Bölüm 8: Ağ Tasarımı ve Yönlendirme

Algoritmik Oyun Kuramı

Yönlendirme temelleri | Bencil yönlendirme | Wardrop dengesi | Braess paradoksu | Tıkanıklık oyunları | Rosenthal potansiyeli | Ağ oluşturma oyunları

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_08_Ag_Tasarimi_ve_Yonlendirme.html

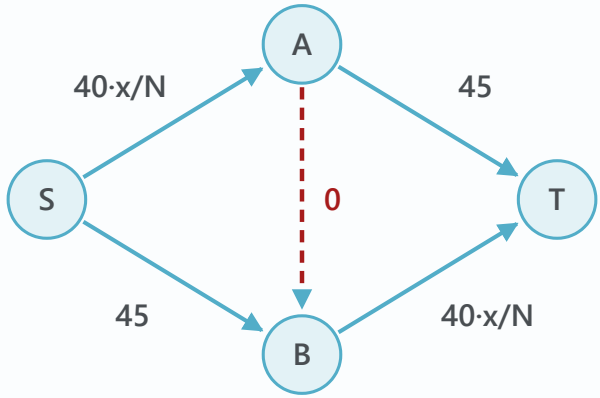
Oyunla Başlangıç: Braess Paradoksu

Oyunun kuralı

Her sürücü S'den T'ye kendi en hızlı yolunu seçer; dar yolların süresi o yolu kullanan sürücü sayısıyla artar, geniş yollarınki sabit 45 dakikadır.

Merak sorusu

- A'dan B'ye süresi 0 olan bir kestirme açılırsa yolculuklar kısalır mı?
- Herkes kendi en hızlı yolunu seçince ortalama süre en küçük olur mu?



N sürücü S → T; x: o yoldaki sürücü sayısı
kesikli: yeni kestirme A → B

Braess Paradoksu oyunu →

Öğrenme Çıktıları

Wardrop dengesini bulmak

İki yollu bir ağda gecikmeleri eşitleyerek dengeyi hesaplamak; sistem optimumunu marjinal maliyetlerle bulup dengeyle karşılaştırmak.

Braess paradoksunu açıklamak

Yeni bir yolun herkesin süresini neden artırabildiğini sayısal olarak göstermek.

Potansiyel ile düşünmek

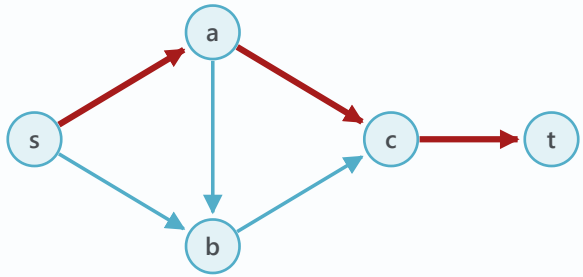
Rosenthal potansiyelini hesaplamak; en iyi tepki dinamiğinin neden durduğunu kanıtlamak.

Ağ oluşturmayı modellemek

Bağlantı kuran oyuncuların hangi ağlarda dengeye geldiğini kontrol etmek.

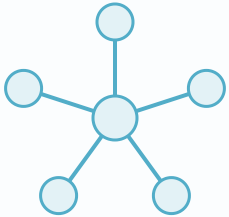
Ağ Tasarımı ve Yönlendirme

- **Ağ tasarımı** (network design): düğümleri, bağlantıları, kapasiteleri ve maliyetleri belirler.
- **Yönlendirme** (routing): akışın kaynak ile hedef arasında hangi yolları kullanacağını belirler.
- Algoritmik oyun kuramı, kullanıcıların *kendi* maliyetlerini azaltmaya çalıştığı trafik ve iletişim ağlarını inceler.
- Temel uygulamalar: şehir trafiği, internet omurgaları, enerji şebekeleri.

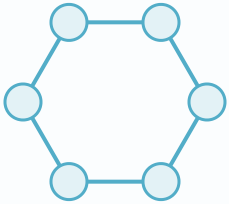


Tasarım: hangi kenar, hangi kapasite?
Yönlendirme: akış hangi yoldan? (kırmızı)

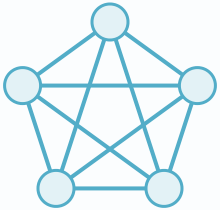
Temel Ağ Tasarım Kavramları



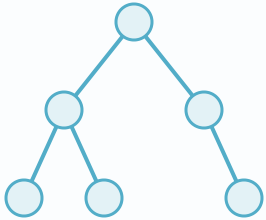
Yıldız



Halka



Tam örgü (mesh)



Ağaç

Düğüm ve bağlantı

Düğüm (node): kavşak, yönlendirici, trafo. Bağlantı (link): yol, fiber hat, iletim hattı.

Kapasite ve bant genişliği

Bir bağlantının birim zamanda taşıyabildiği akış; aşılınca gecikme hızla artar.

Maliyet

Kurulum maliyeti (tasarım) ve kullanım maliyeti: gecikme, enerji, ücret (yönlendirme).

Örnek

Veri merkezinde ağaç/katmanlı topoloji; şehir trafiğinde örgüye yakın yol ağı.

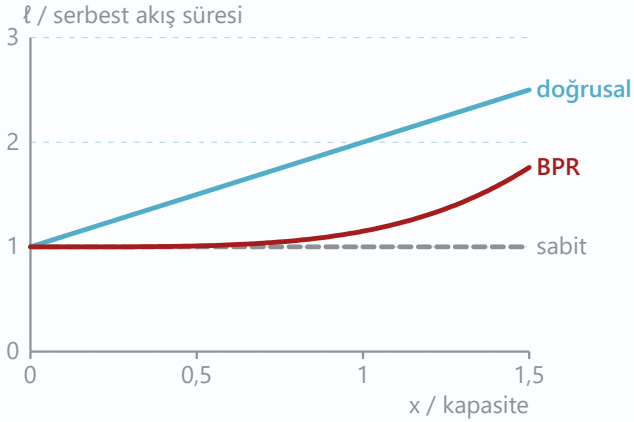
Klasik Yönlendirme Algoritmaları

Yöntem	Ne yapar?	Not
Dijkstra	Negatif olmayan ağırlıklarla tek kaynaktan en kısa yollar	O((V + E) log V); OSPF'nin (en kısa yolu önce açan bağlantı durumu protokolü) çekirdeği
Bellman–Ford	Negatif ağırlıklara izin verir, komşulardan öğrenerek günceller	O(V·E); dağıtık sürümü RIP (uzaklık vektörü protokolü)
Dinamik yönlendirme	Bağlantı durumu ve maliyet değişince yolları yeniden hesaplar	Trafik ve arızalara uyum
Yük dengeleme	Trafiği birden fazla yol veya sunucu arasında dağıtır	Ör. ECMP (eşit maliyetli çok yol)

Klasik algoritmalar kenar ağırlıklarını **sabit** varsayar. Gerçekte gecikme, o yolu kaç kişinin seçtiğine bağlıdır; stratejik kullanıcılar önerilen rotadan sapabildiğinde ağ dengesi ile sistem optimumu farklılaşır. Bu, yönlendirmeyi bir **oyuna** dönüştürür.

Gecikme Yüke Bağlıdır: $\ell_e(x)$

- Her kenarın gecikmesi, üzerindeki akış x'in **azalmayan** bir fonksiyonudur: $\ell_e(x)$.
- **Sabit**: geniş otoyol, $\ell = b$.
- **Doğrusal**: $\ell = a \cdot x + b$ (iki yollu ağ ve Braess örnekleri).
- **BPR** (ABD Karayolları Bürosu): $\ell = t_0(1 + 0{,}15\,(x/c)^4)$; t_0: boş yoldaki süre, c: yolun kapasitesi. Trafik planlamasında standart.
- Ağlarda kuyruk gecikmesi: $\ell = 1/(c - x)$, yalnızca $\mathbf{0 \le x < c}$ için (tek sunuculu kuyruk, M/M/1; c: saniyede işlenebilen, x: saniyede gelen paket sayısı). x, c'ye yaklaştıkça gecikme sınırsız büyür; $x \ge c$'de kuyruk durmadan uzar.



Bencil Yönlendirme (Selfish Routing)

- Her kullanıcı kendi gecikmesini en aza indiren rotayı seçer.
- Kullanıcılar çok sayıda ve küçükse akış **bölünebilir** (nonatomic) kabul edilir: tek kişinin etkisi ihmal edilir.
- Bireysel rota seçimleri, toplam gecikmeyi en aza indiren **sistem optimumundan** uzaklaşabilir.
- Braess paradoksunda yeni bir bağlantı eklemek denge gecikmesini artırabilir.

Model

r birim trafik $s \to t$

f_P: P yolundaki akış

f_e: e kenarındaki toplam akış

$\ell_P(f) = \Sigma_{e \in P} \ell_e(f_e)$

Toplam gecikme

$C(f) = \Sigma_e f_e \cdot \ell_e(f_e)$

Bir yolun gecikmesi üzerindeki kenarların gecikmeleri toplamıdır; toplam gecikme, her kenarda akış × gecikmedir.

BENCİL YÖNLENDİRME

Wardrop İlkeleri (1952)

1. ilke: kullanıcı dengesi

+ Kullanılan tüm yolların gecikmesi eşit ve en düşüktür.
+ Kullanılmayan bir yol daha yavaştır (ya da eşit).
+ Tek bir kullanıcı yol değiştirerek yolculuk süresini azaltamaz.

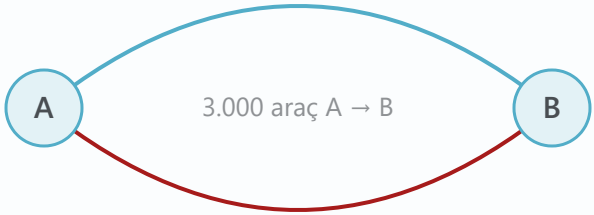
2. ilke: sistem optimumu

– Ortalama (toplam) gecikme en küçüktür.
– Merkezî bir planlayıcının seçeceği dağılım.
– Bazı kullanıcılar daha yavaş yola "feda edilebilir".

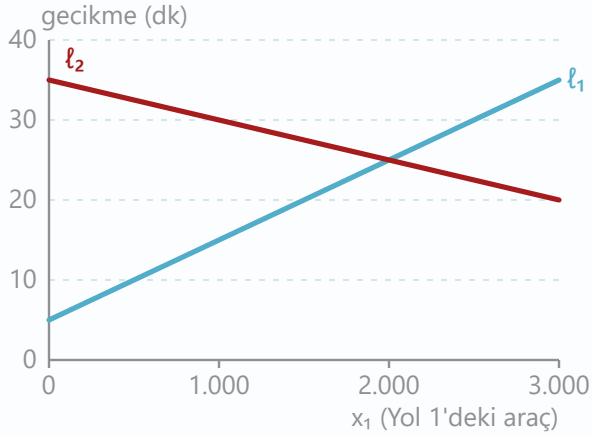
Kısaca: kullanılan her yolun gecikmesi, herhangi bir yolun gecikmesinden büyük değildir. Denge kullanıcıların bireysel tercihlerini açıklar, ancak sistem optimumuyla aynı olmak zorunda değildir. Doğrusal gecikmelerde denge vardır ve her kenarın denge gecikmesi, hangi denge bulunursa bulunsun aynıdır.

Adım Adım: İki Yollu Ağda Wardrop Dengesi

Yol 1: $\ell_1 = 5 + x_1/100$



Yol 2: $\ell_2 = 20 + x_2/200$

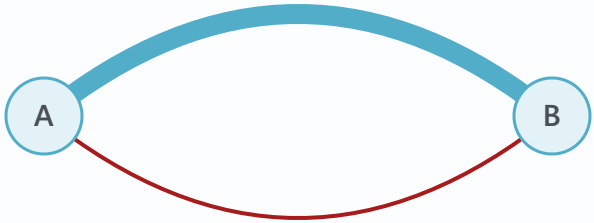


Kurulum: 3.000 araç A'dan B'ye gidiyor. Yol 1 kısa ama dar (gecikme hızla artar), Yol 2 uzun ama geniş.

Adım Adım: İki Yollu Ağda Wardrop Dengesi

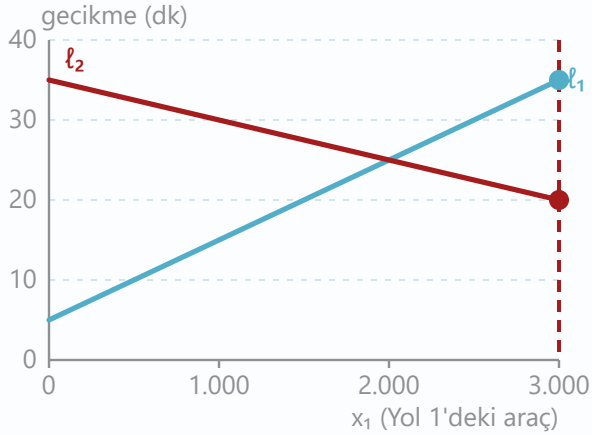
Yol 1: $\ell_1 = 5 + x_1/100$

$x_1 = 3.000$ araç → 35 dk



$x_2 = 0$ araç → 20 dk

Yol 2: $\ell_2 = 20 + x_2/200$

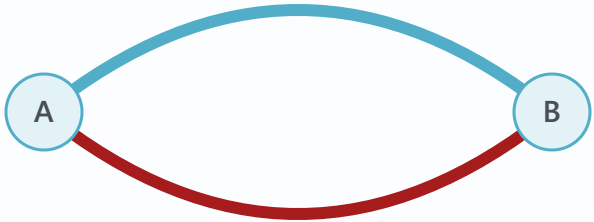


Deneme 1: herkes Yol 1'de → $\ell_1 = 35$ dk, boş Yol 2 ise 20 dk. Yol 1'dekiler **geçmek ister**: denge değil.

Adım Adım: İki Yollu Ağda Wardrop Dengesi

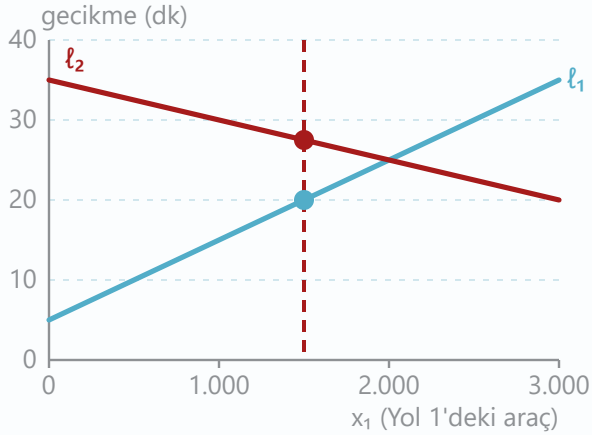
Yol 1: $\ell_1 = 5 + x_1/100$

x_1 = 1.500 araç → 20 dk



x_2 = 1.500 araç → 27,5 dk

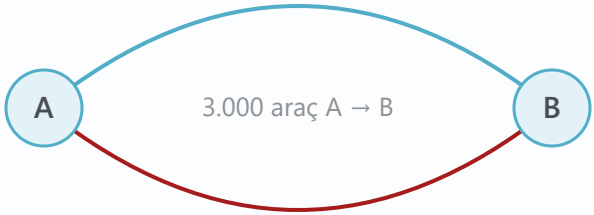
Yol 2: $\ell_2 = 20 + x_2/200$



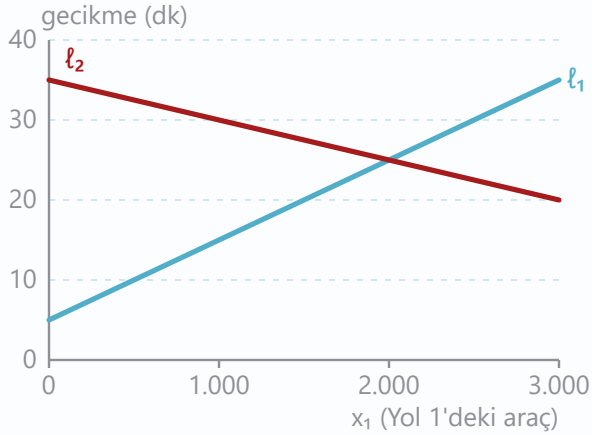
Deneme 2: 1.500 / 1.500 → $\ell_1 = 20$, $\ell_2 = 27{,}5$ dk. Bu kez Yol 2'dekiler Yol 1'e kayar: yine denge değil.

Adım Adım: İki Yollu Ağda Wardrop Dengesi

Yol 1: $\ell_1 = 5 + x_1/100$



Yol 2: $\ell_2 = 20 + x_2/200$

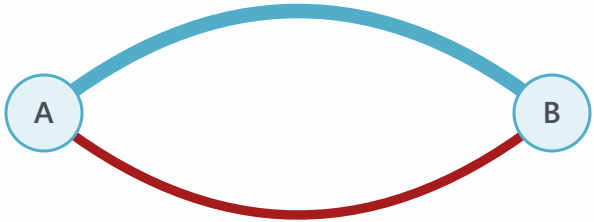


Denge koşulu: iki yol da kullanılıyorsa gecikmeler eşit olmalı: $5 + x_1/100 = 20 + (3000 - x_1)/200$.

Adım Adım: İki Yollu Ağda Wardrop Dengesi

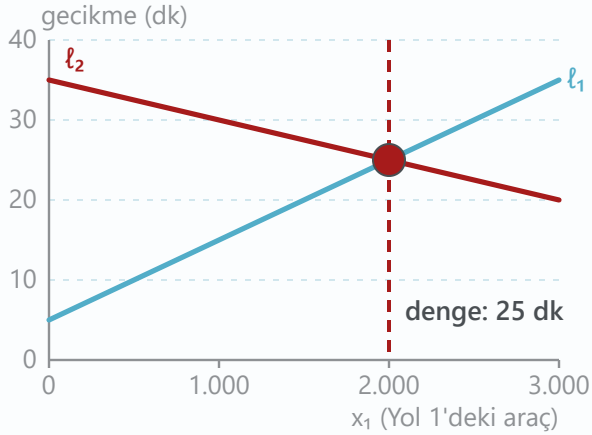
Yol 1: $\ell_1 = 5 + x_1/100$

x_1 = 2.000 araç → 25 dk



x_2 = 1.000 araç → 25 dk

Yol 2: $\ell_2 = 20 + x_2/200$

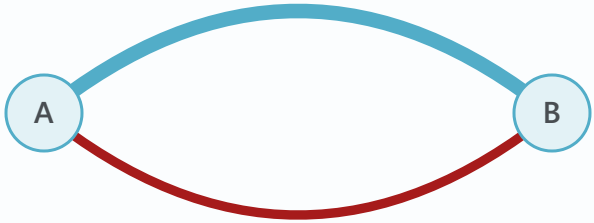


×200: $1000 + 2x_1 = 4000 + 3000 - x_1 \rightarrow 3x_1 = 6000 \rightarrow$ **$x_1 = 2.000$, $x_2 = 1.000$**; ikisi de 25 dk.

Adım Adım: İki Yollu Ağda Wardrop Dengesi

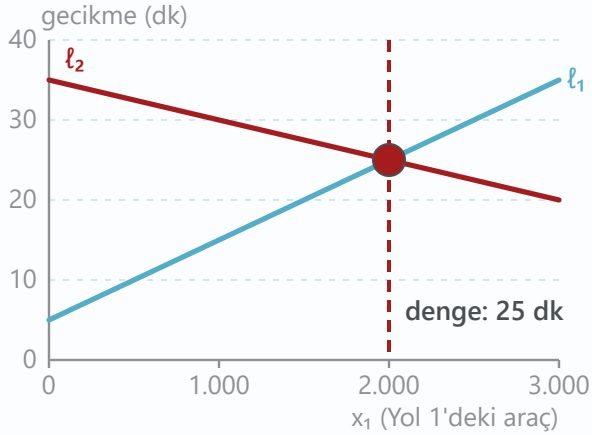
Yol 1: $\ell_1 = 5 + x_1/100$

x_1 = 2.000 araç → 25 dk



x_2 = 1.000 araç → 25 dk

Yol 2: $\ell_2 = 20 + x_2/200$



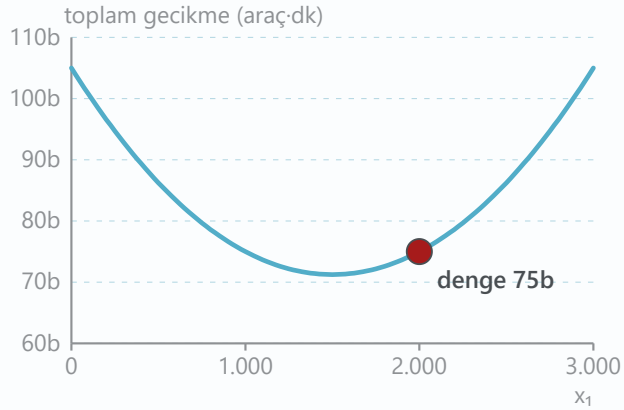
Kontrol: yol değiştiren tek sürücü yine ≈ 25 dk yaşar → kimse kazanmaz. Toplam gecikme 3.000 × 25 = **75.000 araç·dk**.

BENCİL YÖNLENDİRME

Denge ile Sistem Optimumu Aynı Değil

Toplam gecikme

$C = x_1 \cdot \ell_1(x_1) + x_2 \cdot \ell_2(x_2)$



Denge x_1 = 2.000'de toplam 75.000 araç·dk. Bu, eğrinin en alt noktası mı?

Denge ile Sistem Optimumu Aynı Değil

Toplam gecikme

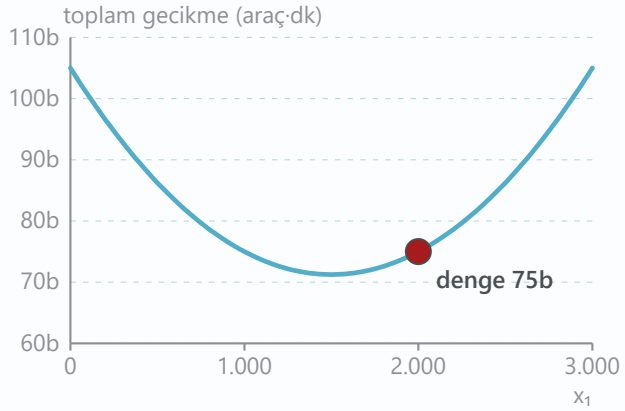
$C = x_1 \cdot \ell_1(x_1) + x_2 \cdot \ell_2(x_2)$

Marjinal maliyet

: bir araç daha eklemenin toplam gecikmeye kattığı miktar

Yol 1: $x_1 \cdot \ell_1 = 5x_1 + x_1^2/100$ → türev $5 + x_1/50$

Yol 2: $x_2 \cdot \ell_2 = 20x_2 + x_2^2/200$ → türev $20 + x_2/100$



Bir araç daha eklemek hem kendi gecikmesini hem de *diğerlerine* yüklediği $x \cdot \ell'$ gecikmesini getirir.

Denge ile Sistem Optimumu Aynı Değil

Toplam gecikme

$C = x_1 \cdot \ell_1(x_1) + x_2 \cdot \ell_2(x_2)$

Marjinal maliyet

: bir araç daha eklemenin toplam gecikmeye kattığı miktar

Yol 1: $x_1 \cdot \ell_1 = 5x_1 + x_1^2/100$ → türev $5 + x_1/50$

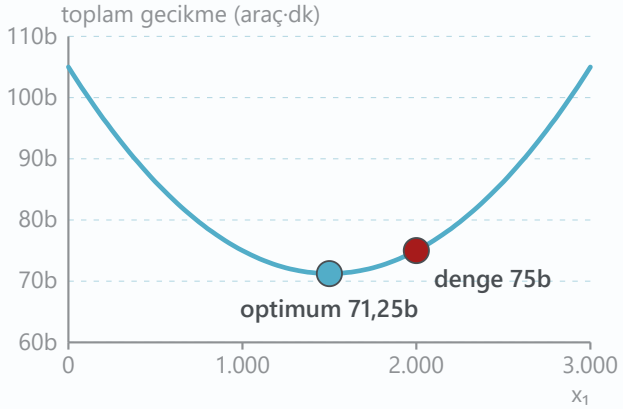
Yol 2: $x_2 \cdot \ell_2 = 20x_2 + x_2^2/200$ → türev $20 + x_2/100$

Eşitle:

$5 + x_1/50 = 20 + (3000 - x_1)/100$

→ $3x_1 = 4500$ →

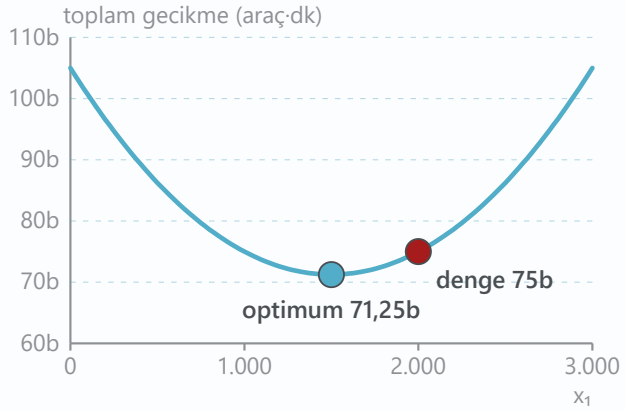
$x_1 = x_2 = 1.500$



Optimumda marjinal maliyetler eşitlenir: trafik 1.500 / 1.500 bölünür, toplam 71.250.

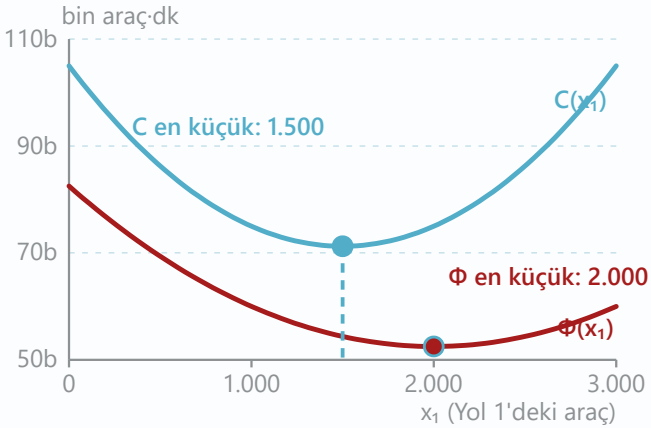
Denge ile Sistem Optimumu Aynı Değil

	x_1 / x_2	ℓ_1 / ℓ_2 (dk)	Toplam
Denge	2.000 / 1.000	25 / 25	75.000
Optimum	1.500 / 1.500	20 / 27,5	71.250



Optimumda Yol 1 sürücüsü 20, Yol 2 sürücüsü 27,5 dk yaşar; Yol 2'dekiler kaçmak ister → **optimum denge değildir**.

Denge ve Optimum: İki Ayrı En Aza İndirme



Wardrop dengesi

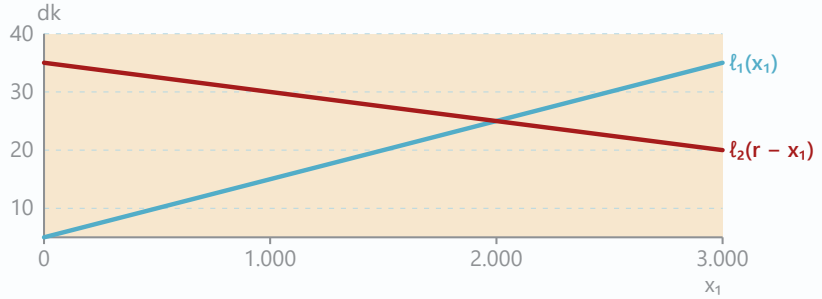
Beckmann potansiyelini (Beckmann, McGuire, Winsten, 1956) en aza indirir:
$\Phi(f) = \Sigma_e \int_0^{f_e} \ell_e(t)\,dt$
Türev ℓ_e: "son gelen sürücünün" gördüğü gecikme.

Sistem optimumu

Toplam gecikmeyi en aza indirir:
$C(f) = \Sigma_e f_e \cdot \ell_e(f_e)$
Türev $\ell_e + f_e \cdot \ell'_e$: marjinal maliyet.

İki eğrinin dipleri farklı yerde: Φ x_1 = 2.000'de (denge, Φ = 52.500), C x_1 = 1.500'de (optimum, C = 71.250). Fark tek terimden gelir: sürücü, kendi varlığının başkalarına yüklediği **x·ℓ′** dışsallığını hesaba katmaz.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 0

hi = 3.000

genişlik = 3.000

Çağrı: iki yollu ağ ($\ell_1 = 5 + x/100$, $\ell_2 = 20 + x/200$), 3.000 araç. Aranan: $\ell_1(x) = \ell_2(r - x)$ olan x.

Adım 1 / 9

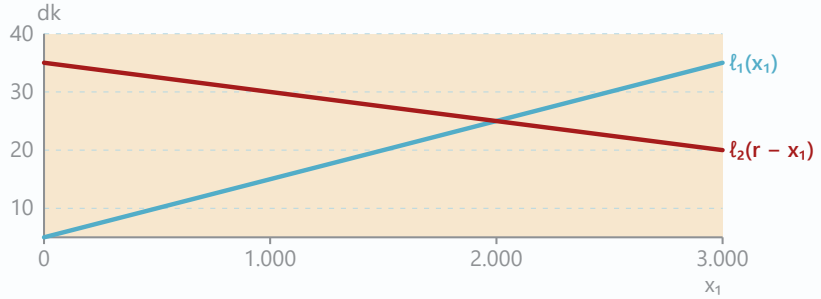
```
▶ def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
4     x = (lo + hi) / 2
5     if l1(x) < l2(r - x):
6       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
8       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 0
hi = 3.000
genişlik = 3.000

Arama aralığı [0, 3.000]: denge bu aralıkta olmak zorunda (x = 0'da Yol 1 hızlı, x = r'de yavaş).

Adım 2 / 9

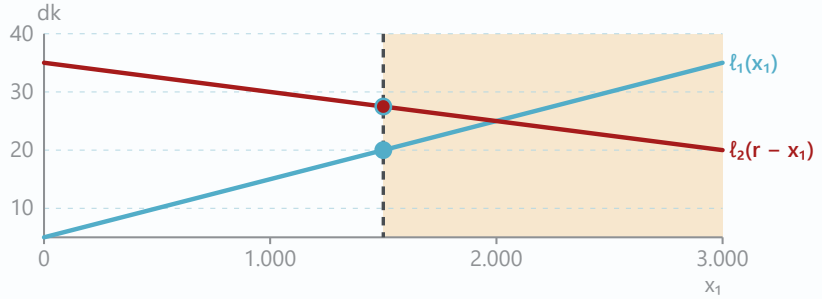
```
1 def denge_bul(l1, l2, r, eps=50):
▶   lo, hi = 0, r
3   while hi - lo > eps:
4     x = (lo + hi) / 2
5     if l1(x) < l2(r - x):
6       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
8       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 0` `hi = 3000`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.500
hi = 3.000
genişlik = 1.500
x = 1.500
$\ell_1 = 20 \cdot \ell_2 = 27{,}5$

Tur 1: x = 1.500 → $\ell_1 = 20 < \ell_2 = 27{,}5$. Yol 1 hâlâ hızlı, daha çok araç ona geçer: **lo = 1.500**.

Adım 3 / 9

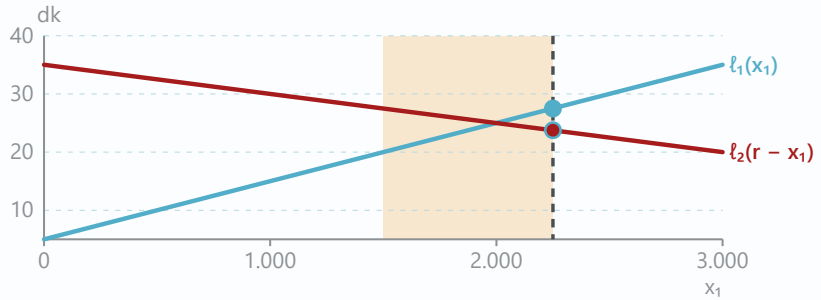
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
▶     x = (lo + hi) / 2
▶     if l1(x) < l2(r - x):
▶       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
8       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1500` `hi = 3000` `x = 1500`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.500
hi = 2.250
genişlik = 750
x = 2.250
$\ell_1 = 27{,}5 \cdot \ell_2 = 23{,}75$

Tur 2: x = 2.250 → $\ell_1 = 27{,}5 \geq \ell_2 = 23{,}75$. Yol 1 fazla kalabalık: **hi = 2.250**.

Adım 4 / 9

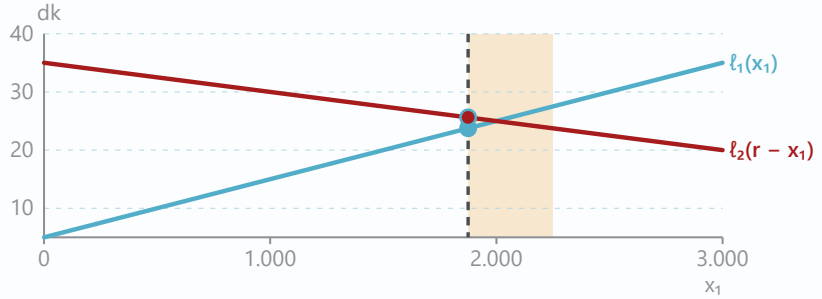
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
▶     x = (lo + hi) / 2
▶     if l1(x) < l2(r - x):
6       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
▶       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1500` `hi = 2250` `x = 2250`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.875
hi = 2.250
genişlik = 375
x = 1.875
ℓ_1 = 23,75 · ℓ_2 = 25,63

Tur 3: x = 1.875 → ℓ_1 = 23,75 < ℓ_2 = 25,63. Yol 1 hâlâ hızlı, daha çok araç ona geçer: **lo = 1.875**.

Adım 5 / 9

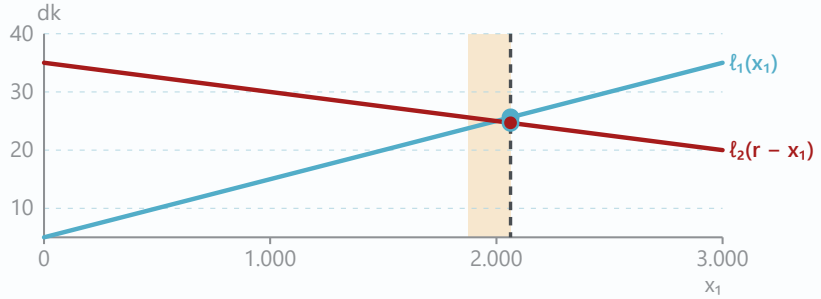
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
▶     x = (lo + hi) / 2
▶     if l1(x) < l2(r - x):
▶       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
8       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1875` `hi = 2250` `x = 1875`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.875
hi = 2.062,5
genişlik = 187,5
x = 2.062,5
ℓ_1 = 25,63 · ℓ_2 = 24,69

Tur 4: x = 2.062,5 → ℓ_1 = 25,63 ≥ ℓ_2 = 24,69. Yol 1 fazla kalabalık: **hi = 2.062,5**.

Adım 6 / 9

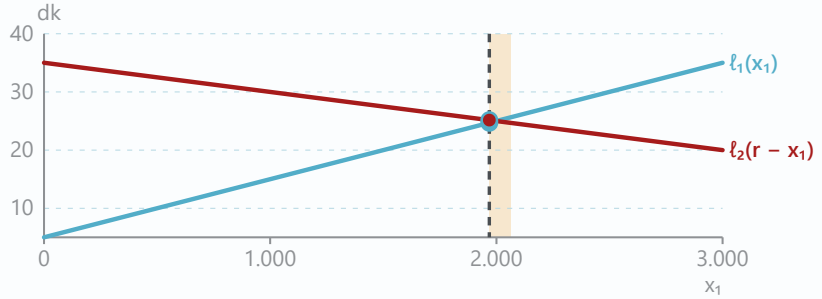
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
▶     x = (lo + hi) / 2
▶     if l1(x) < l2(r - x):
6       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
▶       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1875` `hi = 2062.5` `x = 2062.5`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.968,75
hi = 2.062,5
genişlik = 93,75
x = 1.968,75
ℓ_1 = 24,69 · ℓ_2 = 25,16

Tur 5: x = 1.968,75 → ℓ_1 = 24,69 < ℓ_2 = 25,16. Yol 1 hâlâ hızlı, daha çok araç ona geçer: **lo = 1.968,75**.

Adım 7 / 9

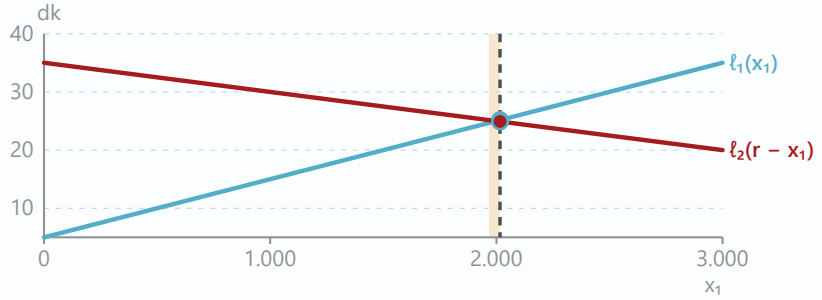
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
▶     x = (lo + hi) / 2
▶     if l1(x) < l2(r - x):
▶       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
8       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1968.75` `hi = 2062.5` `x = 1968.75`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.968,75
hi = 2.015,63
genişlik = 46,88
x = 2.015,63
ℓ_1 = 25,16 · ℓ_2 = 24,92

Tur 6: x = 2.015,63 → ℓ_1 = 25,16 ≥ ℓ_2 = 24,92. Yol 1 fazla kalabalık: **hi = 2.015,63**.

Adım 8 / 9

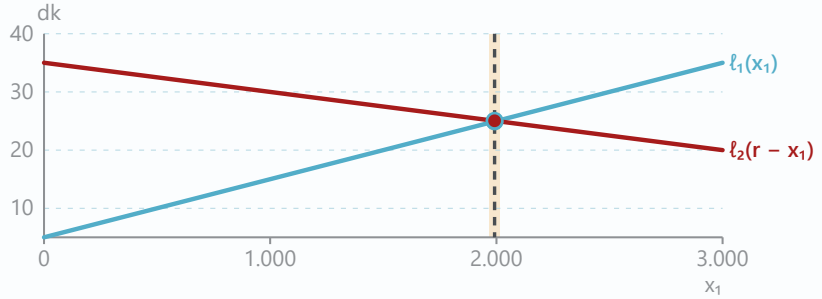
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
▶     x = (lo + hi) / 2
▶     if l1(x) < l2(r - x):
6       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
▶       hi = x      # Yol 1 yavaş: azalt
9   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1968.75` `hi = 2015.625` `x = 2015.625`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Kod İzleme: `denge_bul(l1, l2, r)`



lo = 1.968,75
hi = 2.015,63
genişlik = 46,88
x = 1.992,19
ℓ_1 = 24,92 · ℓ_2 = 25,04

Genişlik 46,88 ≤ 50 → dur. Sonuç ≈ **1.992,19** (gerçek denge 2.000). Her tur aralığı yarıya indirir: k tur sonra hata ≤ $r/2^k$.

Adım 9 / 9

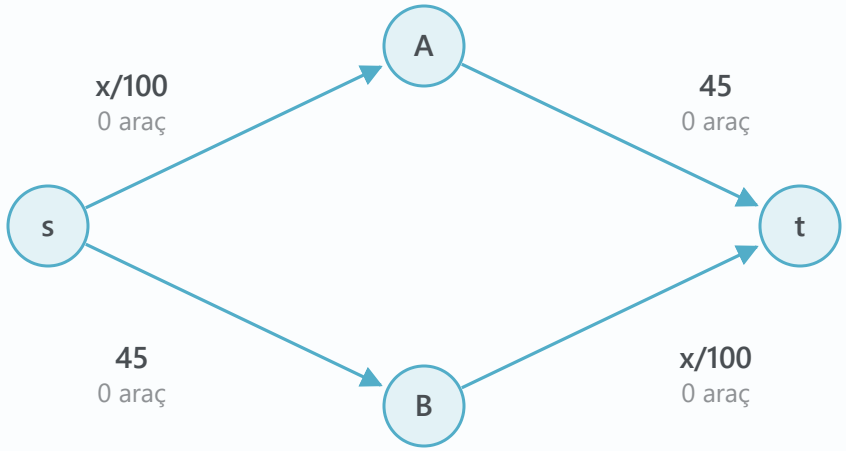
```
1 def denge_bul(l1, l2, r, eps=50):
2   lo, hi = 0, r
3   while hi - lo > eps:
4     x = (lo + hi) / 2
5     if l1(x) < l2(r - x):
6       lo = x      # Yol 1 hâlâ hızlı: artır
7     else:
8       hi = x      # Yol 1 yavaş: azalt
▶   return (lo + hi) / 2
```

DEĞİŞKENLER

`r = 3000` `eps = 50` `lo = 1968.75` `hi = 2015.625`

$\ell_1(x) - \ell_2(r - x)$ artan olduğundan tek sıfırı dengedir. Genel ağlarda Beckmann potansiyeli dışbükey optimizasyonla (ör. Frank–Wolfe) en aza indirilir.

Adım Adım: Braess Paradoksu (4.000 Araç)



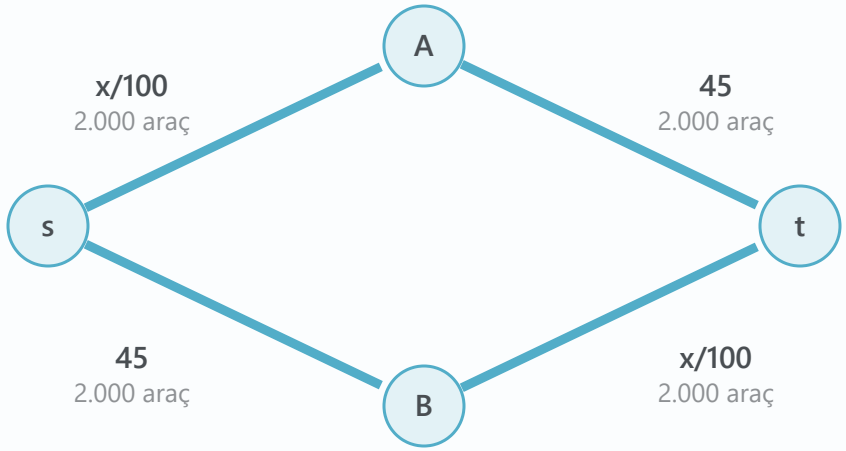
Yol	araç	süre
s→A→t	0	—
s→B→t	0	—
s→A→B→t	—	—

4.000 araç s → t

x: kenardaki araç sayısı

Kurulum: 4.000 araç s'den t'ye. s→A ve B→t dar yollar (x/100 dk), A→t ve s→B geniş otoyol (sabit 45 dk).

Adım Adım: Braess Paradoksu (4.000 Araç)



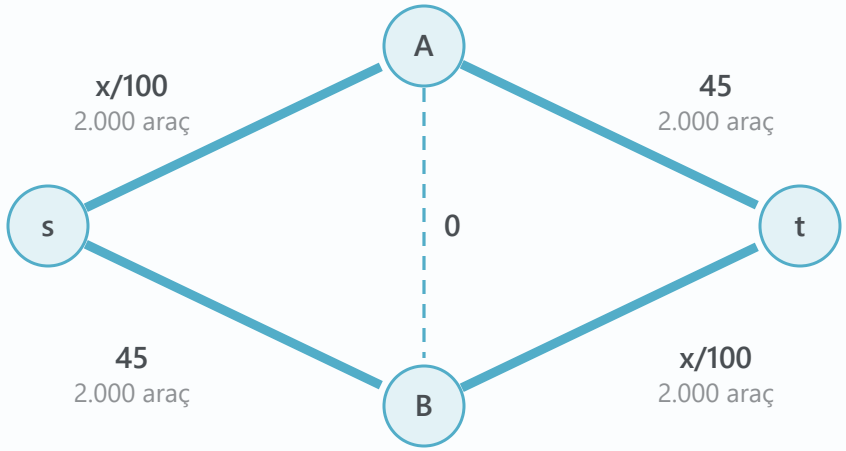
Yol	araç	süre
s→A→t	2.000	65 dk
s→B→t	2.000	65 dk
s→A→B→t	—	—

Ortalama: 65 dk

Toplam: 260.000 araç·dk

Simetri: 2.000 / 2.000 → her iki yol 2000/100 + 45 = **65 dk**. Geçen kazanmaz: Wardrop dengesi (ve optimum).

Adım Adım: Braess Paradoksu (4.000 Araç)



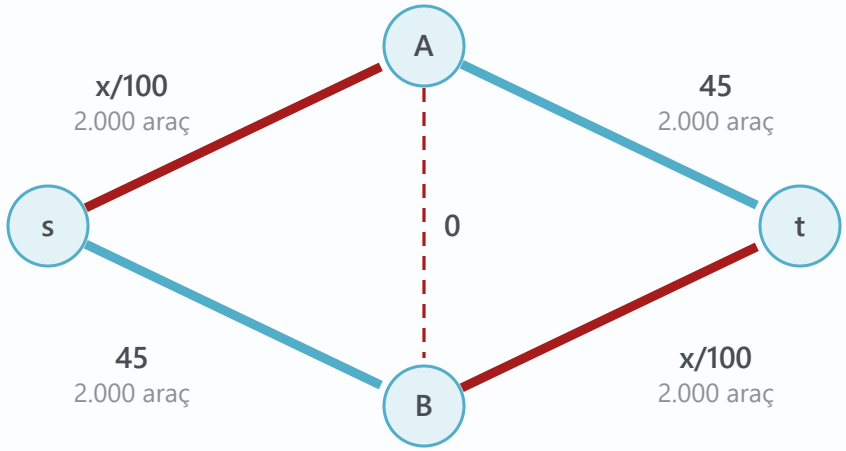
Yol	araç	süre
s→A→t	2.000	65 dk
s→B→t	2.000	65 dk
s→A→B→t	0	—

Ortalama: 65 dk

Toplam: 260.000 araç·dk

Belediye A'dan B'ye çok kısa, **0 dk**'lık bir bağlantı yolu açar. Başlangıçta kimse kullanmıyor.

Adım Adım: Braess Paradoksu (4.000 Araç)



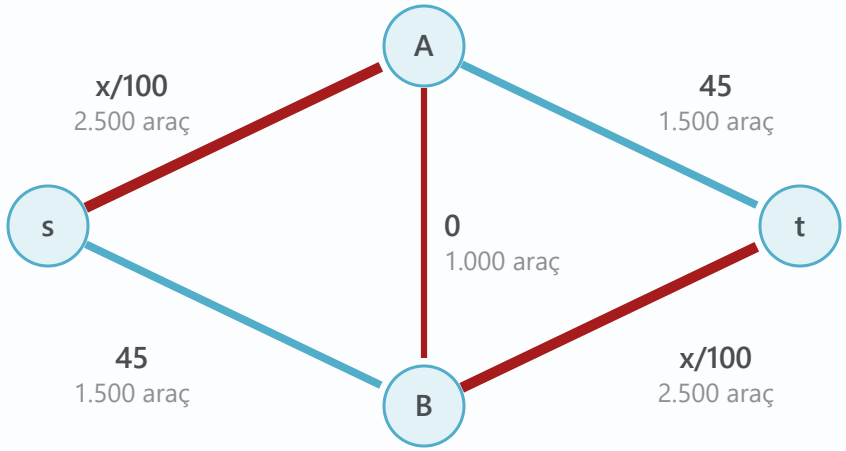
Yol	araç	süre
s→A→t	2.000	65 dk
s→B→t	2.000	65 dk
s→A→B→t	0	40 dk

Zikzak: 20 + 0 + 20

= 40 dk < 65 dk

Tek bir sürücü s→A→B→t dener: 20 + 0 + 20 = **40 dk** < 65. Bireysel olarak çok kârlı → geçer.

Adım Adım: Braess Paradoksu (4.000 Araç)



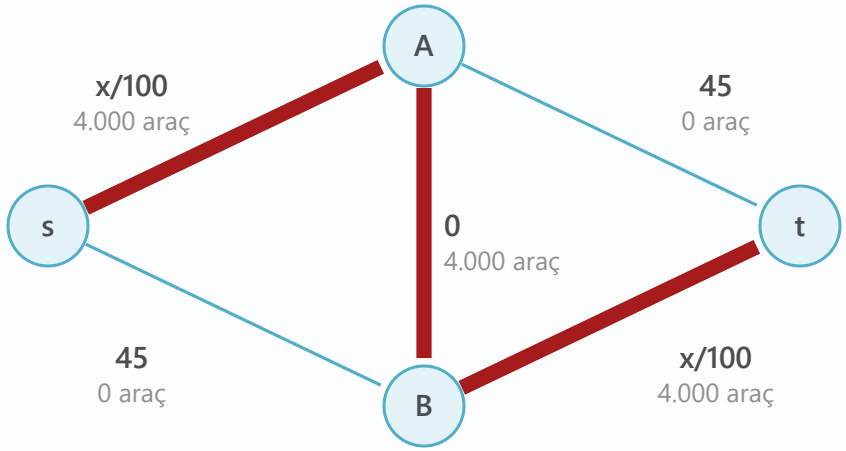
Yol	araç	süre
s→A→t	1.500	70 dk
s→B→t	1.500	70 dk
s→A→B→t	1.000	50 dk

Zikzak hâlâ en hızlı

50 dk < 70 dk

1.000 sürücü geçti: dar yollarda 2.500 araç → 25 dk. Eski yollar 70, zikzak 50 dk: **geçiş sürüyor**.

Adım Adım: Braess Paradoksu (4.000 Araç)



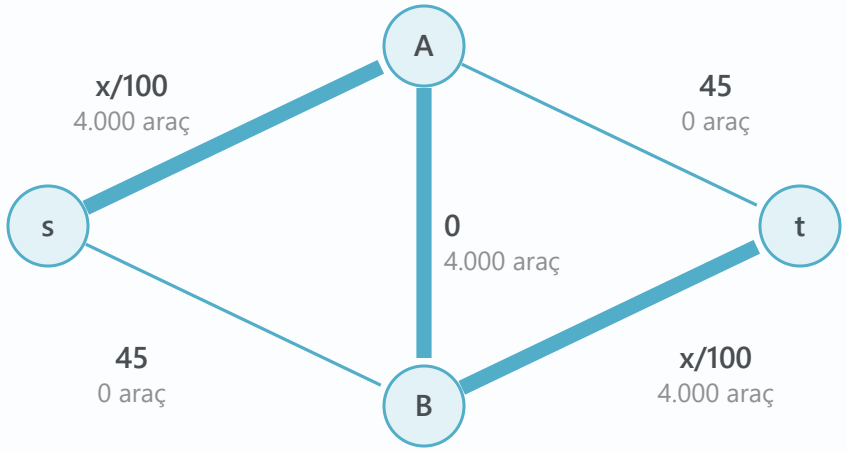
Yol	araç	süre
s→A→t	0	85 dk
s→B→t	0	85 dk
s→A→B→t	4.000	80 dk

Ortalama: 80 dk

Toplam: 320.000 araç·dk

Herkes zikzakta: 40 + 0 + 40 = **80 dk**. Eski yollara dönen 40 + 45 = 85 dk yaşar → kimse dönmez: yeni denge.

Adım Adım: Braess Paradoksu (4.000 Araç)



Yol	araç	süre
s→A→t	0	85 dk
s→B→t	0	85 dk
s→A→B→t	4.000	80 dk

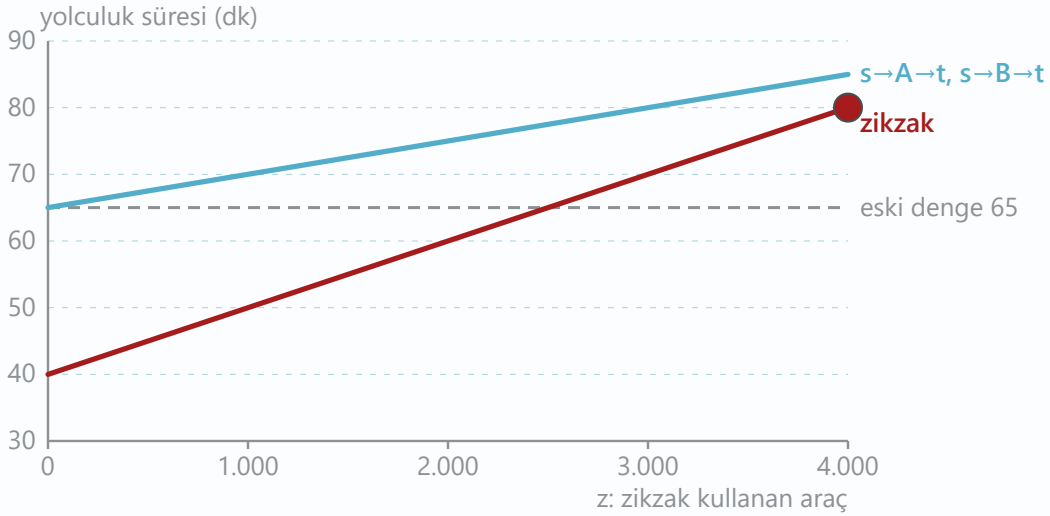
65 → 80 dk

260 bin → 320 bin araç·dk

Sonuç: yol eklendi, **herkes 15 dk kaybetti**. Kapasite artışı + bencil seçim = daha kötü denge. Kestirmeli ağın sistem optimumu ise ≈ 64,7 dk'dır (zikzakta yalnız 500 araç).

BRAESS PARADOKSU

Neden Herkes Zikzağa Geçer?



Kalan trafik iki eski yola eşit bölünürse eski yollar 65 + z/200, zikzak 40 + z/100 dk. Zikzak **her z için** daha hızlıdır (B→t en çok 40 dk < 45), yani herkes için baskın seçimdir. Ama her geçen sürücü dar yollara yük bindirir: z = 4.000'de herkes 80 dk.

BRAESS PARADOKSU

Braess Paradoksu: Ders ve Örnekler

- Bir ağa yeni bağlantı eklemek, bencil rota seçimi nedeniyle **tüm** kullanıcıların gecikmesini artırabilir (Braess, 1968).
- Yeni yol bireysel olarak çekici görünür, ancak kullanıcıların birlikte oluşturduğu trafik deseni daha kötü bir dengeye taşınır.
- Tersine: bir yolu *kapatmak* bazen herkesi hızlandırabilir.

Stuttgart (1969)

Yeni yol yatırımı sonrası bozulan trafiğin, bir cadde kapatılınca düzeldiği sıkça anılır (Knödel, 1969).

New York (1990)

Dünya Günü'nde 42. Cadde kapatıldığında beklenen tıkanıklık oluşmadığı bildirilmiştir (Kolata, 1990).

Mekanik / elektrik

Yay ve direnç ağlarında da benzer paradoks gösterilmiştir (Cohen ve Horowitz, 1991).

Ücretlendirme, erişim kısıtı veya merkezî yönlendirme bu etkiyi azaltabilir.

Oyuna Dönüş: Braess Paradoksu

Kuramın cevabı: hayır

Ortalama süre	Bencil denge	Optimum
Kestirme yok	20 + 45 = 65	65
Kestirme var	**40 + 0 + 40 = 80**	≈ 64,7

Kestirmeyle herkes S→A→B→T'yi seçer. Eski yola dönen 40 + 45 = 85 bekler, bu yüzden kimse dönmez: denge 80. Kestirmeli ağın optimumu zikzağa küçük bir pay verir (1.750 / 1.750 / 500 araç → 258.750 / 4.000 ≈ 64,7).

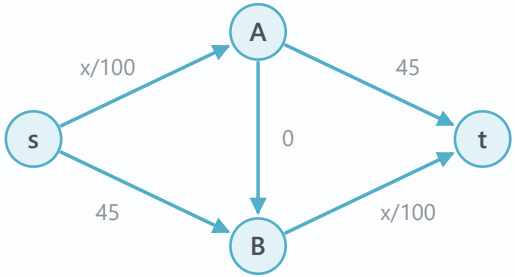
Gerçekte

Laboratuvar deneylerinde de kestirme eklenince denekler zamanla paradoksun öngördüğü daha yavaş dengeye yöneldi (Rapoport vd., 2009).

Kentlerde: Stuttgart ve New York örneklerine ek olarak Seul'de Cheonggyecheon üzerindeki yükseltilmiş otoyolun 2000'lerin başında kaldırılmasından sonra şehir trafiğinin beklenen ölçüde kötüleşmediği sıkça anılır. Gerçek ağlar çok daha karmaşıktır; bu örnekler kesin kanıt değil, paradoksun olası olduğunun göstergesidir.

Yol yapmak ya da kapatmak, sürücülerin varacağı dengeye göre değerlendirilmelidir; tek başına kapasiteye göre değil.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	
s→A	
A→t	
s→B	
B→t	
A→B	

Çağrı: Braess ağı, 5 kenar, 4.000 araç. Soru: hangi kenar *kapatılırsa* denge süresi en çok düşer?

Adım 1 / 13

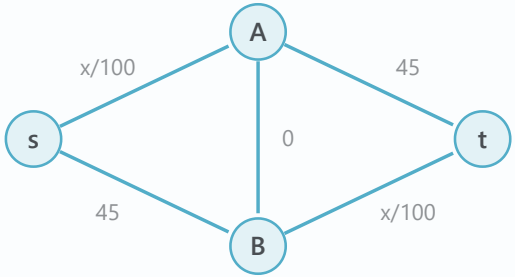
```
▶ def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	
A→t	
s→B	
B→t	
A→B	

eniyi
80 dk
sil = None

Tüm ağın dengesi: herkes zikzakta, **80 dk**. Hiçbir şey silmemek başlangıç adayı.

Adım 2 / 13

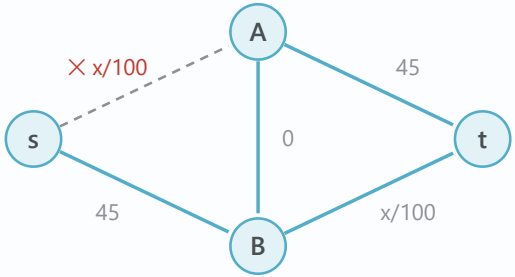
```
1 def en_iyi_alt_ag(ag, r):
▶   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	
s→B	
B→t	
A→B	

eniyi
80 dk
sil = None

s→A kapatılır: yalnız s→B→t kalır: 40 + 45 = 85 dk.

Adım 3 / 13

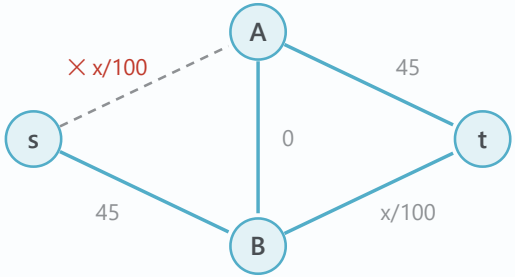
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
▶   for e in ag:
▶     alt = [k for k in ag if k != e]
▶     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = sA` `d = 85`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	
s→B	
B→t	
A→B	

eniyi
80 dk
sil = None

85 < 80 değil → aday değişmez.

Adım 4 / 13

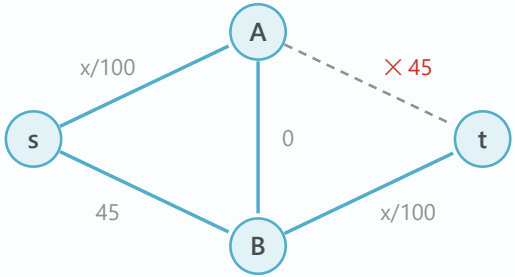
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
▶     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = sA` `d = 85`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	
B→t	
A→B	

eniyi
80 dk
sil = None

A→t kapatılır: zikzak ve alt yol kalır; zikzak baskın: 80 dk.

Adım 5 / 13

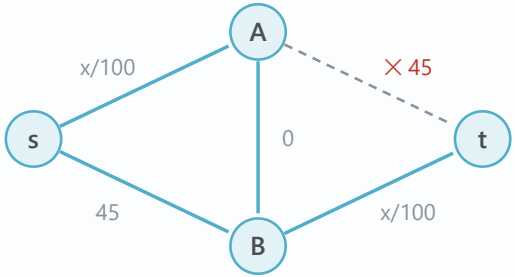
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
▶   for e in ag:
▶     alt = [k for k in ag if k != e]
▶     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = At` `d = 80`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	
B→t	
A→B	

eniyi
80 dk
sil = None

80 < 80 değil → aday değişmez.

Adım 6 / 13

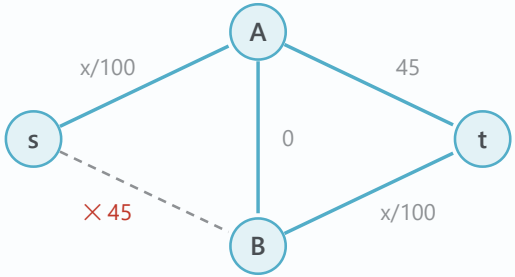
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
▶     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = At` `d = 80`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	
A→B	

eniyi
80 dk
sil = None

s→B kapatılır: üst yol ve zikzak kalır; zikzak baskın: 80 dk.

Adım 7 / 13

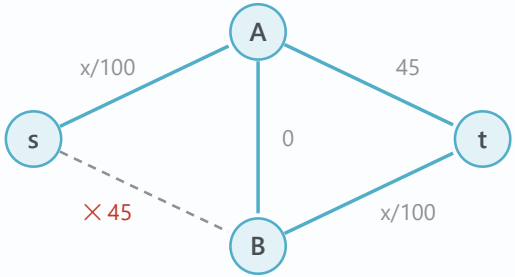
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
▶   for e in ag:
▶     alt = [k for k in ag if k != e]
▶     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = sB` `d = 80`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	
A→B	

eniyi
80 dk
sil = None

80 < 80 değil → aday değişmez.

Adım 8 / 13

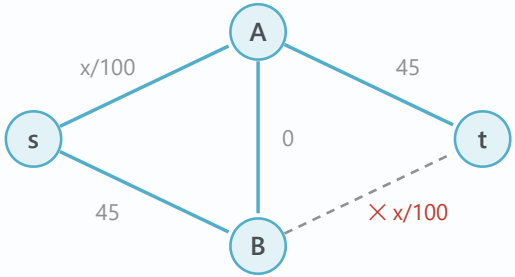
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
▶     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = sB` `d = 80`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	85 dk
A→B	

eniyi
80 dk
sil = None

B→t kapatılır: yalnız s→A→t kalır: 40 + 45 = 85 dk.

Adım 9 / 13

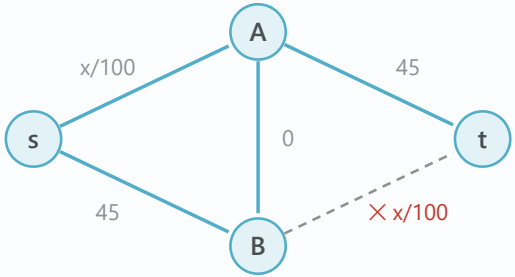
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
▶   for e in ag:
▶     alt = [k for k in ag if k != e]
▶     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = Bt` `d = 85`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	85 dk
A→B	

eniyi
80 dk
sil = None

85 < 80 değil → aday değişmez.

Adım 10 / 13

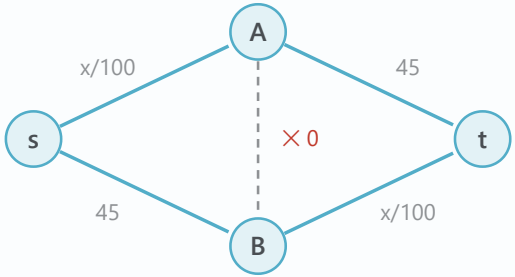
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
▶     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = Bt` `d = 85`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	85 dk
A→B	65 dk

eniyi
80 dk
sil = None

A→B kapatılır: Braess öncesi ağ: 2.000 / 2.000, 20 + 45 = 65 dk.

Adım 11 / 13

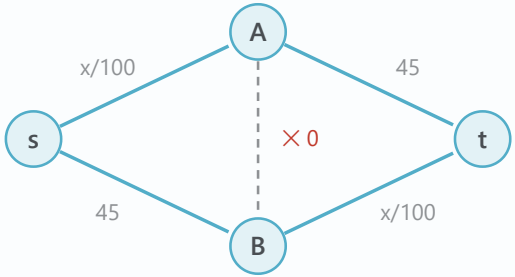
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
▶   for e in ag:
▶     alt = [k for k in ag if k != e]
▶     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 80` `sil = None` `e = AB` `d = 65`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	85 dk
A→B	65 dk

eniyi
65 dk
sil = A→B

65 < önceki en iyi → **güncelle**: yeni en iyi alt ağ "A→B kapalı".

Adım 12 / 13

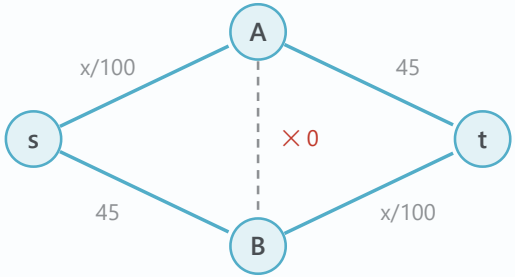
```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
▶     if d < eniyi:
▶       eniyi, sil = d, e
8   return sil, eniyi
```

DEĞİŞKENLER

`r = 4000` `eniyi = 65` `sil = AB` `e = AB` `d = 65`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

Kod İzleme: `en_iyi_alt_ag(ag, r)`



silinen	denge süresi
—	80 dk
s→A	85 dk
A→t	80 dk
s→B	80 dk
B→t	85 dk
A→B	65 dk

eniyi
65 dk
sil = A→B

Sonuç: **A→B kapatılmalı**, denge 80 → 65 dk. Tasarımcı “hangi kenarları açık tutayım?” diye sorar; genel ağda bu seçim zordur.

Adım 13 / 13

```
1 def en_iyi_alt_ag(ag, r):
2   eniyi, sil = denge_suresi(ag, r), None
3   for e in ag:
4     alt = [k for k in ag if k != e]
5     d = denge_suresi(alt, r)   # yol yoksa inf
6     if d < eniyi:
7       eniyi, sil = d, e
▶   return sil, eniyi
```

DEĞİŞKENLER

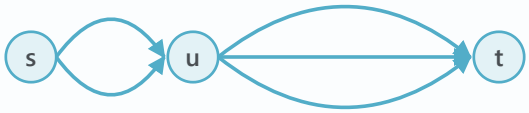
`r = 4000` `eniyi = 65` `sil = AB`

denge_suresi(ag, r): alt ağdaki Wardrop dengesinde kullanılan yolların ortak süresi (betikte Beckmann potansiyeli en aza indirilerek hesaplandı). Kod yalnız tek kenar silmeyi dener; tam arama 2^m alt ağ ister.

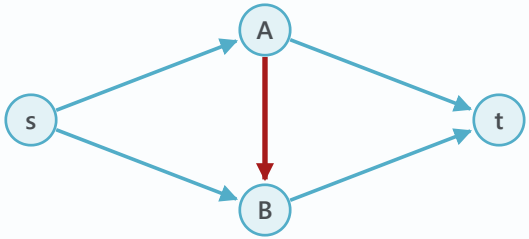
BRAESS PARADOKSU

Bencil Kullanıcılar İçin Ağ Tasarımı

Seri-paralel: paradoks yok



seri ve paralel bağlama ile kurulur



Wheatstone (Braess) alt yapısı

Problem

Verilen ağın hangi alt ağı açık kalırsa Wardrop dengesindeki süre en küçük olur? Yeni yol yapmak yerine yol *kapatmak* da bir tasarım kararıdır.

İleri okuma: zorluk

Kapatılacak kenarları en iyi seçmek genel ağlarda zordur: doğrusal gecikmelerde hiçbir kenarı kapatmamak bile en iyi seçimden en çok 4/3 kat kötüdür ve bundan iyisini verimli biçimde garanti etmek NP-zordur (Roughgarden, 2006).

İleri okuma: topoloji

Paralel yollardan ve bunların ardışık bağlanmasından oluşan (seri-paralel) ağlarda Braess paradoksu hiç görülmez; diğer ağlarda uygun gecikmelerle görülebilir (Milchtaich, 2006).

Tıkanıklık Oyunları (Congestion Games)

- n oyuncu, kaynak kümesi E (kenarlar). Her oyuncunun stratejisi bir kaynak alt kümesidir (ör. bir s–t yolu).
- Kaynağın maliyeti $c_e(n_e)$ yalnızca onu kullanan oyuncu **sayısına** bağlıdır ve bu yükle artar.
- Oyuncunun maliyeti: seçtiği kaynakların maliyetleri toplamı.
- Oyuncular kendi gecikmelerini azaltmaya çalışırken toplam ağ maliyeti yükselebilir.

	Atomik	**Bölünebilir**
Oyuncu	sonlu, n kişi	sürekli akış
Etki	tek kişi yükü 1 artırır	tek kişi ihmal edilir
Denge	saf Nash	Wardrop
Potansiyel	Rosenthal (toplam)	Beckmann (integral)

Rosenthal (1973): sonlu her tıkanıklık oyununda bir potansiyel fonksiyonu vardır, bu yüzden saf Nash dengesi her zaman bulunur. Karışık stratejiye gerek kalmaz.

Rosenthal Potansiyel Fonksiyonu

$$\Phi(s) = \Sigma_e \left[c_e(1) + c_e(2) + \ldots + c_e(n_e) \right]$$

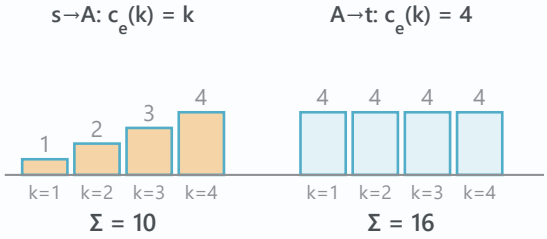
Her kenarda 1'den kullanıcı sayısı n_e'ye kadar maliyetleri topla (toplam maliyet değildir).

Rosenthal Potansiyel Fonksiyonu

$$\Phi(s) = \Sigma_e \left[c_e(1) + c_e(2) + \ldots + c_e(n_e) \right]$$

Her kenarda 1'den kullanıcı sayısı n_e'ye kadar maliyetleri topla (toplam maliyet değildir).

Örnek: 4 sürücü de s→A→t yolunda. s→A: $c_e(k) = k$ → 1 + 2 + 3 + 4 = 10; A→t: $c_e(k) = 4$ → 16; diğer kenarlar boş. **Φ = 26** (toplam maliyet 4 × 8 = 32).

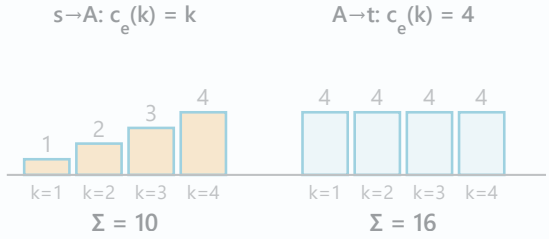


Rosenthal Potansiyel Fonksiyonu

$$\Phi(s) = \Sigma_e \left[c_e(1) + c_e(2) + \ldots + c_e(n_e) \right]$$

Her kenarda 1'den kullanıcı sayısı n_e'ye kadar maliyetleri topla (toplam maliyet değildir).

Örnek: 4 sürücü de s→A→t yolunda. s→A: $c_e(k) = k$ → 1 + 2 + 3 + 4 = 10; A→t: $c_e(k) = 4$ → 16; diğer kenarlar boş. **Φ = 26** (toplam maliyet 4 × 8 = 32).



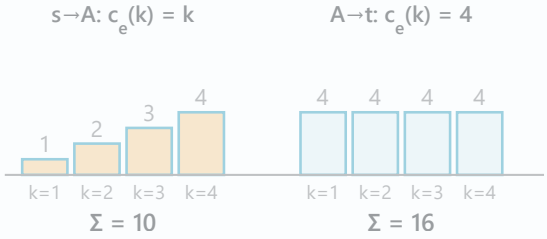
Anahtar özdeşlik. Sürücü i bir kenardan çıkınca o kenarın toplamındaki son terim $c_e(n_e)$ silinir (bu, onun o kenarda ödediği maliyettir); yeni bir kenara girince $c_e(n_e + 1)$ eklenir (onun yeni maliyeti). Bu yüzden Φ'deki değişim tam i'nin maliyet değişimidir: **$\Delta\Phi = c_i$(yeni) − c_i(eski)**.

Rosenthal Potansiyel Fonksiyonu

$$\Phi(s) = \Sigma_e \left[c_e(1) + c_e(2) + \ldots + c_e(n_e) \right]$$

Her kenarda 1'den kullanıcı sayısı n_e'ye kadar maliyetleri topla (toplam maliyet değildir).

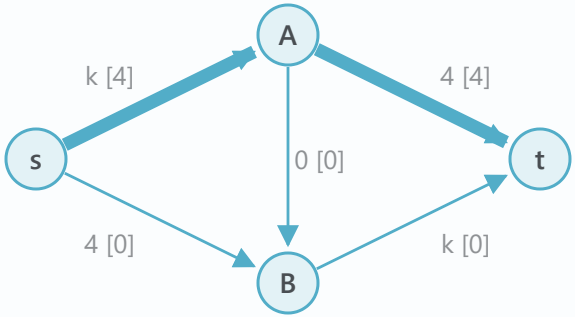
Örnek: 4 sürücü de s→A→t yolunda. s→A: $c_e(k) = k$ → 1 + 2 + 3 + 4 = 10; A→t: $c_e(k) = 4$ → 16; diğer kenarlar boş. **Φ = 26** (toplam maliyet 4 × 8 = 32).



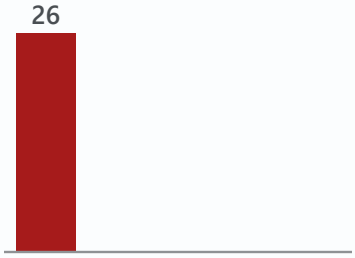
Anahtar özdeşlik. Sürücü i bir kenardan çıkınca o kenarın toplamındaki son terim $c_e(n_e)$ silinir (bu, onun o kenarda ödediği maliyettir); yeni bir kenara girince $c_e(n_e + 1)$ eklenir (onun yeni maliyeti). Bu yüzden Φ'deki değişim tam i'nin maliyet değişimidir: **$\Delta\Phi = c_i$(yeni) − c_i(eski)**.

Sonuç. İyileştiren her hamle Φ'yi kesin azaltır; Φ sonlu sayıda değer alır → dinamik durur. Durduğu yerde kimse iyileştiremez → **saf Nash dengesi vardır** (Rosenthal, 1973).

Adım Adım: En İyi Tepki Dinamiği ve Φ

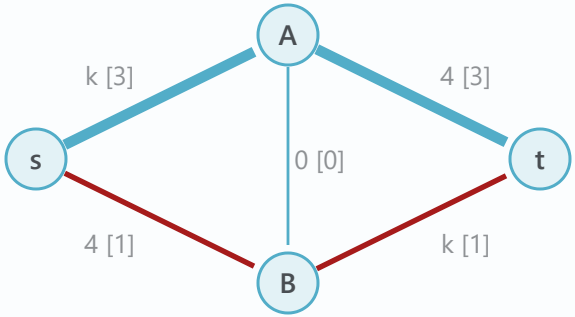


sürücü	yol	maliyet
1	üst	8
2	üst	8
3	üst	8
4	üst	8

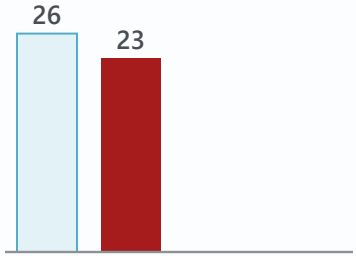


Kenar maliyetleri: s→A ve B→t'de k (kullanıcı sayısı), s→B ve A→t'de 4, A→B'de 0. Herkes s→A→t: maliyet 4 + 4 = 8, **Φ = 26**.

Adım Adım: En İyi Tepki Dinamiği ve Φ

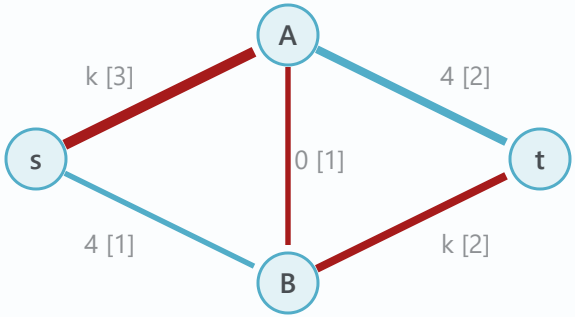


sürücü	yol	maliyet
1	alt	5
2	üst	7
3	üst	7
4	üst	7

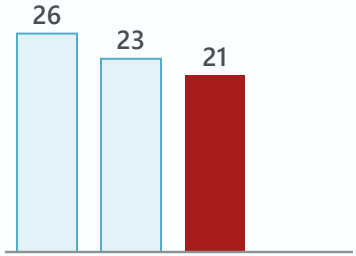


Sürücü 1'in seçenekleri: üst 8, alt 4 + 1 = 5, zik 4 + 0 + 1 = 5. Eşitlikte ilkini seçer → **alt** (8 → 5). Φ: 26 → 23 (Δ = −3).

Adım Adım: En İyi Tepki Dinamiği ve Φ

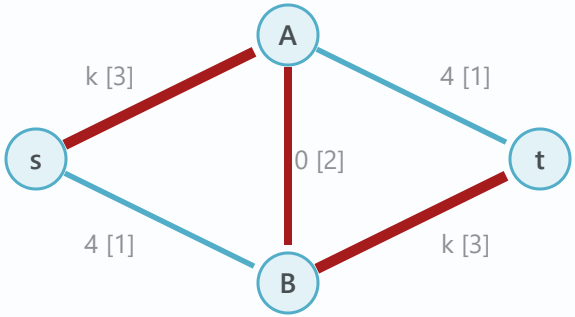


sürücü	yol	maliyet
1	alt	6
2	zik	5
3	üst	7
4	üst	7

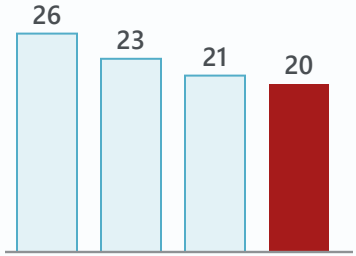


Sürücü 2: üst 7, alt 4 + 2 = 6, zik 3 + 0 + 2 = 5 → **zik** (7 → 5). Φ: 23 → 21 (Δ = −2).

Adım Adım: En İyi Tepki Dinamiği ve Φ

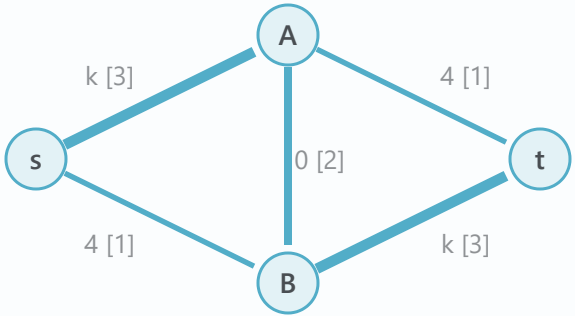


sürücü	yol	maliyet
1	alt	7
2	zik	6
3	zik	6
4	üst	7

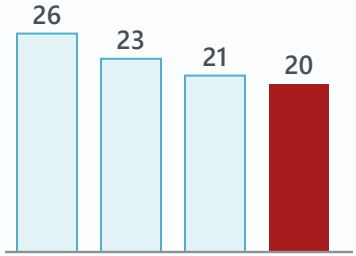


Sürücü 3: üst 7, alt 4 + 3 = 7, zik 3 + 0 + 3 = 6 → **zik** (7 → 6). Φ: 21 → 20 (Δ = −1).

Adım Adım: En İyi Tepki Dinamiği ve Φ

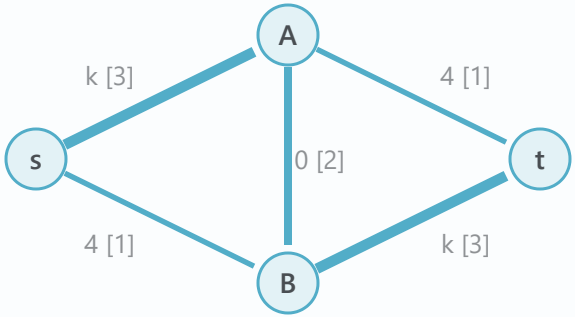


sürücü	yol	maliyet
1	alt	7
2	zik	6
3	zik	6
4	üst	7

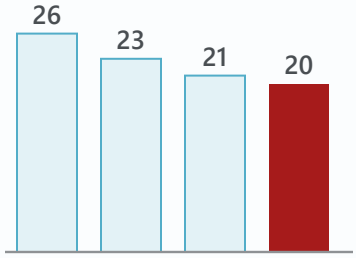


Sürücü 4 (üst, 7): alt 4 + 4 = 8, zik 3 + 0 + 4 = 7 → daha iyisi yok, **kalır**. Φ'nin her düşüşü, hamle yapanın kazancına eşitti.

Adım Adım: En İyi Tepki Dinamiği ve Φ



sürücü	yol	maliyet
1	alt	7
2	zik	6
3	zik	6
4	üst	7



İkinci turda kimse iyileştiremez → **saf Nash dengesi** (Φ = 20). Toplam 7 + 6 + 6 + 7 = 26; A→B olmasaydı 2+2 bölünüp 24 olurdu: atomik Braess!

Potansiyel Oyunlar: Ne Kazandık?

Varlık

Φ'nin en küçük değerini veren durum bir saf Nash dengesidir; denge her zaman vardır.

Yakınsama

İyileştiren hamleler dizisi sonsuza gidemez (sonlu iyileşme özelliği): her en iyi tepki dinamiği durur.

Hız (ileri okuma)

Durmak garanti, hızlı olmak değil: bazı tıkanıklık oyunlarında dengeyi bulmak için hızlı bir yöntem bilinmemektedir (Fabrikant, Papadimitriou, Talwar, 2004).

Genellik

Her (tam) potansiyel oyun bir tıkanıklık oyununa denktir (Monderer ve Shapley, 1996).

Öğrenme

Oyuncular sırayla en iyi tepki verirse dengeye ulaşılır.

Verimlilik

Φ'yi en küçük yapan denge, dengeler içinde iyi bir denge seçmenin yoludur.

AĞ OLUŞTURMA

Ağ Dışsallıkları: Olumsuz ve Olumlu

- **Dışsallık**: bir kullanıcının kararı başkalarının maliyetini ya da faydasını değiştirir, ama kullanıcı bunu kendi hesabına katmaz.

Olumsuz: tıkanıklık

- Her ek sürücü, aynı yoldaki diğer x kişiye toplam $x \cdot \ell'(x)$ gecikme yükler.
- İki yollu örnekte Yol 1'de 2.000 araç varken: $2.000 \times 1/100 = 20$ araç·dk.
- Sonuç: kalabalık yol gereğinden çok kullanılır; çare tıkanıklık ücreti.

Olumlu: ağ etkisi

+ Ağa katılmanın değeri katılan sayısıyla artar: telefon, mesajlaşma, sosyal ağ.
+ n kullanıcıda $n(n-1)/2$ olası bağlantı vardır; değerin n^2 ile büyüdüğü iddiası (Metcalfe) bir sezgidir.
+ Yeni kullanıcı başkalarına sağladığı faydayı hesaba katmaz: benimseme *kritik kitle* aşılana kadar yavaş kalır.

Ağ oluşturma oyununda ikisi bir arada: bir bağ yalnız kuranın değil, başkalarının da uzaklığını kısaltır (olumlu dışsallık), ama bedelini yalnız onu satın alan öder.

Ağ Oluşturma Oyunları (Network Formation)

- Buraya kadar ağ *veriydi*; şimdi ağın kendisini oyuncular kurar: internet eşlemesi (peering), sosyal ağlar, P2P ağları. Ağ oluşturma modellerinde bağlantıya (link) kısaca **bağ** da denir.
- Her bağın bir kurulum maliyeti vardır; iyi bağlı olmak (kısa uzaklık) ise değerlidir. Gerilim: **bağ maliyeti ↔ erişim**.

Fabrikant vd. (2003): tek taraflı

Oyuncu i istediği düğümlere bağ satın alır (her biri α). Maliyet: α·(aldığı bağ) + Σ_j uzaklık(i, j). Denge: Nash.

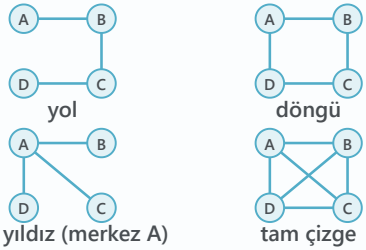
Jackson–Wolinsky (1996): iki taraflı

Bağ ancak iki taraf da isterse kurulur, biri isterse kopar. Denge kavramı: **ikili kararlılık** (pairwise stability).

Sorular: Hangi ağlar dengedir? Sosyal olarak en iyi ağ hangisi? İkisi ne kadar farklı olabilir?

İki Taraflı Bağ: İkili Kararlılık (n = 4)

- Fayda: uzaklığı 1 olan her oyuncudan 4, 2 olandan 2,2, 3 olandan 1.
- Her bağın bedeli iki uçtaki oyuncuların **her birine** 2.
- **İkili kararlı**: (i) hiçbir oyuncu kendi bağlarından birini koparınca kazanmaz; (ii) olmayan hiçbir bağ iki ucu *birlikte* kazançlı çıkarmaz.



Ağ	Faydalar	Toplam	İkili kararlı?
yol	5,2/6,2/6,2/5,2	22,8	Hayır: A–D eklenir (her biri +3 fayda, −2 bedel)
döngü	6,2 (hepsi)	24,8	Evet
yıldız	6/6,4/6,4/6,4	**25,2**	Evet; toplamı en yüksek ağ
tam çizge	6 (hepsi)	24	Hayır: bağ koparan 2 kazanır, 1,8 kaybeder

Yıldızda iki yaprak bağ kursa her biri 1,8 kazanır ama 2 öder: eklemez. Ağ Oluşturma oyununun "Social" senaryosu (4 düğüm) bu sayıları kullanır.

İkili kararlı ağlar birden fazla olabilir ve toplamı en yüksek olanla örtüşmek zorunda değildir: döngü kararlıdır ama yıldızdan 0,4 geridedir.

Adım Adım: Ağ Oluşturma Oyunu (n = 4, α = 1,5)



4 oyuncu, bağ fiyatı α = 1,5

maliyet_i = α · (aldığı bağ)

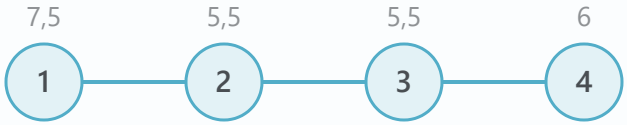
+ Σ_j uzaklık(i, j)

Sosyal maliyet = Σ maliyet_i

Kopuk ağ → maliyet ∞

Her oyuncu hangi bağları *kendisi* satın alacağını seçer; bağı kim alırsa alsın herkes kullanır.

Adım Adım: Ağ Oluşturma Oyunu (n = 4, α = 1,5)



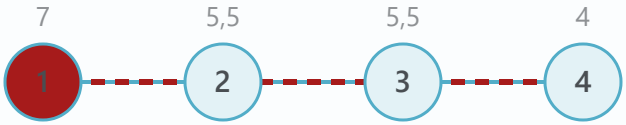
Yol 1–2–3–4 (i, i+1'i alır)

maliyetler 7,5 · 5,5 · 5,5 · 6

sosyal maliyet 24,5

Yol ağı: 1 bir bağ alır (1,5) ve uzaklıkları 1 + 2 + 3 = 6 → 7,5. Uçtaki oyuncular çok uzak kalıyor.

Adım Adım: Ağ Oluşturma Oyunu (n = 4, α = 1,5)



1, 4'e bağ alırsa:

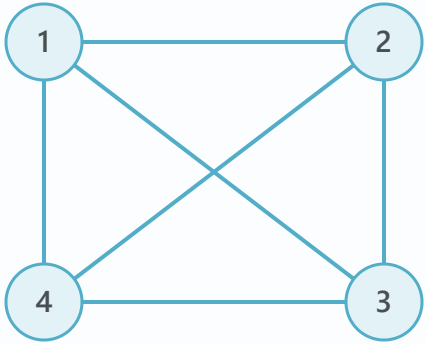
$2 \cdot 1{,}5 + (1 + 2 + 1) = 7$

$7 < 7{,}5 \rightarrow$ sapar

Yol ağı denge değil

Oyuncu 1, 4'e ikinci bir bağ alırsa uzaklıkları 6'dan 4'e iner: $3 + 4 = 7 < 7{,}5 \rightarrow$ **kârlı sapma**.

Adım Adım: Ağ Oluşturma Oyunu (n = 4, α = 1,5)



Tam çizge: 6 bağ

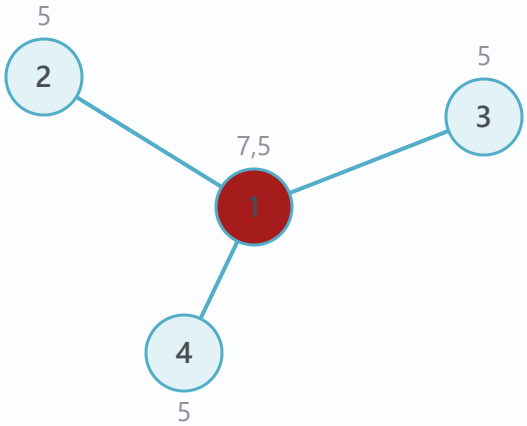
sosyal: 6 · 1,5 + 12 = 21

Bağını bırakan: −1,5 + 1

→ kazanır, denge değil

Tam çizge sosyal açıdan en ucuzu (21). Ama bağ almış biri onu bırakırsa 1,5 kazanır, yalnız 1 uzaklık kaybeder → denge değil.

Adım Adım: Ağ Oluşturma Oyunu (n = 4, α = 1,5)



Merkez 1 üç bağı alır

merkez 4,5 + 3 = 7,5; yaprak 5

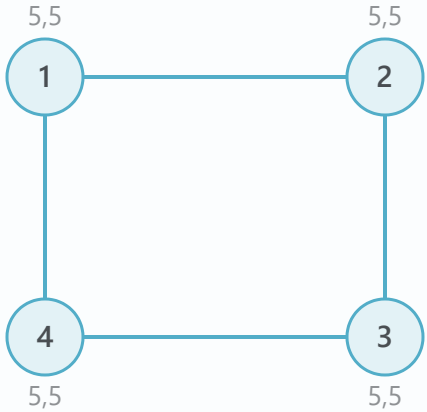
Yaprak bağ alırsa 1,5 + 4 = 5,5

Merkez bırakırsa ağ kopar

→ Nash dengesi (22,5)

Yıldız: yaprak yeni bağla yalnız 1 uzaklık kazanır (1 < 1,5); merkez bağ bırakırsa ağ kopar → **kimse sapmaz**.

Adım Adım: Ağ Oluşturma Oyunu (n = 4, α = 1,5)



4-döngü (her biri bir bağ): 5,5

sosyal maliyet 22

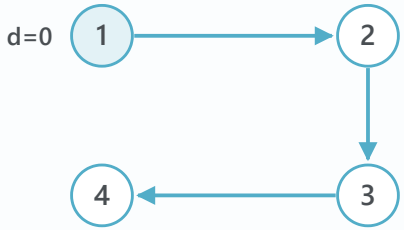
Betik: tüm 4.096 profil tarandı

dengeler: yıldız 22,5 · döngü 22

optimum: tam çizge 21

Kaba kuvvet taraması: α = 1,5'te dengeler yıldız (22,5) ve 4-döngü (22); optimum (21) denge değil.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [1]

uz: 1:0

Σ uzaklık = 0

ok: bağı alan → diğer uç · kırmızı: yeni bağ

Yol ağı 1–2–3–4; 1 yalnız (1, 2) bağını almış. BFS i = 1'den başlar: uz = {1: 0}.

Adım 1 / 12

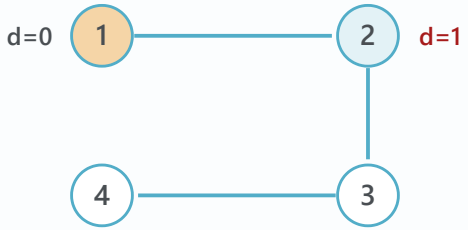
```
1  def maliyet(i, bag, alpha):
▶    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
4      u = kuyruk.pop(0)
5      for v in komsu(bag, u):
6        if v not in uz:
7          uz[v] = uz[u] + 1
8          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

DEĞİŞKENLER

`uz = {1: 0}` `kuyruk = [1]`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [2]

uz: 1:0 2:1

Σ uzaklık = 1

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 1 kuyruktan çıkar; yeni komşular 2 → uzaklık 1.

Adım 2 / 12

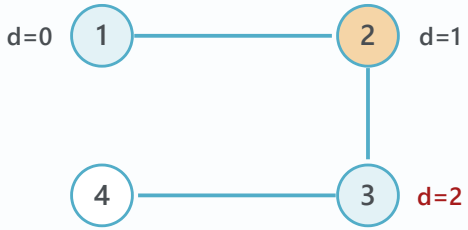
```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

DEĞİŞKENLER

`uz = {1: 0, 2: 1}` `kuyruk = [2]` `u = 1`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [3]

uz: 1:0 2:1 3:2

Σ uzaklık = 3

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 2 kuyruktan çıkar; yeni komşular 3 → uzaklık 2.

Adım 3 / 12

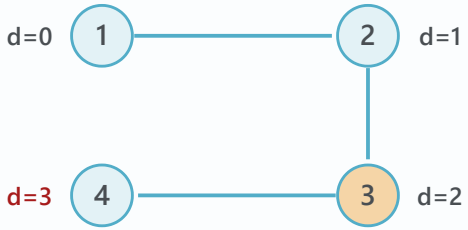
```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2}` `kuyruk = [3]` `u = 2`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [4]

uz: 1:0 2:1 3:2 4:3

Σ uzaklık = 6

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 3 kuyruktan çıkar; yeni komşular 4 → uzaklık 3.

Adım 4 / 12

```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

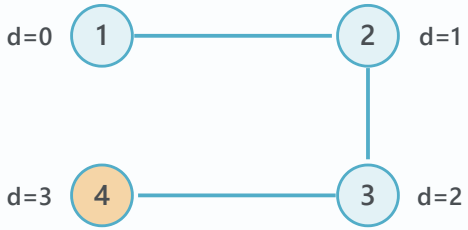
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 3}`

`kuyruk = [4]` `u = 3`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: []

uz: 1:0 2:1 3:2 4:3

Σ uzaklık = 6

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 4 kuyruktan çıkar; bütün komşuları zaten ziyaretli.

Adım 5 / 12

```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

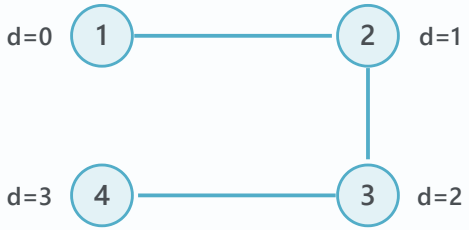
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 3}`

`kuyruk = []` `u = 4`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: []

uz: 1:0 2:1 3:2 4:3

Σ uzaklık = 6

ok: bağı alan → diğer uç · kırmızı: yeni bağ

Herkes erişilebilir. aldigi = 1 → maliyet 1,5·1 + 6 = **7,5**. Şimdi bir bağ daha almayı deneyelim.

Adım 6 / 12

```
1 def maliyet(i, bag, alpha):
2   uz = {i: 0}; kuyruk = [i]
3   while kuyruk:
4     u = kuyruk.pop(0)
5     for v in komsu(bag, u):
6       if v not in uz:
7         uz[v] = uz[u] + 1
8         kuyruk.append(v)
▶   aldigi = sum(1 for (a, b) in bag if a == i)
▶   return alpha * aldigi + sum(uz.values())
```

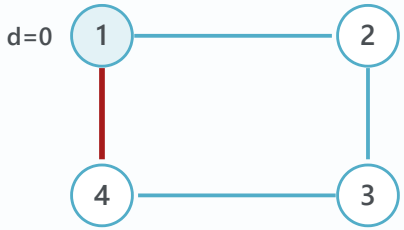
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 3}`

`kuyruk = []` `aldigi = 1`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [1]

uz: 1:0

Σ uzaklık = 0

ok: bağı alan → diğer uç · kırmızı: yeni bağ

Sapma: 1 ayrıca (1, 4) bağını satın alır (turuncu). Aynı hesap yeniden. BFS i = 1'den başlar: uz = {1: 0}.

Adım 7 / 12

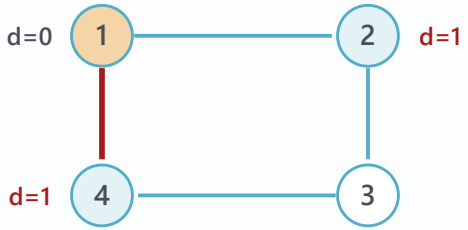
```
1  def maliyet(i, bag, alpha):
▶    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
4      u = kuyruk.pop(0)
5      for v in komsu(bag, u):
6        if v not in uz:
7          uz[v] = uz[u] + 1
8          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

DEĞİŞKENLER

`uz = {1: 0}` `kuyruk = [1]`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [2, 4]

uz: 1:0 2:1 4:1

Σ uzaklık = 2

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 1 kuyruktan çıkar; yeni komşular 2, 4 → uzaklık 1.

Adım 8 / 12

```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

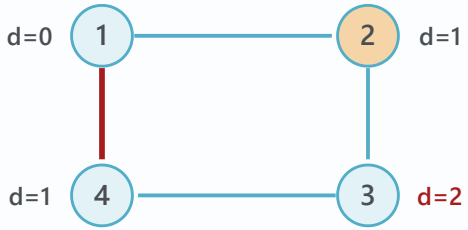
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 4: 1}`

`kuyruk = [2, 4]` `u = 1`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [4, 3]

uz: 1:0 2:1 3:2 4:1

Σ uzaklık = 4

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 2 kuyruktan çıkar; yeni komşular 3 → uzaklık 2.

Adım 9 / 12

```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

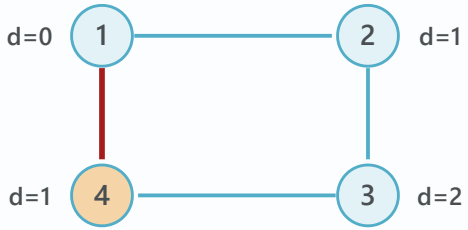
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 1}`

`kuyruk = [4, 3]` `u = 2`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: [3]

uz: 1:0 2:1 3:2 4:1

Σ uzaklık = 4

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 4 kuyruktan çıkar; bütün komşuları zaten ziyaretli.

Adım 10 / 12

```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

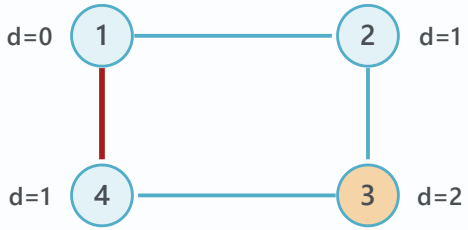
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 1}`

`kuyruk = [3]` `u = 4`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: []

uz: 1:0 2:1 3:2 4:1

Σ uzaklık = 4

ok: bağı alan → diğer uç · kırmızı: yeni bağ

u = 3 kuyruktan çıkar; bütün komşuları zaten ziyaretli.

Adım 11 / 12

```
1  def maliyet(i, bag, alpha):
2    uz = {i: 0}; kuyruk = [i]
3    while kuyruk:
▶      u = kuyruk.pop(0)
▶      for v in komsu(bag, u):
▶        if v not in uz:
▶          uz[v] = uz[u] + 1
▶          kuyruk.append(v)
9    aldigi = sum(1 for (a, b) in bag if a == i)
10   return alpha * aldigi + sum(uz.values())
```

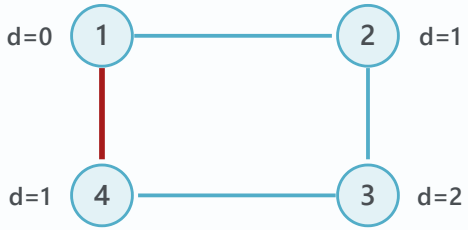
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 1}`

`kuyruk = []` `u = 3`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Kod İzleme: `maliyet(i, bag, alpha)`



oyuncu i = 1

kuyruk: []

uz: 1:0 2:1 3:2 4:1

Σ uzaklık = 4

ok: bağı alan → diğer uç · kırmızı: yeni bağ

Herkes erişilebilir. aldigi = 2 → maliyet 1,5·2 + 4 = **7**. 7 < 7,5 → **kârlı sapma**: yol ağı Nash dengesi değildir.

Adım 12 / 12

```
1 def maliyet(i, bag, alpha):
2   uz = {i: 0}; kuyruk = [i]
3   while kuyruk:
4     u = kuyruk.pop(0)
5     for v in komsu(bag, u):
6       if v not in uz:
7         uz[v] = uz[u] + 1
8         kuyruk.append(v)
▶   aldigi = sum(1 for (a, b) in bag if a == i)
▶   return alpha * aldigi + sum(uz.values())
```

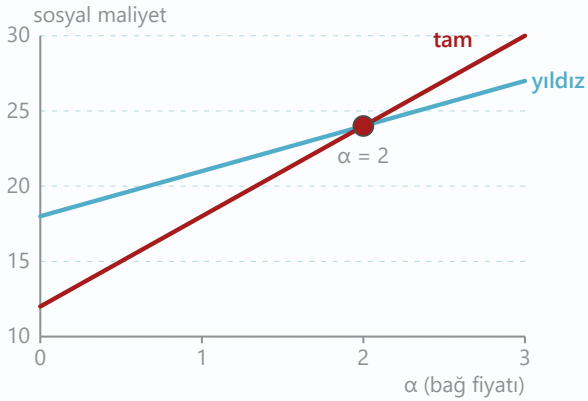
DEĞİŞKENLER

`uz = {1: 0, 2: 1, 3: 2, 4: 1}`

`kuyruk = []` `aldigi = 2`

komsu(bag, u): kimin aldığına bakmadan komşular (bağı herkes kullanır); ağ bağlı varsayılır. Ödeme yalnız *kendi* aldığı bağlar için.

Bağ Fiyatı Değişince Denge ve Optimum



α	Optimum	Dengeler (betik)
0,5	tam (15)	tam (15)
1,5	tam (21)	döngü 22, yıldız 22,5
2,5	yıldız (25,5)	yıldız 25,5 · yol 27,5

n = 4 için yıldız 3α + 18, tam çizge 6α + 12: α < 2 iken tam çizge, α > 2 iken yıldız daha ucuz. Genel n'de de optimum α ≤ 2 için tam çizge, α ≥ 2 için yıldızdır (Fabrikant vd., 2003).

α < 1: bağ ucuz, herkes bağlanır → tam çizge hem denge hem optimum. α > 1: yıldız her zaman bir dengedir; ama yanında daha pahalı dengeler de çıkabilir (α = 2,5'te yol ağı; bağları ortadakiler alır).

Ağlarda Yönlendirme: Merkezî mi, Dağıtık mı?

	Merkezî yönlendirme	Dağıtık (bencil) yönlendirme
Kim karar verir?	Trafik kontrol merkezi, SDN (yazılım tanımlı ağ) denetleyicisi, internet servis sağlayıcısı (İSS) trafik mühendisliği	Her sürücü / her yönlendirici (navigasyon uygulamaları, BGP: ağlar arası yönlendirme protokolü)
Hedef	Toplam gecikme C(f) en küçük (Wardrop 2. ilke)	Kendi gecikmesi en küçük (Wardrop 1. ilke)
İki yollu örnek	1.500 / 1.500 → 71.250 araç·dk	2.000 / 1.000 → 75.000 araç·dk
Gereken	Tüm talep ve gecikme bilgisi; kullanıcıların *uyması*	Yalnızca yerel bilgi; uyum sorunu yok
Zayıf yan	Optimumda bazıları daha yavaş yolda → sapma isteği; tek nokta hatası	Braess gibi verimsiz dengeler

Ağlarda ara çözümler yaygındır: merkez yalnızca **fiyat** (tıkanıklık ücreti), **bilgi** (rota önerisi) veya trafiğin bir kısmını kontrol eder (Stackelberg yönlendirme: o kısmı merkez kendisi yönlendirir); gerisini kullanıcılar seçer.

KONTROL VE TEŞVİKLER

Teşvikler, Yük Dengeleme ve Gerçek Hayat

Doğru teşvikler

+ Kullanıcıları ağ kaynaklarını dengeli kullanmaya yönlendirir → verimlilik artar.
+ Trafik: ücret veya uyarılar yoğunluğu azaltır.
+ Akıllı şebeke: zamana bağlı tarife pik saatlerde yükü dengeler.

Yanlış teşvikler

– Stratejik, bencil davranış artar → tıkanıklık ve düşük verimlilik.
– Herkese aynı "en hızlı" rotayı öneren uygulama yan yolları doldurabilir.
– Ücret yanlış seçilirse trafik başka darboğaza kayar.

İnternet

İSS'ler veri paketlerini yönlendirir; BGP ile ağlar arası rota seçimi.

Trafik yönetimi

Rota optimizasyonu, akıllı sinyalizasyon, navigasyon uygulamaları.

Enerji

Akıllı elektrik şebekelerinde enerji dağıtımı ve yük kaydırma.

Verimlilik ile Adalet: Bant Genişliği Paylaşımı

İki bağlantının kapasitesi 1'dir. Akış A ikisini de kullanır, B yalnız 1. bağlantıyı, C yalnız 2. bağlantıyı: $x_A + x_B \le 1$, $x_A + x_C \le 1$. Hangi paylaşım seçilmeli?

Ölçüt	Ne en iyilenir?	(x_A, x_B, x_C)	Toplam
En büyük toplam akış	Σx_i	(0, 1, 1)	2 (A aç kalır)
Maksimin adalet (eşitlikçi ölçüt)	Önce en küçük payı olabildiğince büyüt, sonra bir sonrakini	(1/2, 1/2, 1/2)	1,5
Orantılı adalet (Nash sosyal refahı)	Payların çarpımını maksimize et (Kelly, 1997)	(1/3, 2/3, 2/3)	≈ 1,67

Orantılı adalette iki bağlantıyı birden kullanan A daha az alır, ama kimse sıfıra düşmez: sıfır pay çarpımı sıfırlar. $x_A(1 - x_A)^2$ çarpımını maksimize etmek $\log x_A + 2\log(1 - x_A)$ toplamını maksimize etmekle aynıdır; log x'in türevi 1/x olduğundan $1/x_A = 2/(1 - x_A) \Rightarrow x_A = 1/3$. Kelly vd. (1998) TCP benzeri tıkanıklık denetimini, bağlantı başına tıkanıklık fiyatıyla bu paylaşıma yakınsayan dağıtık bir algoritma olarak yorumlar.

Tartışma Soruları

- Navigasyon uygulamaları herkese “en hızlı” rotayı önerdiğinde Wardrop dengesine mi, sistem optimumuna mı yaklaşırız? Neden?
- Bir şehir yeni köprü planlıyor. Braess etkisini önceden görmek için, köprü açıldığında sürücülerin varacağı denge nasıl hesaplanır?
- İnternette yönlendiriciler (BGP) bencil ajan mıdır? Kimin maliyetini en aza indirirler?

DEĞERLENDİRME

Özet

Kavram	Özü	Örnek
Bencil yönlendirme	Herkes kendi gecikmesini küçültür	3.000 araç, iki yol
Wardrop dengesi	Kullanılan yollar eşit ve en hızlı	2.000 / 1.000, 25 dk
Sistem optimumu	Marjinal maliyetler eşit	1.500 / 1.500, 71.250 araç·dk
Braess paradoksu	Yeni yol → herkes daha yavaş	65 → 80 dk
Rosenthal potansiyeli	$\Delta\Phi$ = hamle yapanın maliyet farkı	Φ: 26 → 23 → 21 → 20
Ağ oluşturma	Bağ maliyeti ↔ erişim	α = 1,5: yıldız / döngü denge, tam çizge optimum

Denge ile optimum çoğu zaman ayrışır: sürücü kendi gecikmesine, planlayıcı marjinal maliyete bakar.

DEĞERLENDİRME

Örnek Problemler

1. 6.000 araç; Yol 1: $\ell_1 = 10 + x_1/200$, Yol 2: $\ell_2 = 25 + x_2/100$. Wardrop dengesini bulun.
2. Aynı ağda sistem optimumunu bulun.
3. Braess ağında (x/100 ve 45 dk) araç sayısı 6.000 olsun. Yeni yoldan önce ve sonra denge süresi nedir?
4. Atomik örneğin dengesinde (alt, zik, zik, üst) Φ'yi hesaplayın. Sürücü 4 zik'e geçerse Φ nasıl değişir?
5. n = 4, α = 0,5. Tam çizge bir Nash dengesi midir? Yıldız?
6. 1. sorudaki ağda `denge_bul` ilk iki turda hangi x değerlerini dener, aralık ne olur?

▶ **Cevapları göster**

DEĞERLENDİRME

Etkileşimli Oyunlar ve Simülatörler

Braess Paradoksu

Yeni yol açılınca ve kapanınca denge süresi.

Bencil Yönlendirme

Trafiğin yollara dağılımı: Wardrop dengesi ve optimum.

Ağ Oluşturma

İki taraflı bağ ve ikili kararlılık: bağ fiyatına göre hangi ağlar kararlıdır?

Kaynaklar

- T. Roughgarden, *Selfish Routing and the Price of Anarchy*, MIT Press, 2005.
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (ed.), *Algorithmic Game Theory*, Cambridge University Press, 2007 (Bölüm 18–19).
- J. G. Wardrop, "Some Theoretical Aspects of Road Traffic Research", *Proc. Institution of Civil Engineers*, 1952.
- D. Braess, "Über ein Paradoxon aus der Verkehrsplanung", *Unternehmensforschung*, 1968.
- R. W. Rosenthal, "A Class of Games Possessing Pure-Strategy Nash Equilibria", *Int. J. Game Theory*, 1973.
- D. Monderer, L. Shapley, "Potential Games", *Games and Economic Behavior*, 1996.
- A. Fabrikant vd., "On a Network Creation Game", *PODC*, 2003. · M. Jackson, A. Wolinsky, "A Strategic Model of Social and Economic Networks", *J. Economic Theory*, 1996.

DEĞERLENDİRME

Kaynaklar: Makaleler, Kitaplar ve Dersler

- A. Fabrikant, C. Papadimitriou, K. Talwar, "The Complexity of Pure Nash Equilibria", *STOC*, 2004. · M. Beckmann, C. B. McGuire, C. B. Winsten, *Studies in the Economics of Transportation*, Yale University Press, 1956.
- A. Rapoport, T. Kugler, S. Dugar, E. J. Gisches, "Choice of Routes in Congested Traffic Networks: Experimental Tests of the Braess Paradox", *Games and Economic Behavior* 65(2), 2009. · W. Knödel, *Graphentheoretische Methoden und ihre Anwendungen*, Springer, 1969. · G. Kolata, "What if They Closed 42d Street and Nobody Noticed?", *The New York Times*, 25 Aralık 1990.
- I. Milchtaich, "Network Topology and the Efficiency of Equilibrium", *Games and Economic Behavior* 57, 2006. · T. Roughgarden, "On the Severity of Braess's Paradox", *J. Computer and System Sciences* 72, 2006.
- F. Kelly, "Charging and Rate Control for Elastic Traffic", *European Trans. Telecommunications* 8, 1997. · F. Kelly, A. Maulloo, D. Tan, *J. Operational Research Society* 49, 1998.
- D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, Cambridge University Press, 2010 (Bölüm 8: trafik ağları ve Braess; Bölüm 17: ağ etkileri).
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, American Mathematical Society, 2017 (anarşinin bedeli ve trafik yönlendirme).
- Ders sayfaları: T. Roughgarden, CS364A (Stanford) · M. Hoefer, AGT (Frankfurt) · M. Dinitz, AGT (Johns Hopkins)

Son

Bölüm 8: Ağ Tasarımı ve Yönlendirme

Sonraki bölüm: Bölüm 9: Anarşinin Bedeli

Çalışma soruları · Terim sözlüğü