# SEZGİSEL YÖNTEMLER

# 2019

**Fiduccia-Mattheyses (FM) Algoritması**
**Grafik bölümleme**


**ÖMER M. OĞUZOĞLU**

**Pamukkale Üniversitesi**
**Mühendislik Fakültesi | Bilgisayar Mühendisliği | 2018-2019**
**omer1991mustafa@gmail.com**

# Fiduccia-Mattheyses (FM)

Grafik bölümleme algoritması

**PAMUKKALE ÜNİVERSİTESİ**

**SEZGİSEL YÖNTEMLER VE UYGULAMALARI ARA SINAV PROJE RAPORU – BAHAR 2019**

ÖMER MUSTAFA OĞUZOĞLU
13253813

**Danışman** : Dr. Öğr. Ü. Kenan KARAGÜL

Pamukkale Üniversitesi – Bilgisayar Mühendisliği 2018 Bahar Dönemi- IENG 479 Sezgisel Yöntemler ve Uygulamaları dersine hazırlanmıştır.

**İçindkiler**

## 1.Giriş

Bilgisayar bilimlerinde sezgisel yöntemler bir problem çözme tekniğidir. Sonucun kesin doğru olduğunu önemsememekle beraber kabul edilir sonuçlar verir, ayrıca zaman açısından maksimum verimliliği hedefler.
Sezgisel algoritmalar makul bir süre içerisinde bir çözüm vereceklerini garanti ederler, fakat en iyi sonucu vereceklerini garanti etmezler.
Sezgisel algoritmalar, büyük optimizasyon problemler için kısa süre içerisinde optimuma yakın çözüm üretirler, bu algoritmalar genel olarak biyoloji tabanlı, fizik tabanlı, sürü tabanlı, sosyal tabanlı, müzik tabanlı ve kimya tabanlı olmak üzere altı farklı grupta değerlendirilmektedir. Sürü zekâsı tabanlı optimizasyon algoritmaları kuş, balık, kedi ve arı gibi canlı sürülerinin hareketlerinin incelenmesiyle geliştirilmiştir.[1]

**Sezgisel optimizasyon yöntemlerine örnek olarak:**

- Optikten Esinlenen Optimizasyon (Optic Inspired Optimization)(OIO)
- Genetik Algoritma (Genetic Algorithm)(GA)
- Karınca Kolonisi Optimizasyonu (Ant Colony Optimization)(ACO)
- Parçacık Sürü Optimizasyonu (Particle Swarm Optimization)(PSO)
- Yapay Arı Kolonisi (Artificial Bee Colony)(ABC)
- Benzetim Tavlama (Simulated Annealing)(SA)
- Su Döngüsü Optimizasyon Algoritması
- Krill Sürü Optimizasyon Algoritması
- Bakteri Yiyecek Arama Davranışı
- Yarasa Algoritması
- Yapay Alg Algoritması
- Virüs Koloni Arama Algoritması
- Ağaç-Tohum Algoritması(Tree-Seed Algorithm)(TSA)

## 2.Grafik bölümleme:

Matamatikte bir G=(V,E) grafiğini (V=köşe noktası, E=kenar) belirli özellikli küçük parçalara bölmek mümkün. Grafik bölümlemenin en önemli uygulamalarından bilimsel hesaplama, VLSI devre tasarımı ve çok işlemcili sistemlerde görev planlamasıdır. En yaygın Grafik bölümleme algoritmalarından birisi Kernighan-Lin (KL) Algoritması ve diğeri Fiduccia-Mattheyses (FM) algoritmasıdır. [2] Bu raporda KL algoritmasını özet geçtikten sonra FM algoritması detaylı şekilde incelenecek.

### GENETİK ALGORİTMA

**NP-Hard problemleri çözmek için kullanılır, Karmaşık çok boyutlu arama uzayında en iyinin hayatta kalması ilkesine göre bütünsel en iyi çözümü arar, Genetik algoritmaların temel ilkeleri ilk kez Michigan Üniversitesi'nde John Holland tarafından ortaya atılmıştır.**

### VLSI

**Çok geniş ölçekli integer-rasyon (Very Large Scale Integration) binlerce transistörün tek bir yonga üzerinde birleştirilmesi ile Tümleşik Devrelerin oluşturulması işlemidir**

## 2.1 Grafik bölümleme algoritmalarının genel kullanımı:

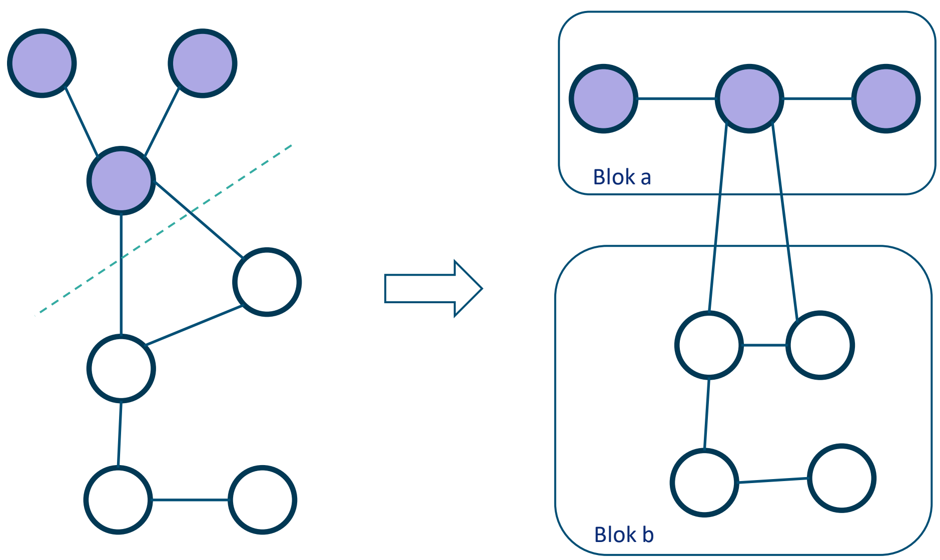
- Divide-and-conquer algoritması
- Devre düzeni ve tasarımları
- Parelel hesaplama
- Biyoinformatik

## 2.2 Grafik bölümleme algoritmalarının büyük grafiklerdeki kullanımı:

- internette arama
- toplulukları tanımlama
- hedefleri izleme
- spam linkleriyle mücadele
- devre bölümleme (VLSI)

## 2.3 Netlist ve Sistem Bölümleme

Son zamanlarda modern entegre devrelerinin tasarım karmaşıklığı eşi görülmemiş bir büyüklüğe ulaştı, Ful-Çip düzeni yapımı, FPGA tabanlı devreler ve benzeri modeller gittikçe daha da zorlaşmaktadır. Genel strateji bu tasarımları daha küçük ve basit parçalara bölümlemek, bunların her biri paralellik ve bağımsızlık derecesi ile işlenebilir. Çip tasarımı için kullanılabilen Divide-and-conquer stratejisi manuel bölümleme için kullanılır fakat bu yöntem Büyük ağlar (Netlist) için pek mümkün değil. Manuel çözüm yerine sezgisel yöntem kullanmak daha uygun olacaktır. [4]




## 2.4 Kernighan-Lin (KL) Algoritması:

Kernighan-Lin, köşe noktası ve kenarları olan bir grafiği ayrı alt kümelere böler, örneğin bir elektrik şebekesini grafik şeklinde temsil edebiliriz bu yüzden Kernighan-Lin algoritmasını kullanmak mümkün olacaktır. Kernighan-Lin deterministik bir algoritmadır, Dolaysıyla algoritmayı her kullandığımızda aynı sonucu elde ederiz. Aynı sonuç, aynı ağlar ve düğümler olmasa da, bölümler arasındaki geçişlerin maliyeti aynı olacaktır. [3]

## 3. Fiduccia-Mattheyses (FM) algoritması

Sezgisel parçalama olan bu algoritma, 1982 yılında C.M. Fiduccia ve R.M. Mattheyses tarafından paylaşıldı. Ağırlığı olan bir G(V,E) grafiği verildiğinde hedefimiz bu grafikteki tüm köşe noktalarından ayrık bölümler oluşturmak olacak, böylece amaç tüm kesilmiş ağların (nets) bölme boyutu kısıtları da dikkate alınarak ağırlık maliyetlerini en aza indirgemek olur. FM algoritması KL algoritmasına benzer bir fikri taşır, KL algoritması her seferde iki düğüm taşırken FM algoritması bir düğüm taşır.

Düğümlerin kazancı hesaplanır ve kazancı maksimum olan düğüm karşı bölgeye geçer. Eğer tüm düğmeler tek bir bölüme geçecek olursa o son işlemde denge ihlali oluşur bu yüzden o düğmeyi baz olarak seçmekten vazgeçilir ve bir diğer maksimum kazancı olan düğme (köşe noktası) seçilir ve taşınır. İşlemler aşağıda belirtilen örneklerde daha detaylı açıklanacaktır.[4]

### 3.1 zaman karmaşıklığı:

- Taşımak için en iyi düğmeyi bulma zamanı sabit.
- Toplam zaman karmaşıklığı O(n), n düğmelerin toplam sayısı.

### 3.2 KL algoritmasından farklı yanları:

- Her taşımada çift düğme yerine tek düğme taşır.
- Düğmelerin toplam sayısı çift olmak zorunda değil.

### 3.3 özellikleri:

- Net-cut (ağ kesimleri) maliyetini azaltmayı hedefler.
- Ağırlığı olan grafikler de çok iyi çalışır.
- Her defasında Sadece bir köşe noktası taşınır.
- Dengesiz bölümlerle baş edebilir.
- Bölüm alanına taşınacak köşe noktası için özel bir veri yapısı kullanılır.

### 3.4 Aşamalar:

- Tüm hücrelerin kazancını hesapla, **$\Delta g(c) = FS(c) - TE(c)$**
- Baz hücreyi seç ($\Delta g$ maksimum olduğu halde o köşe baz olur) ve bu hücreyi taşı.
- Baz hücreyi kilitle (diğer iterasyonlara dahil edilmez).
- Yeni oluşumdaki kazançları güncelle ve yeni baz hücreyi seç.
- En iyi dizilimi seç.
- Gerçekleştirilmiş taşınmaları sabitle.

### 3.5 Uygulama:

**Örnek: kazançları hesaplama**

**a için**: kesişim yok o halde FS(a)=0, n1 bağlı ama kesilmedi O halde TE(a)=1, Δg(a) = -1

**b için**: n4 ve n5 ağları kesildi o halde FS(b)=2, n1 ve n2 b ye bağlı ve kesilmedi o halde TE(b)=2, Δg(b) = 0

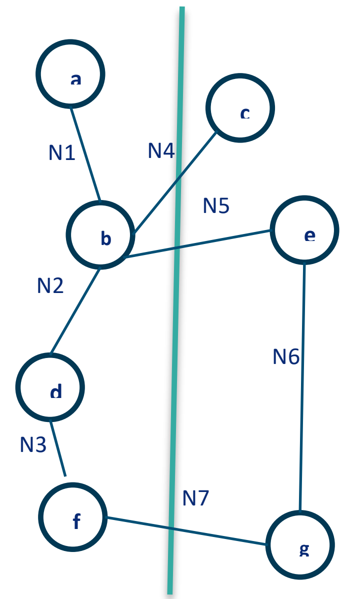
**c için:** FS(c)=1 , TE(c)=0, Δg(c) = 1

**d için:** FS(d)=0 , TE(d)=2, Δg(d) = -2

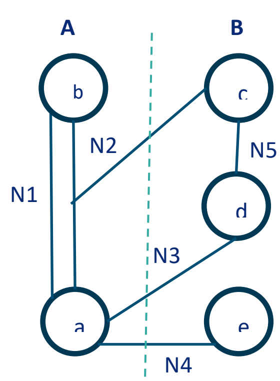
**e için:** FS(e)=1 , TE(e)=1, Δg(e) = 0

**f için:** FS(f)=1 , TE(f)=1, Δg(f) = 0

**g için:** FS(g)=1 , TE(g)=1, Δg(g) = 0




**Örnek: FM algoritması (adım-adım)**




0. iterasyon

**Verilenler:**

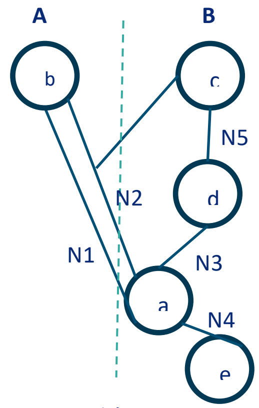
1. Ağırlıklı hücreler a-e, 2. oran faktörü r= 0.3755, 3. Ağları N1-N5 ve

4. Baş bölüm (sağ taraf)

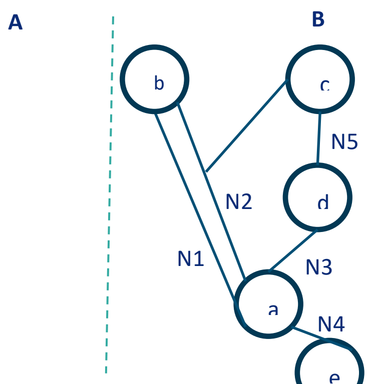
alan(a)=2, alan(b)=4, alan(c)=1, alan(d)=4, alan(e)=5

**Soru**: FM algoritmasının ilk geçişini gerçekleştir.

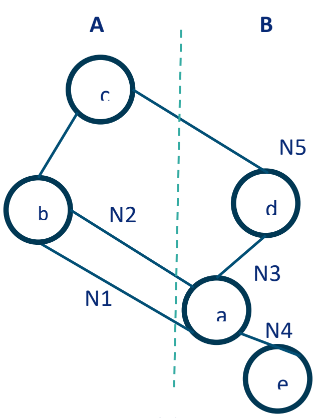
**Çözüm:**




1.iterasyon




Denge İhlali




2.iterasyon

**Denge kriterini hesaplamak:**

$$\boxed{r \cdot alan(V) - alan_{max}(V) \leq alan(A) \leq r \cdot alan(V) + alan_{max}(V)}$$

$r \cdot alan(V) - alan_{max}(V) = 0.375 \times 16 - 5 = 1$

$r \cdot alan(V) + alan_{max}(V) = 0.375 \times 16 + 5 = 11$

aralık: $1 \leq alan(A) \leq 11$

**1. İterasyon:**

**her bir hücrenin kazancını hesaplamak:** (0. İterasyondaki şekle göre kazanç hesabı)

a için : N3 ve N4 kesilmiş o halde: FS(a)=2, N2 kesilmemiş: TE(a)=1,
Δg(a) = 1

Aynı şekilde diğerleri de:

b: FS(b)=0 , TE(b)=1, Δg(b) = -1
c: FS(c)=1 , TE(c)=1, Δg(c) = 0
d: FS(d)=1 , TE(d)=1, Δg(d) = 0
e: FS(e)=1 , TE(e)=0, Δg(e) = 1

**baz hücreyi seçmek:**

Δg'si maksimum olan seçeriz, burada mümkün seçimler a ve e'dir.
a'yı seçtikten sonraki denge kriteri: alan(A)=alan(b)=4
e'yı seçtikten sonraki denge kriteri: alan(A)= alan(a)+alan(b)+alan(e)=11

**baz hücremizi a olarak seçelim:**

Δg değerlerini güncelledikten sonra:
bölümler: $A_1$={b}, $B_1$={c,d,a,e}/ daha önce seçilmiş {a} diğer iterasyonlara dahil edilmeyecek.

2. **İterasyon:**

Kazanç:
(1. İterasyondaki şekle göre)

b: FS(b) = 2, TE(b) = 0, Δg1(b) = 2
c: FS(c) = 0, TE(c) = 1, Δg1(c) = -1
d: FS(d) = 0, TE(d) = 2, Δg1(d) = -2
e: FS(e) = 0, TE(e) = 1, Δg1(e) = -1

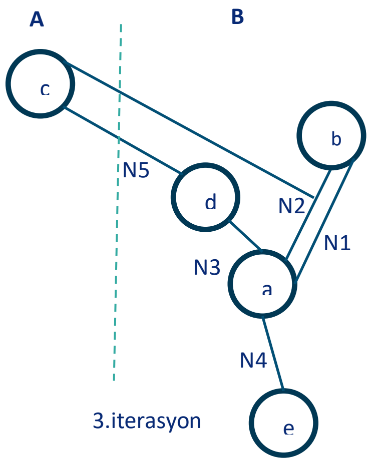
Bu kez Baz hücremiz b'dir çünkü Δg2(b) = 2 maksimum olandır ve

Alan (A)=0 **<u>denge kriteri ihlali O halde bir diğer maksimum kazanca bakalım</u>**

Δg2(c) = -1 ve alan(A)=5 olur, Δg2(e) = -1 ve alan(A)=9 olur bunlardan birini seçebiliriz

**c'yi baz olarak seçelim:**

bölümler: $A_2$={b,c}, $B_2$={a,d,e} / daha önce seçilmiş {a,c}




3.iterasyon

### 3. İterasyon:

Kazanç:

(2. İterasyondaki şekle göre)

b: FS(b) = 2, TE(b) = 1, Δg3(b) = 1

d: FS(d) = 1, TE(d) = 1, Δg3(d) = 0

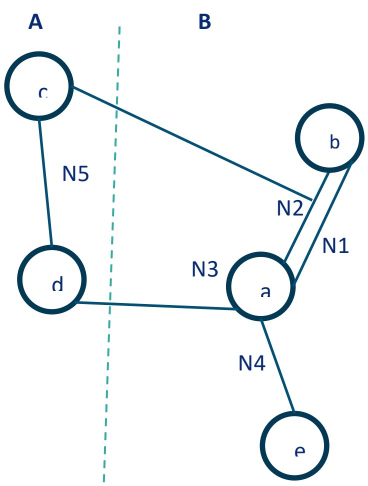
e: FS(e) = 0, TE(e) = 1, Δg3(e) = -1

b hücresi maksimum o halde:

Δg3(b) = 1 alan(A)=1 olur, denge kriterine uygun

**Baz hücremiz b'dir :**

Bölümler: $A_3$={c}, $B_3$={a,b,d,e}/ {a,c,b}




4.iterasyon

### 4. İterasyon:

Kazanç:

(3. İterasyondaki şekle göre)

d: FS(d) = 1, TE(d) = 1, Δg4(d) = 0

e: FS(e) = 0, TE(e) = 1, Δg4(e) = -1

d hücresi maksimum o halde:

Δg4(d) = 0, alan(A)=5 denge kriterine uygun

**Baz hücremiz d:**

Bölümler: $A_4$={c,d}, $B_4$={a,b,e} /{a,c,b,d}

### 5. İterasyon:

Kazanç:

(4. İterasyondaki şekle göre)

e: FS(e) = 0, TE(e) = 1, Δg5(e) = -1

Δg5(e) = -1, Alan(A)=10, denge kriteri uygun

**Baz hücremiz e'dir:**

Bölümler: $A_4$={c,d,e}, $B_4$={a,b}/ **tüm hücreler dahil edilmiştir**.

**En iyi dizlimi seçmek:** seçilmiş bazlara göre

$G_1$= Δg1(a) = 1

$G_2$= Δg1(a)+ Δg2(c) =0

$G_3$=Δg1(a)+ Δg2(c) +Δg3(b) = 1

$G_4$=Δg1(a)+ Δg2(c) +Δg3(b)+ Δg4(d) = 1

$G_5$=Δg1(a)+ Δg2(c) +Δg3(b)+ Δg4(d)+ Δg5(e)=0

**Maksimum pozitif kümülatif kazanç:** $\boxed{Gm = \sum_{i=1}^{m} \Delta \mathbf{gi} = 1}$ **1,3 ve 4. İterasyonda**

en iyi denge oranı M=4,çünkü en küçük aana sahip (alan(A)=5)1
**birinci geçişin sonucu**: 4. İterasyondaki grafik gibidir Bölümler: $A_4$={c,d}, $B_4$={a,b,e}.
**Diğer geçişler aynı yöntemle bulunur.**

### 3.6 Fiduccia-Mattheyses (FM) algoritması sözde kodu:[4]

```
Fiduccia-Mattheyses Algorithm
Input: graph G(V,E), ratio factor r
Output: partitioned graph G(V,E)
(lb,ub) = BALANCE_CRITERION(G,r)   // compute balance criterion
(A,B) = PARTITION(G)               // initial partition
Gm = ∞
while (Gm > 0)
  i = 1
  order = Ø
  foreach (cell c ∈ V)             // for each cell, compute the
    Δg[i][c] = FS(c) – TE(c)       //   gain for current iteration,
    status[c] = FREE               //   and set each cell as free
  while (!IS_FIXED(V))             // while there are free cells, find
    cell = MAX_GAIN(Δg[i],lb,ub)   //   the cell with maximum gain
    ADD(order,(cell,Δg[i]))        // keep track of cells moved
    critical_nets = CRITICAL_NETS(cell) // critical nets connected to cell
    if (cell ∈ A)                  // if cell belongs to partition A,
      TRY_MOVE(cell,A,B)           //   move cell from A to B
    else                           // otherwise, if cell belongs to B,
      TRY_MOVE(cell,B,A)           //   move cell from B to A
    status[cell] = FIXED           // mark cell as fixed
    foreach (net net ∈ critical_nets) // update gains for critical cells
      foreach (cell c ∈ net, c ≠ cell)
        if (status[c] == FREE)
          UPDATE_GAIN(Δg[i][c])
    i = i + 1
  (Gm, m) = BEST_MOVES(order)      // move sequence c1 … cm that
                                   //   maximizes Gm
  if (Gm > 0)
    CONFIRM_MOVES(order,m)         // execute move sequence
```

## 4. Projeyi programlama

### 4.1 Bir örnek

Bu örnekte FM algoritmasının ağırlıksız köşe noktalı bir grafiği incelenecektir, bundan dolay denge kriterimiz toplam köşe noktasının yarısı olarak belirlenecek, ve bu işleme en yakın olanı makbul olarak sayılacaktır.

7 toplam köşe noktası ve 9 toplam ağ sayısı olan bir grafiği işleyelim.

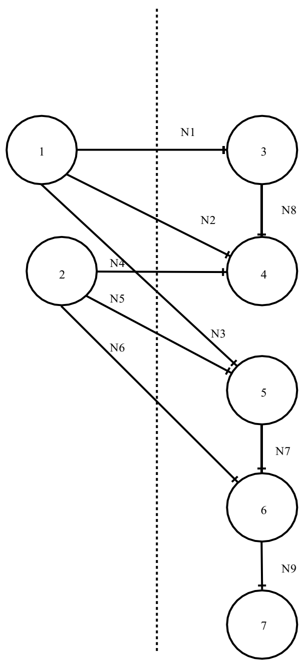
**Bölmeler**: alanA ve alanB

**Baş bölme**: alanA

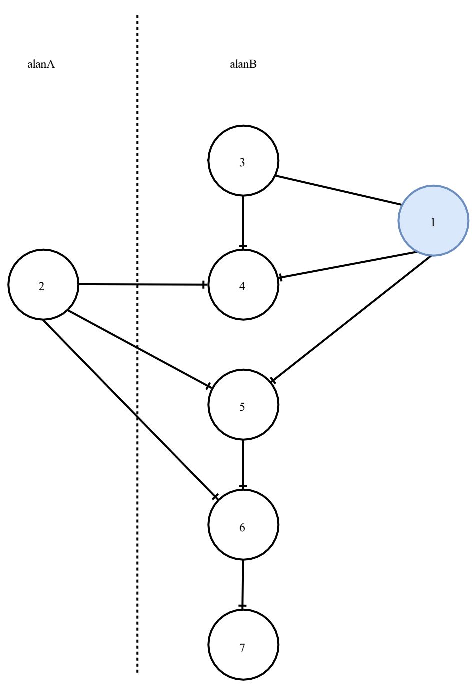
**Denge**: 7/2=3.5, Round(3.5)=4

İterasyonlarda maksimim komulatif kazancı bulduktan sonra alanA daki düğme sayısı 4 en yakın olan makbul olacaktır.

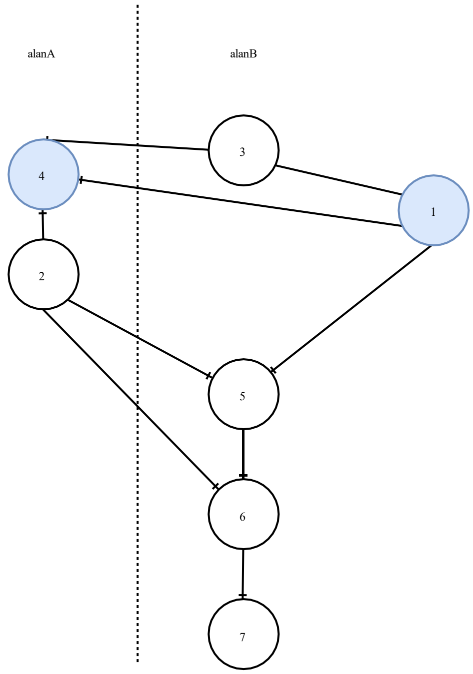
Bu örnek sadece Tablo ve köşe noktalarını boyayarak daha net bir çizimle açıklanacak.



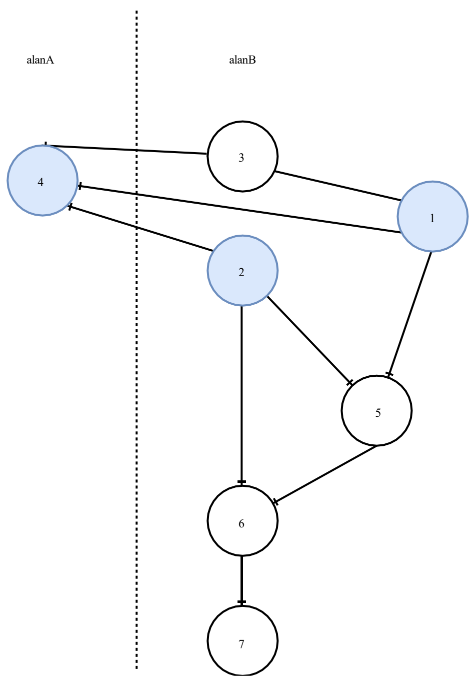

| Node | Kazanç |
|---|---|
| 1 | 3 |
| 2 | 3 |
| 3 | 0 |
| 4 | 1 |
| 5 | 1 |
| 6 | -1 |
| 7 | -1 |



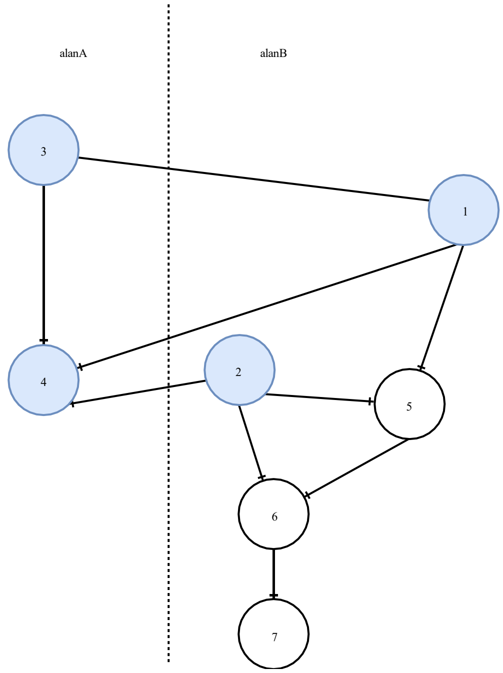

| Node | Kazanç | |
|---|---|---|
| 1 | -3 | |
| 2 | 3 | Denge ihlali |
| 3 | -2 | |
| 4 | -1 | |
| 5 | -1 | |
| 6 | -1 | |
| 7 | -1 | |
| Iterason | 1 | |



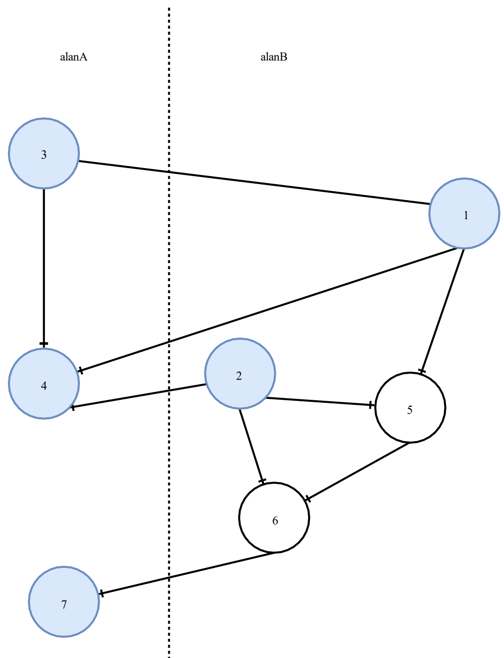

| Node | Kazanç |
| --- | --- |
| 1 | -1 |
| 2 | 1 |
| 3 | 0 |
| 4 | 1 |
| 5 | -1 |
| 6 | -1 |
| 7 | -1 |
| Iterason | 2 |



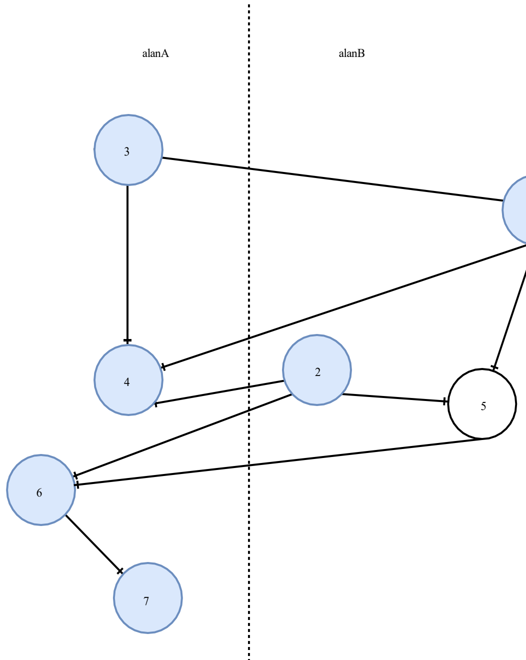

| Node | Kazanç |
| --- | --- |
| 1 | -1 |
| 2 | -1 |
| 3 | 0 |
| 4 | 3 |
| 5 | -3 |
| 6 | -3 |
| 7 | -1 |
| Iterason | 3 |



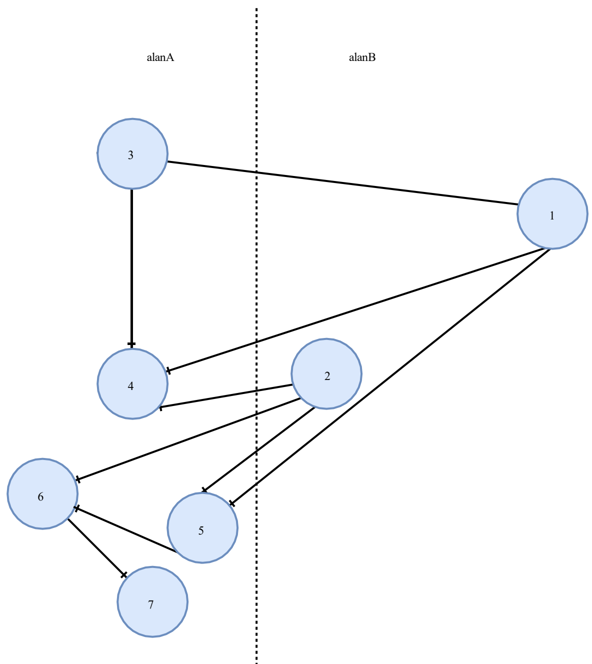

| Node | Kazanç |
|---|---|
| 1 | 1 |
| 2 | -1 |
| 3 | 0 |
| 4 | 1 |
| 5 | -3 |
| 6 | -3 |
| 7 | -1 |
| Iterason | 4 |




| Node | Kazanç |
|---|---|
| 1 | 1 |
| 2 | -1 |
| 3 | 0 |
| 4 | 1 |
| 5 | -3 |
| 6 | -1 |
| 7 | 1 |
| Iterason | 5 |




| Node | Kazanç |
|---|---|
| 1 | 1 |
| 2 | 1 |
| 3 | 0 |
| 4 | 1 |
| 5 | -1 |
| 6 | 1 |
| 7 | -1 |
| Iterason | 6 |




| Node | Kazanç |
|---|---|
| 1 | 3 |
| 2 | 3 |
| 3 | 0 |
| 4 | 1 |
| 5 | 1 |
| 6 | -1 |
| 7 | -1 |
| Iterason | 7 |

| Iterasyon | maximum kazanc(denge şartı dahil |
|---|---|
| 1 | 3 |
| 2 | -1 |
| 3 | 1 |
| 4 | 0 |
| 5 | -1 |
| 6 | -1 |
| 7 | -1 |

| Komulatif kazanc | degeri |
|---|---|
| G1 | 3 |
| G2 | 2 |
| G3 | 3 |
| G4 | 3 |
| G5 | 2 |
| G6 | 1 |
| G7 | 0 |

Maximum komulatif 1. , 3. Ve 4. Iterasyonda – bu üç iterasyon arasında maksimum düğüm taşıyan ve Denge durumundan küçük olan (round(7/2) ) **iterasyon 4 tür**

## 4.2 Julia da kodlama

### 4.2.1 Anasayfa (1MainFile.jl )

Kodlamada alanA, alanB, net ve nod değişkenlerdeki 0 değeri orda bir değer olmadığını gösterir Yazılmış koda göre her bir köşe noktasında maksimum 3 ağ olabilir ve her bir ağ iki köşe noktasına bağlı olması şart.

**[1 3]; # net, her ağdaki köşe noktalarıdır**

**[6 7 9];#node, köşe noktasındaki ağlardır**

```
nodSayisi=7# kose sayisini belirle
netsaysisi=9
###### her net(ağ) sadece iki nokta arasinda olmak zorunda
net=[
[1 3];# net1
[1 4];# net2
[1 5];# net3
[2 4];# net4
[2 5];# net5
[2 6];# net6
[5 6];# net7
[3 4];# net8
[6 7]]# net9
###### her nodda(kose noktasi) en fazla 3 net olabilir 0 sayisi nod
nodNet=[
[1 2 3];#node1
[4 5 6];#node2
[1 8 0];#node3
[2 4 8];#node4
[3 5 7];#node5
[6 7 9];#node6
[9 0 0]]#node7
alanA=[1,2]
alanB=[3,4,5,6,7]
```

### 4.2.2 kazanc heasplamak (CalcGain.jl ) ve (CalcGainToArray.jl)

Bu iki fonksyonu tek fonksyon gibi düşünebiliriz, kazançları hesaplamak için kullanılacak FC bir noda bağlı olan ve kesilen ağları gösteriyor TE aynı noda bağlı olan ve kesilmeyen ağları gösteriyor Dolaysıyla kazanç= FC-TE

```
function CalcGain(MyNode) #[1 0 0] gibi olan nodu netlerini belir
    BlunanNodes=[0,0]
        FC=0
        TE=0
        for j=1:3
            if MyNode[j]!=0
                BlunanNodes=findNodesOfNet(MyNode[j])
                yeniBulunan=ThisNetNodesInWhichPart(BlunanNodes)
                alanfirst=yeniBulunan[1]
                alansecond=yeniBulunan[2]
                if alanfirst==alansecond
                    TE=TE+1
                else
                    FC=FC+1
                end
            end
        end
    return FC-TE
end
```

```
function CalcGainToArray(nodeNetArray) # butun nodlarin ka
    resultArray=[i=0 for i=1:nodSayisi]
    temp=[0,0,0]
    for i=1:nodSayisi
        for j=1:3
            temp[j]=nodeNetArray[i,j]
        end
        resultArray[i]=CalcGain(temp)
    end
    return resultArray
end
```

### 4.2.3 Maksimum komulatif kazanca göre en dengeli iterasyonu bulmak (FindBestIter.jl )

Eğer birden fazla maksimum komulatif varsa onların en dengelisini bulmak için bu maksimum noktalara bağlı olan baz alandaki düğmelri sayar ve round(nodsaysıs/2) en yakın olan iterasyonu seçer. Sadece bir makimum varsa o en iyi İterasyon seçilir.

```
function FindBestIter(p2) # en iyi iterasyonu bulmak
    iter=-5
    CounNode=0
    countdizi=[i=0 for i=1:length(p2) ]
    if length(p2)==1
        iter=p2
    else
        roundme=round(nodSayisi/2)
        for i=1:length(p2)
            for j=1:nodSayisi-1
                pow=newalanA[p2[i],j]
                if pow!=0
                    CounNode+=1
                end
            end
            if CounNode>roundme
                countdizi[i]=0
            else
                countdizi[i]=CounNode
            end
        end
        cctv=findmax(countdizi)
        iter=p2[cctv[2]]
    end
    return iter
end
```

4.2.4 bir noda bağlı olan ağları verir (findNetsOfNode.jl)

```
function findNetsOfNode(y)# findNetsOfNode(5)= 3 4 0 s
    BlunanNodes=[0,0,0]
    counter=1
    for i=1:nodSayisi
        if y==i
            for j=1:3
                BlunanNodes[counter]=nodNet[i,j]
                counter=counter+1
            end
        end
    end
    return BlunanNodes
end
```

4.2.5 bir ağa bağlı olan nodları verir (findNodesOfNet.jl)

```
function findNodesOfNet(y) # findNodesOfNet(5)= 2 4  siras
    BlunanNodes=[0,0]
    counter=1
    for i=1:netsaysisi
        if y==i
            for j=1:2
                BlunanNodes[counter]=net[i,j]
                counter=counter+1
            end
        end
    end
    return BlunanNodes
end
```

4.2.6 seçilen nodu bir dahaki kazanç hesabına dahil etmez(FixChoosen.jl)

```
function FixChoosen()# secileni atla
    kazanc2=[i=0 for i=1:nodSayisi]
    kazanc2=CalcGainToArray(nodNet)
    for j=1:length(temp) # fix choosen -50 en kucuk de
        if temp[j]!=0
            kazanc2[temp[j]]=-50
        end
    end
    return kazanc2
end
```

4.2.7 seçilen nodu diğer alana taşımak için (moveToOtherPart.jl)

```
function moveToOtherPart(node) #bir node u diger parta tasimak
    test=0
    Dengeli=0
    for i=1:length(alanA)
        if alanA[i]==node&&test!=2
            for j=1:length(alanB)
                if alanB[j]==0
                    alanB[j]=node
                    test=1
                    alanA[i]=0
                    if sum(alanA)==0
                        dengeli=-1
                    else
                        Dengeli=1
                    end
                    break
                end
            end
        elseif node==alanB[i]&&test==0
            for j=1:length(alanA)
                if alanA[j]==0
                    alanA[j]=node
                    test=2
                    alanB[i]=0
                    if sum(alanB)==0
                        dengeli=-1
                    else
                        Dengeli=1
                    end
                    break
                end
            end
        end
    end
```

4.2.8 bir nodun hangi alanda olduğunu döndürür(ThisNodeInWhichPart.jl)

```
function ThisNodeInWhichPart(node) #kose noktasinin alanA do
    whichArea=0
    for i=1:length(alanA)
        if node==alanA[i]&&alanA[i]!=0
            whichArea=1
            break
        elseif node==alanB[i]alanB[i]!=0
            whichArea=2
        end
    end
    return whichArea
end
```

4.2.9 Sonuç (1MainFile.jl)

1MainFile.jl çalıştırıldığında yukarıdaki işlenen örneğin aynısı aynı şekilde gösterilecektir.

FM algoritması iki bölme arasındaki bağlantı köprüsünü mümkün olan en aza indirmeyi ve dengeli bir grafik oluşturmayı hedefler.

Bu örneği ele aldığımızda algoritma 1,3 ve 4 üncü iterasyonlarda minimum köprü sayısını elde etmiştir ancak bu üç iterasyondan en dengeli durumu seçmesi gerekmektedir,

1.iterasyonda alanA da 1 node vardır

3. iterasyonda alanA da 1 node vardır

4.iterasyonda alanA da 2 node vardır

Toplam nod sayısı= 7

Denge kriteri= round(7/2)=4

ve alanA daki node sayısı denge kriterine yakınlaştıkça grafik daha dengeli olacaktır,

**buna göre en dengeli durum 4. iterasyondadır**

```
kazanc: [3, 3, 0, 1, 1, -1, -1]
1.iterasyonda da alan A= { 2 } ve alan B= { 3  4  5  6  7  1 }

kazanc: [-3, 3, -2, -1, -1, -1, -1]
2.iterasyonda da denge ihlali tespiti

Diger maksimum kazanc degeri=:-1, indexi=:4
2.iterasyonda da alan A= { 4  2 } ve alan B= { 3  5  6  7  1 }

kazanc: [-1, 1, 0, 1, -1, -1, -1]
3.iterasyonda da alan A= { 4 } ve alan B= { 3  2  5  6  7  1 }

kazanc: [-1, -1, 0, 3, -3, -3, -1]
4.iterasyonda da alan A= { 4  3 } ve alan B= { 2  5  6  7  1 }

kazanc: [1, -1, 0, 1, -3, -3, -1]
5.iterasyonda da alan A= { 4  3  7 } ve alan B= { 2  5  6  1 }

kazanc: [1, -1, 0, 1, -3, -1, 1]
6.iterasyonda da alan A= { 4  3  7  6 } ve alan B= { 2  5  1 }

kazanc: [1, 1, 0, 1, -1, 1, -1]
7.iterasyonda da alan A= { 4  3  7  6  5 } ve alan B= { 2  1 }

Komulatif kazanc degeri:
G1= 3
G2= 2
G3= 3
G4= 3
G5= 2
G6= 1
G7= 0
Dengeli durumda olma sartiyla maximum kazanc:s
Iterasyon1= 3
Iterasyon2= -1
Iterasyon3= 1
Iterasyon4= 0
Iterasyon5= -1
Iterasyon6= -1
Iterasyon7= -1
en iyi iterasyon=  4
En iyi alan A dagilimi=  4 3 0 0 0 0
En iyi alan B dagilimi=  0 2 5 6 7 1
```

## 5. kaynakça


1. http://mm.iit.uni-miskolc.hu/data/texts/BOOKS/Artificial_Intelligence2/node23.html "Multimedia Maniacs Artificial Intelligence"
2. Andreev, Konstantin; Räcke, Harald (2004). *Balanced Graph Partitioning*. *Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*. Barcelona, Spain. pp. 120–124
3. Sait, S.M., and Youssef, H.: "VLSI Physical Design Automation", McGraw-Hill Book Company, 1995.
4. Andrew B. Kahng • Jens Lienig Igor L. Markov • Jin Hu "VLSI Physical Design From Graph Partitioning to Timing Closure", Springer Science+Business Media B.V. 2011.