:=Ayri
tSet[i].Ayrit_Ag;

Y_L_PartitionCutSet[i].Ayrit_Uc1:=Ayr
itSet[i].Ayrit_Uc1;

Y_L_PartitionCutSet[i].Ayrit_Uc2:=Ayr
itSet[i].Ayrit_Uc2;
    if (((AyritSet[i].Ayrit_Uc1 in
L_Set1) and (AyritSet[i].Ayrit_Uc2 in
L_Set2))
       xor ((AyritSet[i].Ayrit_Uc2 in
L_Set1) and (AyritSet[i].Ayrit_Uc1 in
L_Set2))) then

Y_L_PartitionCutSet[i].Kesme:='E';
    i:=i+1;
  end;
  i:=1;
  while
Y_L_PartitionCutSet[i].Ayrit_No>0 do
  begin

L_PartitionCutSet[i].Ayrit_No:=Y_L_Pa
rtitionCutSet[i].Ayrit_No;

L_PartitionCutSet[i].Ayrit_Ag:=Y_L_Pa
rtitionCutSet[i].Ayrit_Ag;

L_PartitionCutSet[i].Ayrit_Uc1:=Y_L_P
artitionCutSet[i].Ayrit_Uc1;

L_PartitionCutSet[i].Ayrit_Uc2:=Y_L_P
artitionCutSet[i].Ayrit_Uc2;

L_PartitionCutSet[i].Kesme:=Y_L_Parti
tionCutSet[i].Kesme;
    i:=i+1;
  end;
end;

procedure GetLinearPartitionSet;
var
  i : integer;
begin
  for i:=1 to Max do
  begin

Y_L_PartitionSet[i].Dugum_No:=DugumSe
t[i].Dugum_No;

Y_L_PartitionSet[i].Dugum_Ag:=DugumSe
t[i].Dugum_Ag;
    if N[i]<0 then

Y_L_PartitionSet[i].Partition_No:=1
    else
Y_L_PartitionSet[i].Partition_No:=2;
  end;

  for i:=1 to Max do
  begin

L_PartitionSet[i].Dugum_No:=Y_L_PartitionSe
t[i].Dugum_No;

L_PartitionSet[i].Dugum_Ag:=Y_L_PartitionSe
t[i].Dugum_Ag;

L_PartitionSet[i].Partition_No:=Y_L_Partiti
onSet[i].Partition_No;
  end;
end;

procedure LinearPartition;
var
  i,j,k : integer;
begin
  GetLaplaceMatrix;
  L_Set1:=[];
  L_Set2:=[];
  for i:=1 to Max do
  begin
    L_Set1:=L_Set1+[DugumSet[i].Dugum_No];
    N[i]:=-1;
  end;

  N[DugumSet[1].Dugum_No]:=1;
  L_Set2:=L_Set2+[DugumSet[1].Dugum_No];
  L_Set1:=L_Set1-[DugumSet[1].Dugum_No];
  k:=1;

  i:=1;
  while ((i<=Max) and (k<(Max div 2))) do
  begin
    j:=i+1;
    while ((j<=Max) and (k<(Max div 2))) do
    begin
      if Matrix[i,j]=-1 then
        if ((i in L_Set2) and not(j in
L_Set2)) then
        begin
          L_Set2:=L_Set2+[j];
          L_Set1:=L_Set1-[j];
          N[j]:=1;
          k:=k+1;
        end;
      j:=j+1;
    end;
    i:=i+1;
  end;
  GetLinearPartitionSet;
  GetLinearCutSet;
end;

end.
```

//Spektral yöntemi ile çizge bölmelemenin kaynak kodu

```
unit spartition;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun2;

  procedure
FindSecondSmallestEigenVector;
  procedure GetCutSet;
  procedure IdentityMatrix;
  procedure SpectralPartition;
  procedure FindEigenVectors;
  procedure ZeroOut;
  procedure GetPartitionSet;
  function RMS:real;

var
  VectorXP,VectorXQ,VectorYP,VectorYQ :
array[1..MaxVertex] of real;
  V : array[1..MaxVertex,1..MaxVertex] of
real;
  p,q,index : integer;
  Epsilon : real;
```

```
implementation

procedure IdentityMatrix;
var
  i,j : integer;
begin
  for i:=1 to Max do
    for j:=1 to Max do
      V[i,j]:=0.0;
  for i:=1 to Max do
    V[i,i]:=1.0;
end;

function RMS:real;
var
  i : integer;
  sum : real;
begin
  sum:=0.0;
  for i:=1 to Max do
    sum:=sum+Matrix[i,i]*Matrix[i,i];
  RMS:=sqrt(sum/Max);
end;

procedure ZeroOut;
var
  j : integer;
  Theta,t,c,s,r : real;
begin
  Theta:=(Matrix[q,q]-
Matrix[p,p])/(2*Matrix[p,q]);
  if Theta<0.0 then
    t:=-1.0
  else t:=1.0;

t:=t/(abs(Theta)+sqrt(Theta*Theta+1))
;
  c:=1.0/sqrt(t*t+1);
  s:=c*t;

  for j:=1 to Max do
  begin
    VectorXP[j]:=Matrix[j,p];
    VectorXQ[j]:=Matrix[j,q];
  end;

  Matrix[p,q]:=0.0;
  Matrix[q,p]:=0.0;

Matrix[p,p]:=c*c*VectorXP[p]+s*s*Vect
orXQ[q]-2*s*c*VectorXQ[p];

Matrix[q,q]:=s*s*VectorXP[p]+c*c*Vect
orXQ[q]+2*c*s*VectorXQ[p];

  for j:=1 to Max do
    if ((j<>p) and (j<>q)) then
    begin
      Matrix[j,p]:=c*VectorXP[j]-
s*VectorXQ[j];
      Matrix[p,j]:=Matrix[j,p];

Matrix[j,q]:=c*VectorXQ[j]+s*VectorXP
[j];
      Matrix[q,j]:=Matrix[j,q];
    end;

  for j:=1 to Max do
  begin
    VectorYP[j]:=V[j,p];
    VectorYQ[j]:=V[j,q];
  end;

  for j:=1 to Max do
  begin
    V[j,p]:=c*VectorYP[j]-
s*VectorYQ[j];
    V[j,q]:=s*VectorYP[j]+c*VectorYQ[j];
  end;
end;


procedure FindEigenVectors;
var
  Flag : boolean;
  CountS : integer;
  T2 : real;
begin
  GetLaplaceMatrix;
  IdentityMatrix;
  CountS:=0;
  repeat
    T2:=RMS;
    CountS:=CountS+1;
    Flag:=false;
    for p:=1 to Max-1 do
      for q:=p+1 to Max do
        if
(abs(Matrix[p,q])/abs(T2))>Epsilon then
        begin
          ZeroOut;
          flag:=true;
        end;
  until ((not(Flag)) or (CountS>50));
end;

procedure  FindSecondSmallestEigenVector;
var
  i : integer;
  m1,m2 : real;
begin
   m1:=100.0;
   m2:=100.0;
   index:=2;
   for i:=1 to Max do
     if Matrix[i,i]<m1 then
       m1:=Matrix[i,i];

   for i:=1 to Max do
     if ((m2>Matrix[i,i]) and
(m1<Matrix[i,i])) then
     begin
       m2:=Matrix[i,i];
       index:=i;
     end;

end;

procedure GetCutSet;
var
  i : integer;
begin
  i:=1;
  while AyritSet[i].Ayrit_No>0 do
  begin

Sp_PartitionCutSet[i].Ayrit_No:=AyritSet[i]
.Ayrit_No;

Sp_PartitionCutSet[i].Ayrit_Ag:=AyritSet[i]
.Ayrit_Ag;

Sp_PartitionCutSet[i].Ayrit_Uc1:=AyritSet[i
].Ayrit_Uc1;

Sp_PartitionCutSet[i].Ayrit_Uc2:=AyritSet[i
].Ayrit_Uc2;
    if (((AyritSet[i].Ayrit_Uc1 in Sp_Set1)
and (AyritSet[i].Ayrit_Uc2 in Sp_Set2))
       xor ((AyritSet[i].Ayrit_Uc2 in
Sp_Set1) and (AyritSet[i].Ayrit_Uc1 in
Sp_Set2))) then
      Sp_PartitionCutSet[i].Kesme:='E';
    i:=i+1;
  end;
```

```
  i:=1;
  while
Sp_PartitionCutSet[i].Ayrit_No>0 do
  begin

Y_Sp_PartitionCutSet[i].Ayrit_No:=Sp_
PartitionCutSet[i].Ayrit_No;

Y_Sp_PartitionCutSet[i].Ayrit_Ag:=Sp_
PartitionCutSet[i].Ayrit_Ag;

Y_Sp_PartitionCutSet[i].Ayrit_Uc1:=Sp
_PartitionCutSet[i].Ayrit_Uc1;

Y_Sp_PartitionCutSet[i].Ayrit_Uc2:=Sp
_PartitionCutSet[i].Ayrit_Uc2;

Y_Sp_PartitionCutSet[i].Kesme:=Sp_Par
titionCutSet[i].Kesme;
    i:=i+1;
  end;
end;

procedure GetPartitionSet;
var
  i : integer;
begin
  for i:=1 to Max do
  begin

Sp_PartitionSet[i].Dugum_No:=DugumSet
[i].Dugum_No;

Sp_PartitionSet[i].Dugum_Ag:=DugumSet
[i].Dugum_Ag;
    if V[i,index]<0.0 then
    begin

Sp_Set1:=Sp_Set1+[DugumSet[i].Dugum_No];
      Sp_PartitionSet[i].Partition_No:=1;
    end
    else begin
      Sp_PartitionSet[i].Partition_No:=2;

Sp_Set2:=Sp_Set2+[DugumSet[i].Dugum_No];
    end;
  end;
  for i:=1 to Max do
  begin

Y_Sp_PartitionSet[i].Dugum_No:=Sp_Partition
Set[i].Dugum_No;

Y_Sp_PartitionSet[i].Dugum_Ag:=Sp_Partition
Set[i].Dugum_Ag;

Y_Sp_PartitionSet[i].Partition_No:=Sp_Parti
tionSet[i].Partition_No;
  end;
end;

procedure SpectralPartition;
begin
  Epsilon:=0.00000007;
  GetLaplaceMatrix;
  IdentityMatrix;
  FindEigenVectors;
  FindSecondSmallestEigenVector;
  Sp_Set1:=[];
  Sp_Set2:=[];
  GetPartitionSet;
  GetCutSet;
end;

end.
```

//İyileştirme işlemini yapan kaynak kod

```
unit refine;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun2;

type

  VertexGain=record
    EdgeInside : integer;
    EdgeOutside : integer;
  end;

  procedure LinearRefinement;
  procedure
CopyPartitionCutSet(PartitionCutSet :
array of Ayrit;
          var P_CutSet3 : array of
Ayrit);
  procedure
CopyPartitionSet(PartitionSet2 :
array of PartitionSet;
          var P_Set3 : array of
PartitionSet);
  procedure SpectralRefinement;
  procedure
FindMaxDifference(NPartitionSet :
array of PartitionSet;
          PartitionCutSet : array of
Ayrit;
          PartW : PartWeights;
          Set1,Set2 : SetOfByte);
  procedure VertexGainEvaluation(Set1,Set2
: SetOfByte );
  procedure Weights(var PartW :
PartWeights; NPartitionSet : array of
PartitionSet);
  procedure InitializeVGain;
  function
EvaluateRefinement(PartitionCutSet : array
of Ayrit;
          PartW : PartWeights):integer;
  procedure ChangeNodes(var NPartitionSet :
array of PartitionSet);
  procedure ChangeCutSet(var Set1,Set2 :
SetOfByte;
                    var PartitionCutSet
:array of Ayrit);

var
  VGain : array[1..MaxVertex] of
VertexGain;
  MaxDifIndex : integer;
  MaxDifFlag : boolean;

implementation

procedure InitializeVGain;
var
  i : integer;
begin
  for i:=1 to MaxVertex do
  begin
    VGain[i].EdgeInside:=0;
    VGain[i].EdgeOutside:=0;
  end;
end;
```

```
procedure Weights(var PartW :
PartWeights; NPartitionSet : array of
PartitionSet);
var
  i : integer;
begin
  PartW.Part_One:=0;
  PartW.Part_Two:=0;
  for i:=0 to Max-1 do
    if
NPartitionSet[i].Partition_No=1 then

PartW.Part_One:=PartW.Part_One+NParti
tionSet[i].Dugum_Ag
    else

PartW.Part_Two:=PartW.Part_Two+NParti
tionSet[i].Dugum_Ag;
end;

procedure
VertexGainEvaluation(Set1,Set2 :
SetOfByte );
var
  i : integer;
begin
  for i:=1 to Max do
  begin
    VGain[i].EdgeInside:=0;
    VGain[i].EdgeOutside:=0;
  end;
  i:=1;
  while AyritSet[i].Ayrit_No<>0 do
  begin
    if (((AyritSet[i].Ayrit_Uc1 in
Set1) and (AyritSet[i].Ayrit_Uc2 in
Set1)) or
    ((AyritSet[i].Ayrit_Uc1 in Set2)
and (AyritSet[i].Ayrit_Uc2 in Set2)))
then
    begin

VGain[AyritSet[i].Ayrit_Uc1].EdgeInsi
de:=VGain[AyritSet[i].Ayrit_Uc1].Edge
Inside+AyritSet[i].Ayrit_Ag;

VGain[AyritSet[i].Ayrit_Uc2].EdgeInsi
de:=VGain[AyritSet[i].Ayrit_Uc2].Edge
Inside+AyritSet[i].Ayrit_Ag;
    end
    else if (((AyritSet[i].Ayrit_Uc1
in Set1) and (AyritSet[i].Ayrit_Uc2
in Set2)) or
    ((AyritSet[i].Ayrit_Uc1 in Set2)
and (AyritSet[i].Ayrit_Uc2 in Set1)))
then
    begin

VGain[AyritSet[i].Ayrit_Uc1].EdgeOuts
ide:=VGain[AyritSet[i].Ayrit_Uc1].Edg
eOutside+AyritSet[i].Ayrit_Ag;

VGain[AyritSet[i].Ayrit_Uc2].EdgeOuts
ide:=VGain[AyritSet[i].Ayrit_Uc2].Edg
eOutside+AyritSet[i].Ayrit_Ag;
    end;
    i:=i+1;
  end;
end;

procedure ChangeNodes(var
NPartitionSet : array of
PartitionSet);
var
  i : integer;
begin
  for i:=0 to Max-1 do
    if
NPartitionSet[i].Dugum_No=MaxDifIndex then
      NPartitionSet[i].Partition_No:=3-
NPartitionSet[i].Partition_No;
end;

procedure ChangeCutSet(var Set1,Set2 :
SetOfByte;
                    var PartitionCutSet
:array of Ayrit);
var
  i : integer;
begin
  if MaxDifIndex in Set1 then
  begin
    Set1:=Set1-[MaxDifIndex];
    Set2:=Set2+[MaxDifIndex];
  end
  else begin
    Set2:=Set2-[MaxDifIndex];
    Set1:=Set1+[MaxDifIndex];
  end;
  i:=0;
  while PartitionCutSet[i].Ayrit_No>0 do
  begin
    PartitionCutSet[i].Kesme:='H';
    i:=i+1;
  end;
  i:=0;
  while PartitionCutSet[i].Ayrit_No>0 do
  begin
    if (((PartitionCutSet[i].Ayrit_Uc1 in
Set1) and (PartitionCutSet[i].Ayrit_Uc2 in
Set2))
      or ((PartitionCutSet[i].Ayrit_Uc2 in
Set1) and (PartitionCutSet[i].Ayrit_Uc1 in
Set2))) then
      PartitionCutSet[i].Kesme:='E';
    i:=i+1;
  end;
end;

procedure
CopyPartitionCutSet(PartitionCutSet : array
of Ayrit;
          var P_CutSet3 : array of Ayrit);
var
  i : integer;
begin
  i:=0;
  while PartitionCutSet[i].Ayrit_No>0 do
  begin

P_CutSet3[i].Ayrit_No:=PartitionCutSet[i].A
yrit_No;

P_CutSet3[i].Ayrit_Ag:=PartitionCutSet[i].A
yrit_Ag;

P_CutSet3[i].Ayrit_Uc1:=PartitionCutSet[i].
Ayrit_Uc1;

P_CutSet3[i].Ayrit_Uc2:=PartitionCutSet[i].
Ayrit_Uc2;

P_CutSet3[i].Kesme:=PartitionCutSet[i].Kesm
e;
    i:=i+1
  end;
end;

procedure CopyPartitionSet(PartitionSet2 :
array of PartitionSet;
         var P_Set3 : array of
PartitionSet);
var
  i : integer;
begin
```

```
  for i:=0 to Max-1 do
  begin

P_Set3[i].Dugum_No:=PartitionSet2[i].
Dugum_No;

P_Set3[i].Dugum_Ag:=PartitionSet2[i].
Dugum_Ag;

P_Set3[i].Partition_No:=PartitionSet2
[i].Partition_No;
  end;
end;

procedure
FindMaxDifference(NPartitionSet :
array of PartitionSet;
          PartitionCutSet : array of
Ayrit;
          PartW : PartWeights;
          Set1,Set2 : SetOfByte);
var
  P_Set2 : array[1..MaxVertex] of
PartitionSet;
  P_CutSet2 : array[1..MaxEdge] of
Ayrit;
  PartW2 : PartWeights;
  Set3,Set4 : SetOfByte;
  Temp1,Temp2,i,j,k : integer;
  Set5 : set of byte;
begin
  MaxDifIndex:=1;
  j:=-1;
  i:=0;
  while PartitionCutSet[i].Ayrit_No>0
do
  begin
    if PartitionCutSet[i].Kesme='E'
then

Temp2:=Temp2+PartitionCutSet[i].Ayrit
_Ag;
    i:=i+1;
  end;
  Set5:=[];
  i:=0;
  while PartitionCutSet[i].Ayrit_No>0
do
  begin
    if PartitionCutSet[i].Kesme='E'
then
    begin

Set5:=Set5+[PartitionCutSet[i].Ayrit_
Uc1];

Set5:=Set5+[PartitionCutSet[i].Ayrit_
Uc2];
    end;
    i:=i+1;
  end;
  repeat

k:=VGain[MaxDifIndex].EdgeOutside-
VGain[MaxDifIndex].EdgeInside;
    if ((MaxDifIndex in set5) and
(k>0)) then
    begin
      Set3:=Set1;
      Set4:=Set2;

PartW2.Part_One:=PartW.Part_One;

PartW2.Part_Two:=PartW.Part_Two;

CopyPartitionSet(NPartitionSet,P_Set2
);

CopyPartitionCutSet(PartitionCutSet,P_CutSe
t2);
      ChangeNodes(P_Set2);
      ChangeCutSet(Set3,Set4,P_CutSet2);
      Weights(PartW2,P_Set2);
      Temp1:=0;
      i:=1;
      while P_CutSet2[i].Ayrit_No>0 do
      begin
        if P_CutSet2[i].Kesme='E' then

Temp1:=Temp1+P_CutSet2[i].Ayrit_Ag;
        i:=i+1;
      end;
      if (Temp2+abs(PartW.Part_One-
PartW.Part_Two))>(Temp1+abs(PartW2.Part_One
-PartW2.Part_Two)) then
      begin
        Temp2:=Temp1;
        j:=MaxDifIndex;
      end;
    end;
    MaxDifIndex:=MaxDifIndex+1;
  until ((Set5=[]) or (MaxDifIndex>Max));
  if j>0 then
  begin
    MaxDifIndex:=j;
    MaxDifFlag:=true;
  end
  else MaxDifFlag:=false;
end;

function EvaluateRefinement(PartitionCutSet
: array of Ayrit;
          PartW : PartWeights):integer;
var
  Refinement,i : integer;
begin
  Refinement:=abs(PartW.Part_One-
PartW.Part_Two);
  i:=0;
  while PartitionCutSet[i].Ayrit_No>0 do
  begin
    if PartitionCutSet[i].Kesme='E' then

Refinement:=Refinement+PartitionCutSet[i].A
yrit_Ag;
    i:=i+1;
  end;
end;

procedure SpectralRefinement;
var
  refine1,refine2 : integer;
  flag : boolean;
begin
  MaxDifFlag:=true;
  InitializeVGain;
  while MaxDifFlag do
  begin

Weights(Sp_PartWeights,Sp_PartitionSet);
    VertexGainEvaluation(Sp_Set1,Sp_Set2);
    MaxDifFlag:=true;

FindMaxDifference(Sp_PartitionSet,Sp_Partit
ionCutSet,Sp_PartWeights,Sp_Set1,Sp_Set2);
    if MaxDifFlag then
    begin
      ChangeNodes(Sp_PartitionSet);

ChangeCutSet(Sp_Set1,Sp_Set2,Sp_PartitionCu
tSet);

refine2:=EvaluateRefinement(Sp_PartitionCut
Set,Sp_PartWeights);
      if refine2<refine1 then
```

```
        refine1:=refine2
      else flag:=false;
    end;
  end;
end;

procedure LinearRefinement;
var
  refine1,refine2 : integer;
  flag : boolean;
begin
  MaxDifFlag:=true;
  InitializeVGain;
  while MaxDifFlag do
  begin

Weights(L_PartWeights,L_PartitionSet)
;

VertexGainEvaluation(L_Set1,L_Set2);
    MaxDifFlag:=true;

FindMaxDifference(L_PartitionSet,L_Partitio
nCutSet,L_PartWeights,L_Set1,L_Set2);
    if MaxDifFlag then
    begin
      ChangeNodes(L_PartitionSet);

ChangeCutSet(L_Set1,L_Set2,L_PartitionCutSe
t);

refine2:=EvaluateRefinement(L_PartitionCutS
et,L_PartWeights);
      if refine2<refine1 then
        refine1:=refine2
      else flag:=false;
    end;
  end;
end;

end.
```

//Minimum ağırlıklı kapsar ağacı bulan kaynak kod

```
unit SpanTree;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun, graphun2;

  procedure MinimumSpanningTree;
  procedure GetEdgeSet;
  procedure SetVertexSet;

var
  AyritSet2 : array[1..MaxEdge] of
Ayrit;
  Min, MaxEdge2 : integer;
  V1,V2 : set of byte;

implementation


procedure GetEdgeSet;
var
  i : integer;
begin
  i:=1;
  while i<=MaxEdge do
  begin
    if AyritSet[i].Ayrit_No=0 then
    begin
      MaxEdge2:=i-1;
      i:=MaxEdge+1;
    end;
    i:=i+1;
  end;
  for i:=1 to MaxEdge2 do
  begin

AyritSet2[i].Ayrit_Ag:=AyritSet[i].Ay
rit_Ag;

AyritSet2[i].Ayrit_No:=AyritSet[i].Ay
rit_No;

AyritSet2[i].Ayrit_Uc1:=AyritSet[i].A
yrit_Uc1;

AyritSet2[i].Ayrit_Uc2:=AyritSet[i].A
yrit_Uc2;
  end;
end;

procedure SetVertexSet;
var
  i : integer;
begin
  V1:=[];
  V2:=[];
  for i:=1 to Max do
    V2:=V2+[i];
end;

procedure MinimumSpanningTree;
var
  i,j,index : integer;
  flag : boolean;
begin
  GetEdgeSet;
  SetVertexSet;
  flag:=true;
  V1:=[1];
  for i:=1 to (Max-1) do
  begin
    Min:=100;
    for j:=1 to MaxEdge2 do
      if ((AyritSet2[j].Ayrit_Uc1 in V1)
xor (AyritSet2[j].Ayrit_Uc2 in V1)) then
        if AyritSet2[j].Ayrit_Ag<Min then
        begin
          Min:=AyritSet2[j].Ayrit_Ag;
          index:=j;
          flag:=false;
        end;
    if not(flag) then
    begin
      if AyritSet2[index].Ayrit_Uc1 in V1
then
        V1:=V1+[AyritSet2[index].Ayrit_Uc2]
      else if AyritSet2[index].Ayrit_Uc2 in
V1 then

V1:=V1+[AyritSet2[index].Ayrit_Uc1];

SpanTreeSet[i].Ayrit_No:=AyritSet2[index].A
yrit_No;

SpanTreeSet[i].Ayrit_Uc1:=AyritSet2[index].
Ayrit_Uc1;

SpanTreeSet[i].Ayrit_Uc2:=AyritSet2[index].
Ayrit_Uc2;

SpanTreeSet[i].Ayrit_Ag:=AyritSet2[index].A
yrit_Ag;
    end;
  end;
```

```
  SpanTree2:=true;
end;
```

```
end.
```

//Bütün düğümler arasındaki en kısa yolu bulan kaynak kod

```
unit SPath;

interface

uses
  Windows, Messages, SysUtils,
Classes, Graphics, Controls, Forms,
Dialogs,
  Menus, StdCtrls, graphun2;

  procedure ShortestPath;
  procedure Set_ZandW_Matrices;

implementation

procedure Set_ZandW_Matrices;
var
  i,j,k : integer;
begin
  for i:=1 to Max do
    for j:=i+1 to Max do
    begin
      k:=1;
      while AyritSet[k].Ayrit_No>0 do
      begin
        if ((AyritSet[k].Ayrit_Uc1=i)
and (AyritSet[k].Ayrit_Uc2=j))
          or
((AyritSet[k].Ayrit_Uc1=j) and
(AyritSet[k].Ayrit_Uc2=i)) then
        begin

ShortestPaths[i,j]:=AyritSet[k].Ayrit
_Ag;

ShortestPaths[j,i]:=ShortestPaths[i,j
];
        end;
        k:=k+1;
      end;
    end;
  for i:=1 to Max do
    for j:=1 to Max do
      if ((ShortestPaths[i,j]<>32567)
and (ShortestPaths[i,j]<>0)) then
        ShortPathSet[i,j]:=j;
end;

procedure ShortestPath;
var
  i,j,k,M,n : integer;
begin
  Set_ZandW_Matrices;
  ShortPath:=1;
  for k:=1 to Max do
  begin
    for i:=1 to Max do
      for j:=1 to Max do
        if ((i<>k) and (j<>k) and
(ShortestPaths[i,k]<>32567)
          and (ShortestPaths[k,j]<>32567))
then
        begin
          if
ShortestPaths[i,j]<(ShortestPaths[i,k]+Shor
testPaths[k,j]) then
            M:=ShortestPaths[i,j]
          else

M:=ShortestPaths[i,k]+ShortestPaths[k,j];
          if M<ShortestPaths[i,j] then
          begin

ShortPathSet[i,j]:=ShortPathSet[i,k];
            ShortestPaths[i,j]:=M;
          end;
        end;
    for n:=1 to Max do
      if ShortestPaths[n,n]<0 then
      begin
        showmessage('Negatif ağırlıklı bir
çevre var');
        ShortPath:=2;
        exit;
      end;
  end;
end;

end.
```