Bölüm 9: Anarşinin Bedeli

Algoritmik Oyun Kuramı

PoA ve PoS | Pigou ağı | 4/3 sınırı | Pürüzsüzlük | Braess ağında PoA | Yük dengeleme | İstikrarın Bedeli | Tıkanıklık ücreti

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_09_Anarsinin_Bedeli.html

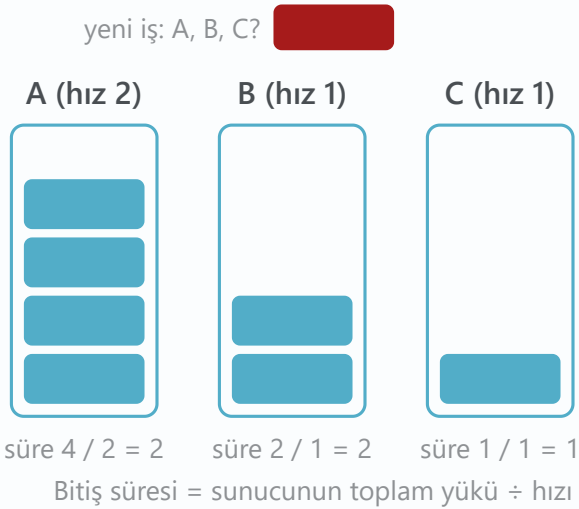
Oyunla Başlangıç: Yük Dengeleme Oyunu

Oyunun kuralı

İşler sırayla gelir ve A, B, C sunucularından birine yerleşir; bir işin bitiş süresi, sunucunun toplam yükünün sunucu hızına bölümüdür.

Merak sorusu

- Her iş kendi bitiş süresini en aza indirirse sistemin toplam maliyeti de en küçük olur mu?
- Bencil işlerin dengesi en iyi atamadan en çok ne kadar kötü olabilir?



[Yük Dengeleme oyunu →]

TEMEL KAVRAMLAR

Öğrenme Çıktıları

Tanımlamak

Anarşinin Bedeli (PoA) ve İstikrarın Bedeli'ni (PoS) tanımlamak; aralarındaki farkı örnekle göstermek.

Hesaplamak

Pigou, Braess ve yük dengeleme örneklerinde denge ve optimum maliyetini bulup oranı hesaplamak.

Sınırı açıklamak

Doğrusal gecikmede PoA ≤ 4/3 sınırının sezgisel ispatını ve sıkılığını açıklamak.

İyileştirmek

Tıkanıklık ücretinin optimumu nasıl dengeye dönüştürdüğünü iki yollu ağda ve Pigou ağında göstermek.

TEMEL KAVRAMLAR

Anarşinin Bedeli: Bencilliğin Faturası

- Anarşinin Bedeli (Price of Anarchy, PoA), bencil davranışın oluşturduğu **en kötü denge** ile sosyal optimum arasındaki verimlilik kaybını ölçer.
- Maliyet problemlerinde sosyal optimum toplam maliyeti en aza indirir.
- Trafik ağında kullanıcılar kendi yolculuk sürelerini azaltmaya çalışırken toplam gecikme artabilir.
- Kavramı Koutsoupias ve Papadimitriou (1999) "koordinasyon oranı" adıyla tanımladı; "anarşinin bedeli" adını Papadimitriou (2001) verdi: merkezî bir "diktatör" yerine herkesin kendi başına karar vermesinin (anarşi) bedeli.

Örnek: iki yollu ağda Wardrop dengesi 75.000, sistem optimumu 71.250 araç·dk (oran 75.000 / 71.250 ≈ 1,053); Braess ağında yeni yol süreyi 65'ten 80 dk'ya çıkardı. Anarşinin Bedeli bu farkları **tek bir sayıyla** ölçer ve sınırlar.

TEMEL KAVRAMLAR

Nash Dengesi ve Sosyal Optimum

Nash dengesi (bireysel)

+ Hiçbir oyuncu tek taraflı strateji değiştirerek kendi sonucunu iyileştiremez.
+ Bireysel teşvikleri temsil eder.
+ Birden fazla olabilir (iyi ve kötü dengeler).

Sosyal optimum (sistem)

– Toplam maliyeti en aza indirir veya sosyal refahı en yükseğe çıkarır.
– Sistemin ortak hedefini temsil eder.
– Çoğu zaman bir denge değildir: birileri sapmak ister.

Neden ayrışırlar? Bireysel rasyonellik (herkes kendi faydasına bakar), dışsallıklar (başkasına yüklenen maliyet hesaba katılmaz) ve bilgi eksikliği. Bu iki sonuç arasındaki oran Anarşinin Bedeli ile ölçülür.

TEMEL KAVRAMLAR

Tanımlar: PoA ve PoS

Anarşinin Bedeli
$PoA = \max_{s \in NE} C(s) / C(OPT)$
en kötü denge: kötümser garanti

İstikrarın Bedeli (Price of Stability)
$PoS = \min_{s \in NE} C(s) / C(OPT)$
en iyi denge: iyimser senaryo

NE: oyunun Nash dengeleri; C(s): s durumundaki toplam maliyet; OPT: toplam maliyeti en küçük durum (sosyal optimum). PoA = en kötü dengenin maliyeti ÷ optimumun maliyeti; PoS = en iyi dengenin maliyeti ÷ optimumun maliyeti.



- Her zaman **1 ≤ PoS ≤ PoA**. Tek denge varsa ikisi eşittir.
- Fayda (refah) problemlerinde oran ters çevrilir: C(OPT) yerine SW(OPT) / SW(denge) (SW: sosyal refah); böylece büyük değer yine daha kötü verimlilik demektir.

TEMEL KAVRAMLAR

Verimlilik Ölçütleri ve Okuma

- **Toplam maliyet**: tüm kullanıcıların gecikme veya kaynak kullanım maliyetlerinin toplamı.
- Maliyet problemlerinde PoA, en kötü denge maliyetinin sosyal optimum maliyetine oranıdır.
- PoA'nın 1 olması verimlilik kaybı bulunmadığını gösterir.
- Oran büyüdükçe stratejik davranışın oluşturduğu kayıp artar.

PoA	Anlamı	Örnek
1	kayıp yok	Braess, yeni yol öncesi
4/3	%33 fazla maliyet	Pigou, doğrusal sınır
$2 - 2/(m+1)$	< 2 kat	yük dengeleme
sınırsız	garanti yok	dik gecikme (x^p)

Neden *en kötü* durum? PoA, oyuncuların hangi dengeye gideceğini bilmeden verilebilecek bir **garanti**dir: "Ne olursa olsun kayıp en fazla bu kadar." Algoritmalardaki yaklaşım oranı fikrinin oyun kuramındaki karşılığıdır.

TEMEL KAVRAMLAR

İki Kötü Durum Oranı: Neyin Eksikliği?

Oran	Optimuma ulaşmayı engelleyen	Pay / payda
Yaklaşım oranı	hesaplama gücü: polinom zamanda çözülemeyen (NP-zor) problem	hızlı bir algoritmanın bulduğu çözüm ÷ optimum
Anarşinin Bedeli	koordinasyon: merkezî otorite yok, herkes kendi çıkarına bakar	en kötü denge / optimum

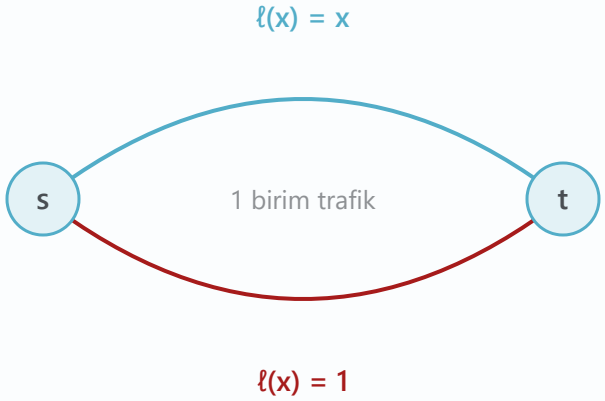
Aynı problem, iki kayıp

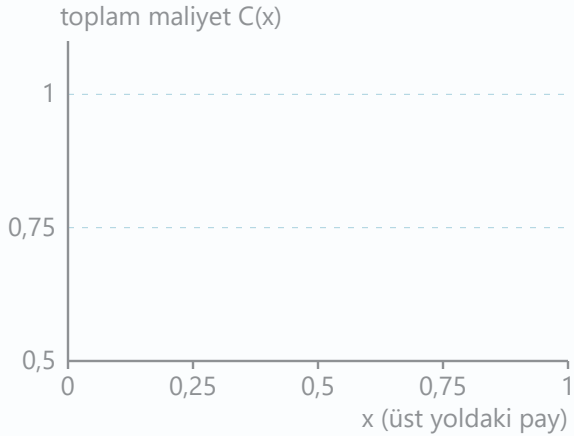
Özdeş m makinede en büyük yük: Graham'ın açgözlü liste algoritması en çok 2 − 1/m kat kötü; bencil işlerin saf dengesi en çok 2 − 2/(m+1) kat. m = 2 için 3/2 ve 4/3.

Analizin sınırları

Gerçek oyuncular yalnız bencil değildir: güven, sosyal normlar, itibar ve tekrarlanan etkileşim işbirliğini artırabilir. PoA bu yüzden kötümser bir garantidir, tahmin değil.

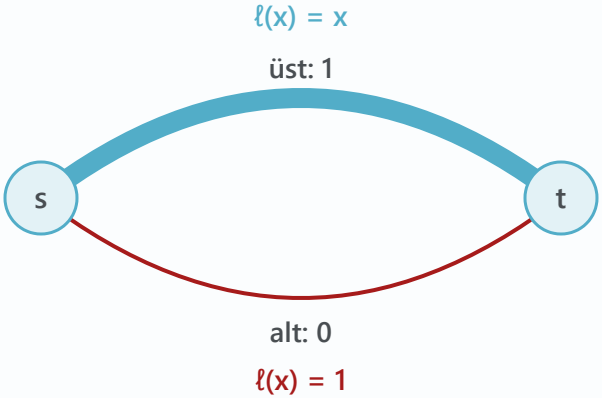
Adım Adım: Pigou Ağında Anarşinin Bedeli

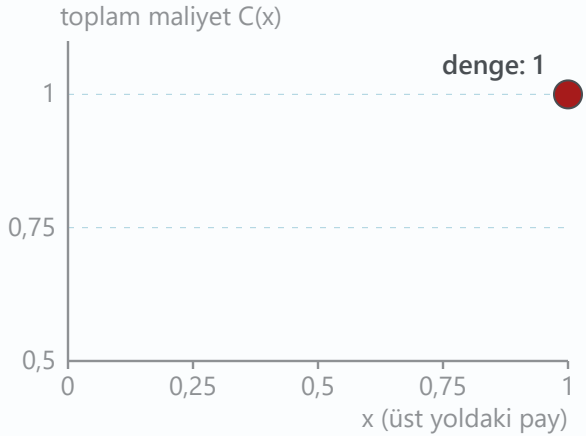




Kurulum (Pigou, 1920): 1 birim trafik, çok sayıda küçük sürücü. Üst yol: gecikme x (üstteki pay); alt yol: her zaman 1.

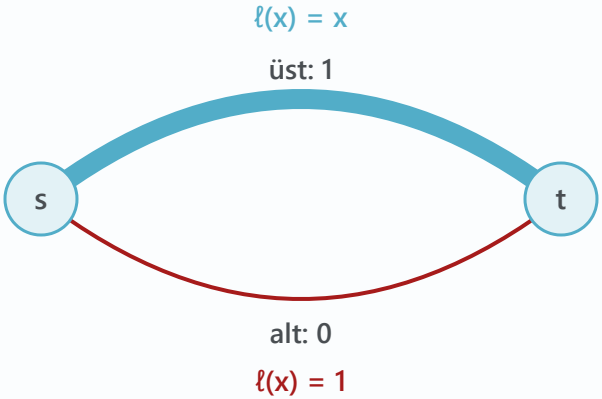
Adım Adım: Pigou Ağında Anarşinin Bedeli

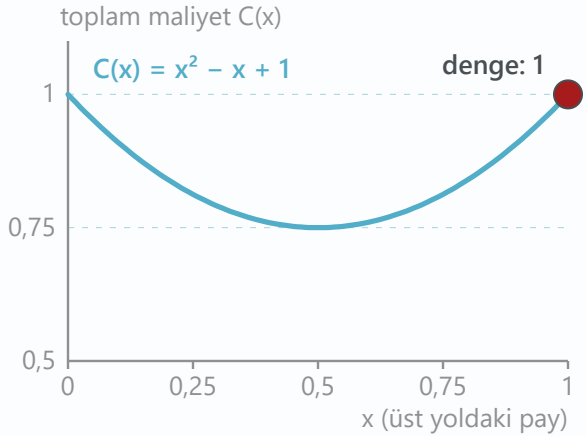




Denge: üst yol hiçbir zaman alttan yavaş değil (x ≤ 1) → herkes üste geçer. x = 1, herkes 1 dk: **C(denge) = 1**.

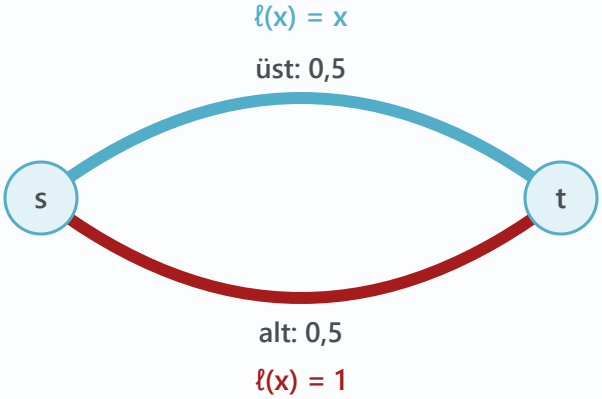
Adım Adım: Pigou Ağında Anarşinin Bedeli

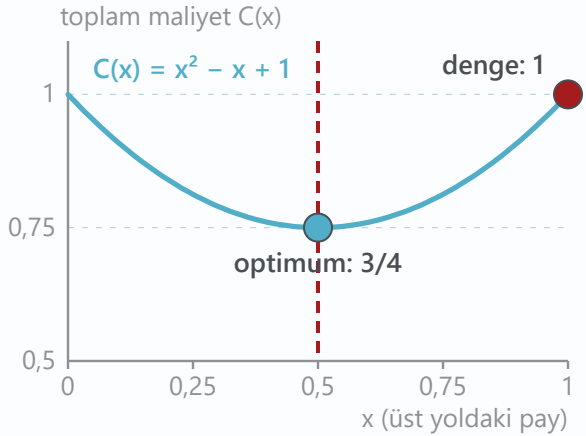




Toplam maliyet: x pay üstte x dk, 1 − x pay altta 1 dk → $C(x) = x \cdot x + (1 - x) \cdot 1 = x^2 - x + 1$.

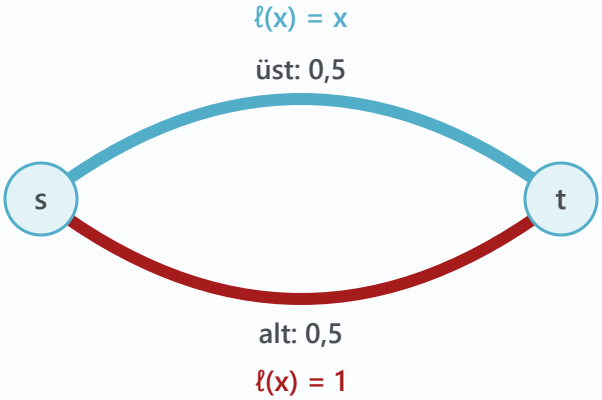
Adım Adım: Pigou Ağında Anarşinin Bedeli

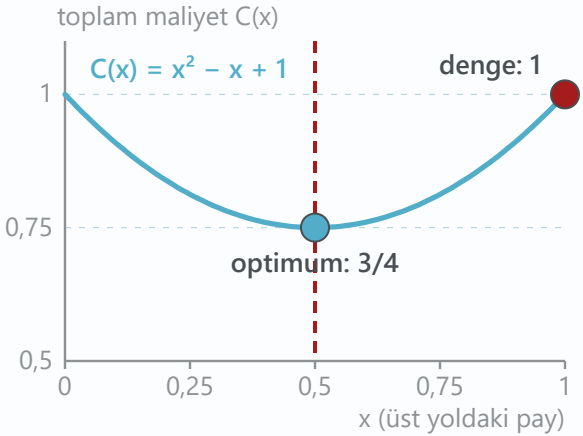




En küçük değer: C′(x) = 2x − 1 = 0 → **x = 1/2**. Trafiğin yarısı “feda edilip” alta gönderilir.

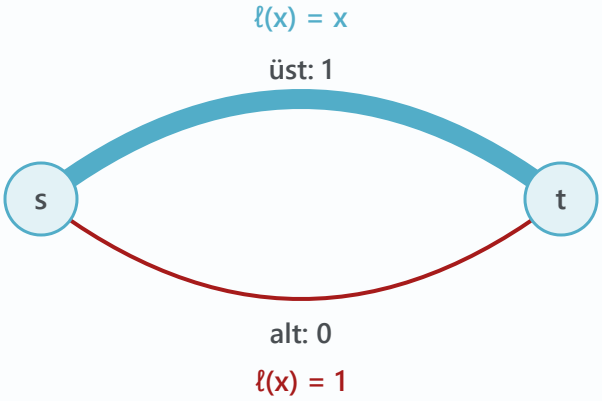
Adım Adım: Pigou Ağında Anarşinin Bedeli

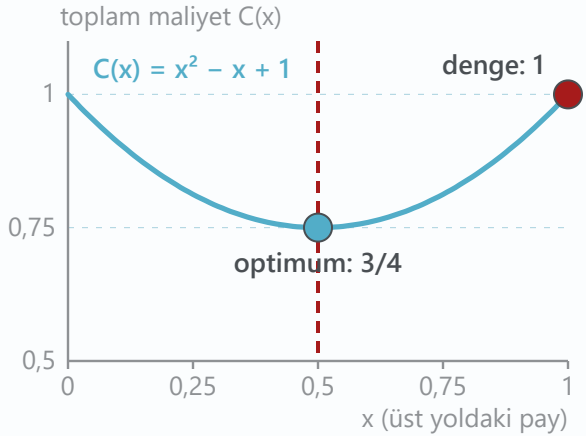




C(1/2) = 1/4 + 1/2 = **3/4**. Üsttekiler 1/2 dk, alttakiler 1 dk; alttakiler üste geçmek ister → optimum denge değil.

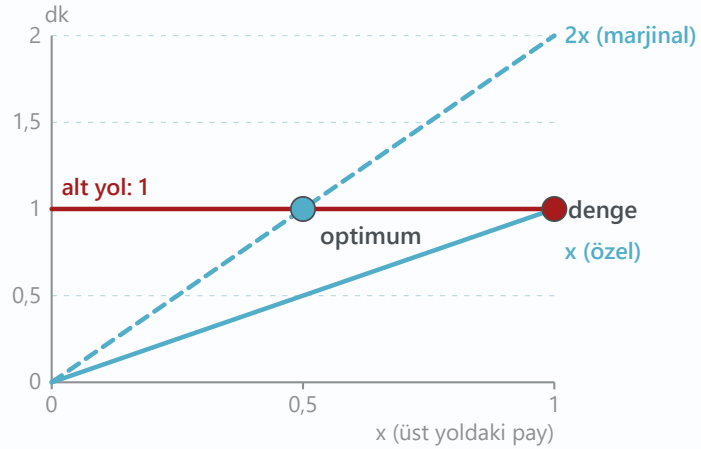
Adım Adım: Pigou Ağında Anarşinin Bedeli





PoA = C(denge) / C(optimum) = 1 / (3/4) = 4/3 ≈ 1,33: bencil yönlendirme %33 daha fazla toplam gecikme üretir.

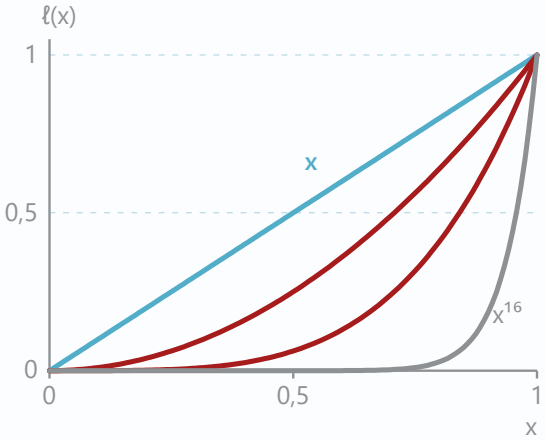
Neden Farklı? Özel ve Marjinal Maliyet



- Sürücü **özel** maliyetine bakar: x = 1 ile karşılaştırır → x = 1.
- Toplum **marjinal** maliyete bakar: üst yolun toplam maliyeti x·x, türevi 2x; 2x = 1 → x = 1/2.
- Aradaki fark x·ℓ′(x): bir sürücünün diğerlerine yüklediği gecikme (dışsallık).

Genel ilke: dengede kullanılan yolların gecikmeleri eşitlenir. Optimumda ise “kendi gecikmem + benim yüzümden ötekilere eklenen gecikme” eşitlenir. Bir yolda f araç varsa ve gecikme ℓ(f) ise eklenen gecikme f·ℓ′(f)'dir (ℓ′: gecikmenin türevi). Bu fark, tıkanıklık ücretinin dayanağıdır.

Gecikme Dikleşince PoA Büyür



p ($\ell = x^p$)	x^{OPT}	C(OPT)	PoA
1	0,5	0,75	**1,333**
2	0,577	0,615	**1,626**
4	0,669	0,465	**2,151**
8	0,76	0,325	**3,081**
16	0,838	0,212	**4,727**

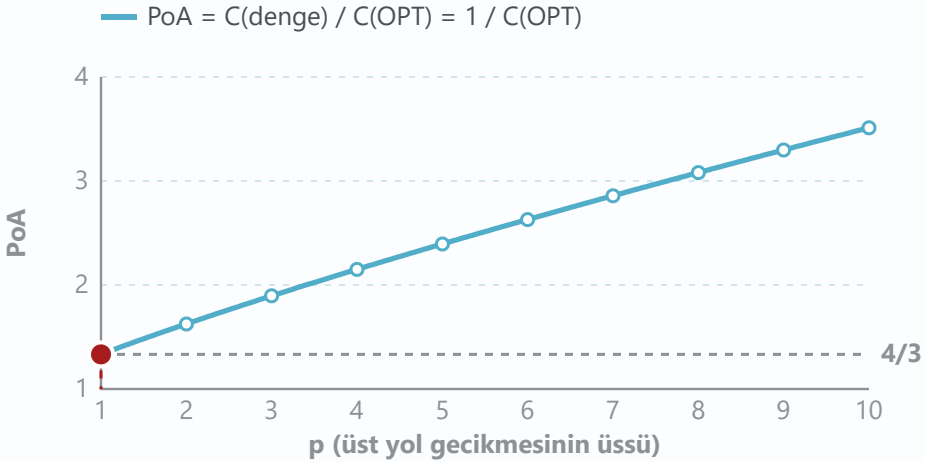
Denge her durumda herkesi üst yola yollar (maliyet 1). Gecikme dikleştikçe (p büyüdükçe) optimum iyileşir ve **PoA yavaş ama sınırsız büyür**: p = 16'da yaklaşık 4,7. Yani 4/3 evrensel değildir; doğrusal gecikmeye özgüdür.

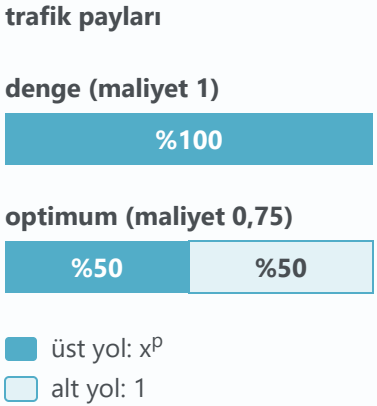
Pigou Ağında Gecikme Üssü ve PoA

Üst yol x^p, alt yol 1 · $C(x) = x^{p+1} + (1 - x)$ · $C'(x) = 0 \Rightarrow$ **$x^* = (p + 1)^{-1/p}$** · **PoA = 1 / C(x*)**

p = 1

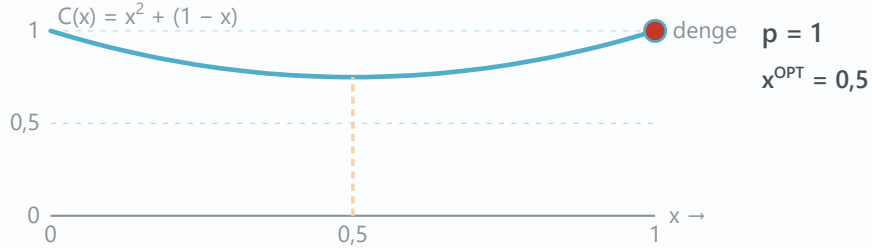
$x^* = (p + 1)^{-1/p}$ = **0,5** | $C(OPT) = 1 - p(p + 1)^{-(p+1)/p}$ = **0,75** | C(denge) = **1** | PoA = **1,333**





Gecikme fonksiyonu dikleştikçe optimum iyileşir, denge maliyeti 1'de kalır ve PoA sınırsız büyür; doğrusal gecikmede (p = 1) PoA = **4/3**'tür.

Kod İzleme: `pigou_poa(p)`



p = 1 (Pigou): $C(x) = x^2 + 1 - x$, $C'(x) = 2x - 1 = 0 \Rightarrow x^{OPT} = 2^{-1} = 0{,}5$.

Adım 1 / 10

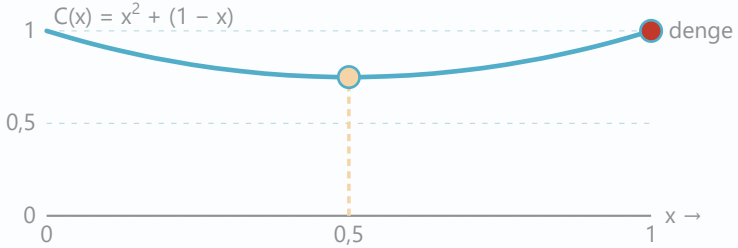
```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
    x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
3   opt = x ** (p + 1) + (1 - x)
4   return 1 / opt            # denge maliyeti 1
5
for p in [1, 2, 4, 8, 16]:
    print(p, round(pigou_poa(p), 3))
```

DEĞİŞKENLER

`p = 1` `x = 0.5`

ÇIKTI

Kod İzleme: `pigou_poa(p)`



p = 1

x^{OPT} = 0,5

C(OPT) = 0,75

PoA = 1 / C(OPT) = 1,333

C(OPT) = 0,25 + 0,5 = 0,75 ⇒ PoA = 4/3: Pigou'nun bilinen sonucu.

Adım 2 / 10

```
1 def pigou_poa(p):          # üst yol x**p, alt yol 1
2   x = (p + 1) ** (-1 / p)  # C'(x) = 0 noktası
▶   opt = x ** (p + 1) + (1 - x)
▶   return 1 / opt           # denge maliyeti 1
5
6 for p in [1, 2, 4, 8, 16]:
▶   print(p, round(pigou_poa(p), 3))
```

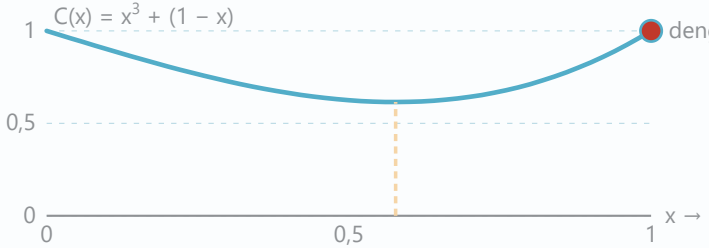
DEĞİŞKENLER

`p = 1` `opt = 0.75`

ÇIKTI

```
1 1.333
```

Kod İzleme: `pigou_poa(p)`



p = 2: $C(x) = x \cdot x^2 + (1 - x) \cdot 1$, $C'(x) = 3x^2 - 1 = 0 \Rightarrow x^{OPT} = 3^{-1/2} \approx 0{,}577$.

Adım 3 / 10

```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
▶   x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
3   opt = x ** (p + 1) + (1 - x)
4   return 1 / opt            # denge maliyeti 1
5
▶ for p in [1, 2, 4, 8, 16]:
▶   print(p, round(pigou_poa(p), 3))
```

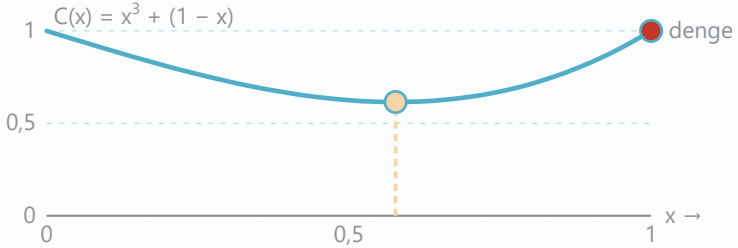
DEĞİŞKENLER

`p = 2` `x = 0.5774`

ÇIKTI

```
1 1.333
```

Kod İzleme: `pigou_poa(p)`



p = 2

x^{OPT} = 0,577

C(OPT) = 0,615

PoA = 1 / C(OPT) = 1,626

Eğri dikleşti: optimum 0,615'e indi, denge yine 1 ⇒ PoA 1,626.

Adım 4 / 10

```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
2   x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
▶   opt = x ** (p + 1) + (1 - x)
▶   return 1 / opt            # denge maliyeti 1
5
6 for p in [1, 2, 4, 8, 16]:
▶   print(p, round(pigou_poa(p), 3))
```

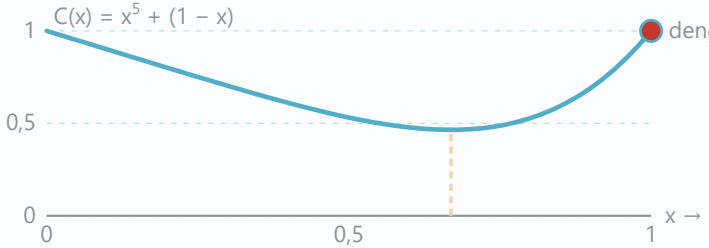
DEĞİŞKENLER

`p = 2` `opt = 0.6151`

ÇIKTI

```
1 1.333
2 1.626
```

Kod İzleme: `pigou_poa(p)`



p = 4

x^{OPT} = 0,669

p = 4: C(x) = x·x⁴ + (1 − x)·1, C′(x) = 5x⁴ − 1 = 0 ⇒ $x^{OPT} = 5^{-1/4} \approx 0{,}669$.

Adım 5 / 10

```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
    x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
3   opt = x ** (p + 1) + (1 - x)
4   return 1 / opt            # denge maliyeti 1
5
for p in [1, 2, 4, 8, 16]:
    print(p, round(pigou_poa(p), 3))
```

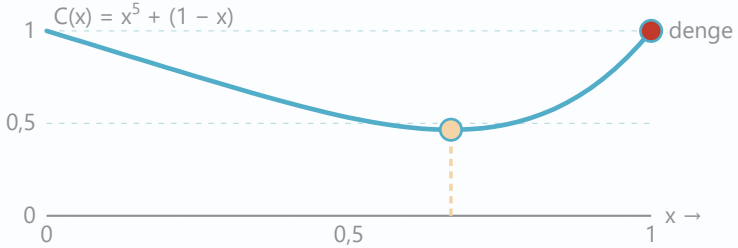
DEĞİŞKENLER

`p = 4` `x = 0.6687`

ÇIKTI

```
1 1.333
2 1.626
```

Kod İzleme: `pigou_poa(p)`



p = 4

x^{OPT} = 0,669

C(OPT) = 0,465

PoA = 1 / C(OPT) = 2,151

p = 4: optimumda trafiğin %67'si üstte; PoA 2,151 > 2.

Adım 6 / 10

```
1 def pigou_poa(p):          # üst yol x**p, alt yol 1
2   x = (p + 1) ** (-1 / p)  # C'(x) = 0 noktası
▶   opt = x ** (p + 1) + (1 - x)
▶   return 1 / opt           # denge maliyeti 1
5
6 for p in [1, 2, 4, 8, 16]:
▶   print(p, round(pigou_poa(p), 3))
```

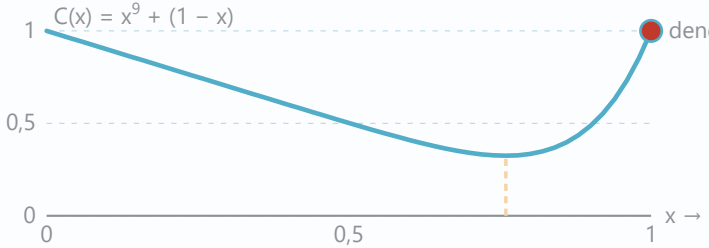
DEĞİŞKENLER

`p = 4` `opt = 0.465`

ÇIKTI

```
1 1.333
2 1.626
4 2.151
```

Kod İzleme: `pigou_poa(p)`



p = 8

$x^{OPT} = 0{,}76$

p = 8: $C(x) = x \cdot x^8 + (1 - x) \cdot 1$, $C'(x) = 9x^8 - 1 = 0 \Rightarrow x^{OPT} = 9^{-1/8} \approx 0{,}76$.

Adım 7 / 10

```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
    x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
3   opt = x ** (p + 1) + (1 - x)
4   return 1 / opt            # denge maliyeti 1
5
for p in [1, 2, 4, 8, 16]:
    print(p, round(pigou_poa(p), 3))
```

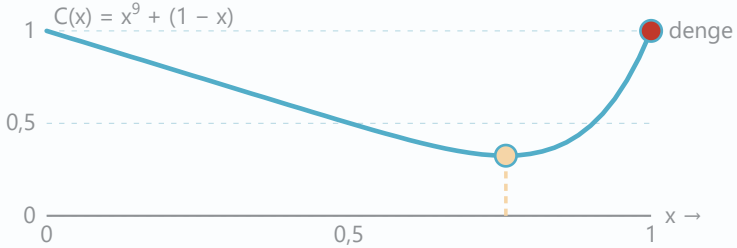
DEĞİŞKENLER

`p = 8` `x = 0.7598`

ÇIKTI

```
1 1.333
2 1.626
4 2.151
```

Kod İzleme: `pigou_poa(p)`



p = 8

x^{OPT} = 0,76

C(OPT) = 0,325

PoA = 1 / C(OPT) = 3,081

PoA 3,081: artış yavaş ama sınırsız.

Adım 8 / 10

```
1 def pigou_poa(p):          # üst yol x**p, alt yol 1
2   x = (p + 1) ** (-1 / p)  # C'(x) = 0 noktası
▶   opt = x ** (p + 1) + (1 - x)
▶   return 1 / opt           # denge maliyeti 1
5
6 for p in [1, 2, 4, 8, 16]:
▶   print(p, round(pigou_poa(p), 3))
```

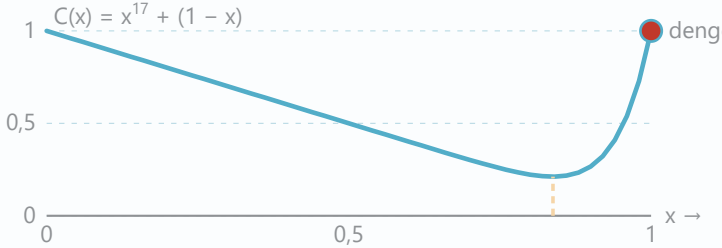
DEĞİŞKENLER

`p = 8` `opt = 0.3246`

ÇIKTI

```
1 1.333
2 1.626
4 2.151
8 3.081
```

Kod İzleme: `pigou_poa(p)`



p = 16

x^{OPT} = 0,838

p = 16: C(x) = x·x^{16} + (1 − x)·1, C′(x) = 17x^{16} − 1 = 0 ⇒ x^{OPT} = $17^{-1/16}$ ≈ 0,838.

Adım 9 / 10

```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
    x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
3   opt = x ** (p + 1) + (1 - x)
4   return 1 / opt            # denge maliyeti 1
5
for p in [1, 2, 4, 8, 16]:
    print(p, round(pigou_poa(p), 3))
```

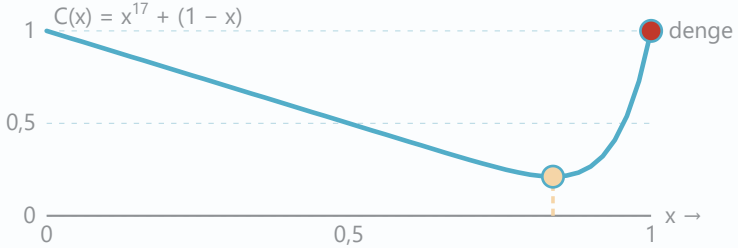
DEĞİŞKENLER

`p = 16` `x = 0.8377`

ÇIKTI

```
1 1.333
2 1.626
4 2.151
8 3.081
```

Kod İzleme: `pigou_poa(p)`



p = 16

x^{OPT} = 0,838

C(OPT) = 0,212

PoA = 1 / C(OPT) = 4,727

p = 16: PoA 4,727. 4/3 garantisi yalnız doğrusal gecikme içindir.

Adım 10 / 10

```
1 def pigou_poa(p):           # üst yol x**p, alt yol 1
2   x = (p + 1) ** (-1 / p)   # C'(x) = 0 noktası
▶   opt = x ** (p + 1) + (1 - x)
▶   return 1 / opt            # denge maliyeti 1
5
6 for p in [1, 2, 4, 8, 16]:
▶   print(p, round(pigou_poa(p), 3))
```

DEĞİŞKENLER

`p = 16` `opt = 0.2116`

ÇIKTI

```
1 1.333
2 1.626
4 2.151
8 3.081
16 4.727
```

Doğrusal Gecikmede 4/3 Üst Sınırı

Teorem (Roughgarden–Tardos, 2002). Gecikmeler doğrusal ($\ell_e(x) = a_e x + b_e$, $a, b \geq 0$) ise *her* ağda, *her* talepte: C(denge) ≤ **4/3** · C(optimum).

Doğrusal Gecikmede 4/3 Üst Sınırı

Teorem (Roughgarden–Tardos, 2002). Gecikmeler doğrusal ($\ell_e(x) = a_e x + b_e$, $a, b \geq 0$) ise *her* ağda, *her* talepte: $C(\text{denge}) \leq \mathbf{4/3} \cdot C(\text{optimum})$.

Gösterim. f_e: dengede e yolundaki trafik; f_e^{OPT}: optimumdaki trafik. Toplam gecikme, her yolda "trafik × gecikme" toplamıdır: $C(f) = \Sigma\, \ell_e(f_e) \cdot f_e$.

Doğrusal Gecikmede 4/3 Üst Sınırı

Teorem (Roughgarden–Tardos, 2002). Gecikmeler doğrusal ($\ell_e(x) = a_e x + b_e$, $a, b \geq 0$) ise *her* ağda, *her* talepte: C(denge) ≤ **4/3** · C(optimum).

Gösterim. f_e: dengede e yolundaki trafik; f_e^{OPT}: optimumdaki trafik. Toplam gecikme, her yolda "trafik × gecikme" toplamıdır: $C(f) = \Sigma\, \ell_e(f_e) \cdot f_e$.

1. Denge koşulu. Dengede herkes en kısa yolu kullanır. Gecikmeleri dondurup sürücüleri optimumdaki yollara taşısak kimse daha kısa yol bulamaz: $\Sigma\, \ell_e(f_e) \cdot f_e \leq \Sigma\, \ell_e(f_e) \cdot f_e^{OPT}$.

Doğrusal Gecikmede 4/3 Üst Sınırı

Teorem (Roughgarden–Tardos, 2002). Gecikmeler doğrusal ($\ell_e(x) = a_e x + b_e$, $a, b \geq 0$) ise *her* ağda, *her* talepte: $C(\text{denge}) \leq \mathbf{4/3} \cdot C(\text{optimum})$.

Gösterim. f_e: dengede e yolundaki trafik; f_e^{OPT}: optimumdaki trafik. Toplam gecikme, her yolda "trafik × gecikme" toplamıdır: $C(f) = \Sigma\, \ell_e(f_e) \cdot f_e$.

1. Denge koşulu. Dengede herkes en kısa yolu kullanır. Gecikmeleri dondurup sürücüleri optimumdaki yollara taşısak kimse daha kısa yol bulamaz: $\Sigma\, \ell_e(f_e) \cdot f_e \leq \Sigma\, \ell_e(f_e) \cdot f_e^{OPT}$.

Sol yan dengenin gerçek maliyeti C(f). Sağ yan bir düşünce deneyi: yollar dengedeki kadar yavaş, ama trafik optimumdaki gibi dağılmış. Geriye bu sağ yanı $C(f^{OPT})$ ile sınırlamak kalıyor.

4/3 Sınırı: Kenar Eşitsizliği ve Toplama

2. Kenar eşitsizliği. Her yol için: $\ell_e(f_e)\cdot f_e^{OPT} \le \ell_e(f_e^{OPT})\cdot f_e^{OPT} + \tfrac{1}{4}\cdot\ell_e(f_e)\cdot f_e$. Sözle: düşünce deneyindeki bedel, optimumun kendi bedelini en çok dengenin bedelinin dörtte biri kadar aşar.

4/3 Sınırı: Kenar Eşitsizliği ve Toplama

2. Kenar eşitsizliği. Her yol için: $\ell_e(f_e)\cdot f_e^{OPT} \le \ell_e(f_e^{OPT})\cdot f_e^{OPT} + \tfrac{1}{4}\cdot\ell_e(f_e)\cdot f_e$. Sözle: düşünce deneyindeki bedel, optimumun kendi bedelini en çok dengenin bedelinin dörtte biri kadar aşar.

Neden doğru? $\ell = ax + b$ yerine konur. b'li terimler: solda $b\cdot f^{OPT}$, sağda $b\cdot f^{OPT} + \tfrac{1}{4}\cdot b\cdot f$; sağ yan küçük değil. a'lı terimler: $a\cdot f\cdot f^{OPT} \le a\cdot(f^{OPT})^2 + a\cdot f^2/4$ gerekir. Her şey sağa toplanınca $a\cdot(f^{OPT} - f/2)^2 \ge 0$ çıkar: bir karenin negatif olmaması.

4/3 Sınırı: Kenar Eşitsizliği ve Toplama

2. Kenar eşitsizliği. Her yol için: $\ell_e(f_e)\cdot f_e^{OPT} \le \ell_e(f_e^{OPT})\cdot f_e^{OPT} + \tfrac{1}{4}\cdot\ell_e(f_e)\cdot f_e$. Sözle: düşünce deneyindeki bedel, optimumun kendi bedelini en çok dengenin bedelinin dörtte biri kadar aşar.

Neden doğru? $\ell = ax + b$ yerine konur. b'li terimler: solda $b\cdot f^{OPT}$, sağda $b\cdot f^{OPT} + \tfrac{1}{4}\cdot b\cdot f$; sağ yan küçük değil. a'lı terimler: $a\cdot f\cdot f^{OPT} \le a\cdot(f^{OPT})^2 + a\cdot f^2/4$ gerekir. Her şey sağa toplanınca $a\cdot(f^{OPT} - f/2)^2 \ge 0$ çıkar: bir karenin negatif olmaması.

Örnek (Pigou, $\ell = x$ yolu): $f = 1$, $f^{OPT} = \tfrac{1}{2}$. Sol: $1\cdot\tfrac{1}{2} = \tfrac{1}{2}$. Sağ: $\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot 1\cdot 1 = \tfrac{1}{2}$. İki yan eşit.

4/3 Sınırı: Kenar Eşitsizliği ve Toplama

2. Kenar eşitsizliği. Her yol için: $\ell_e(f_e)\cdot f_e^{OPT} \le \ell_e(f_e^{OPT})\cdot f_e^{OPT} + \tfrac{1}{4}\cdot\ell_e(f_e)\cdot f_e$. Sözle: düşünce deneyindeki bedel, optimumun kendi bedelini en çok dengenin bedelinin dörtte biri kadar aşar.

Neden doğru? $\ell = ax + b$ yerine konur. b'li terimler: solda $b\cdot f^{OPT}$, sağda $b\cdot f^{OPT} + \tfrac{1}{4}\cdot b\cdot f$; sağ yan küçük değil. a'lı terimler: $a\cdot f\cdot f^{OPT} \le a\cdot(f^{OPT})^2 + a\cdot f^2/4$ gerekir. Her şey sağa toplanınca $a\cdot(f^{OPT} - f/2)^2 \ge 0$ çıkar: bir karenin negatif olmaması.

Örnek (Pigou, $\ell = x$ yolu): $f = 1$, $f^{OPT} = \tfrac{1}{2}$. Sol: $1\cdot\tfrac{1}{2} = \tfrac{1}{2}$. Sağ: $\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot 1\cdot 1 = \tfrac{1}{2}$. İki yan eşit.

3. Topla. 2. adımı tüm yollar için topla ve 1. adımla birleştir: $C(f) \le C(f^{OPT}) + \tfrac{1}{4}\cdot C(f)$. $\tfrac{1}{4}\cdot C(f)$'yi sola geçir: $\tfrac{3}{4}\cdot C(f) \le C(f^{OPT}) \rightarrow$ **$C(f) / C(f^{OPT}) \le 4/3$**.

4/3 Sınırı: Kenar Eşitsizliği ve Toplama

2. Kenar eşitsizliği. Her yol için: $\ell_e(f_e)\cdot f_e^{OPT} \le \ell_e(f_e^{OPT})\cdot f_e^{OPT} + \tfrac{1}{4}\cdot\ell_e(f_e)\cdot f_e$. Sözle: düşünce deneyindeki bedel, optimumun kendi bedelini en çok dengenin bedelinin dörtte biri kadar aşar.

Neden doğru? $\ell = ax + b$ yerine konur. b'li terimler: solda $b\cdot f^{OPT}$, sağda $b\cdot f^{OPT} + \tfrac{1}{4}\cdot b\cdot f$; sağ yan küçük değil. a'lı terimler: $a\cdot f\cdot f^{OPT} \le a\cdot(f^{OPT})^2 + a\cdot f^2/4$ gerekir. Her şey sağa toplanınca $a\cdot(f^{OPT} - f/2)^2 \ge 0$ çıkar: bir karenin negatif olmaması.

Örnek (Pigou, $\ell = x$ yolu): $f = 1$, $f^{OPT} = \tfrac{1}{2}$. Sol: $1\cdot\tfrac{1}{2} = \tfrac{1}{2}$. Sağ: $\tfrac{1}{2}\cdot\tfrac{1}{2} + \tfrac{1}{4}\cdot 1\cdot 1 = \tfrac{1}{2}$. İki yan eşit.

3. Topla. 2. adımı tüm yollar için topla ve 1. adımla birleştir: $C(f) \le C(f^{OPT}) + \tfrac{1}{4}\cdot C(f)$. $\tfrac{1}{4}\cdot C(f)$'yi sola geçir: $\tfrac{3}{4}\cdot C(f) \le C(f^{OPT}) \rightarrow$ **$C(f) / C(f^{OPT}) \le 4/3$**.

Pigou'da eşitlik olur: sınır sıkıdır. Üstelik en kötü örnek her zaman Pigou gibi iki bağlantılıdır (Roughgarden, 2003): ağın karmaşıklığı PoA'yı büyütmez.

4/3 Sınırını Sınayalım

Örnek	Denge	Optimum	Bu örnekte gözlenen	Kuramsal üst sınır
Pigou (x ve 1)	1	3/4	**4/3** (sınıra eşit)	4/3
Braess (4.000 araç)	320.000	258.750	80 / 64,69 ≈ 1,237	4/3
Braess, normalleştirilmiş (x, 1, 1, x, 0)	2	3/2	**4/3** (sınıra eşit)	4/3
İki yol, 3.000 araç	75.000	71.250	1,053	4/3
20.000 rastgele doğrusal paralel ağ	—	—	en büyük 1,33	4/3
Yük Dengeleme Oyunu, Seviye 2 (toplam süre)	13	12,5	1,04	5/2 (atomik oyuncular)

Kuramsal üst sınır, bir sınıftaki (ör. doğrusal gecikmeli tüm ağlar) **en kötü durumdur**; belirli bir örnekte gözlenen oran çoğu zaman daha küçüktür. Betik 2–4 bağlantılı, rastgele a, b ve talep değerli 20.000 ağda dengeyi ve optimumu hesapladı: oran hiçbir örnekte 4/3'ü aşmadı, Pigou benzeri örneklerde 4/3'e yaklaştı.

Tek Bir Reçete: Pürüzsüzlük (Smoothness)

Düşünce deneyi. s bir durum (her oyuncunun bir stratejisi var), s^{OPT} optimum. Ötekiler s'de kalırken *yalnız* oyuncu i, s^{OPT}'deki stratejisine geçseydi ne öderdi? Bu, i'nin "deneme maliyeti"dir.

Tek Bir Reçete: Pürüzsüzlük (Smoothness)

Düşünce deneyi. s bir durum (her oyuncunun bir stratejisi var), s^{OPT} optimum. Ötekiler s'de kalırken *yalnız* oyuncu i, s^{OPT}'deki stratejisine geçseydi ne öderdi? Bu, i'nin "deneme maliyeti"dir.

Gösterim. s_{-i}: i dışındaki oyuncuların stratejileri. $C_i(s_i^{OPT}, s_{-i})$: i'nin deneme maliyeti. $C(s) = \Sigma_i C_i(s)$: toplam maliyet.

Tek Bir Reçete: Pürüzsüzlük (Smoothness)

Düşünce deneyi. s bir durum (her oyuncunun bir stratejisi var), s^{OPT} optimum. Ötekiler s'de kalırken *yalnız* oyuncu i, s^{OPT}'deki stratejisine geçseydi ne öderdi? Bu, i'nin "deneme maliyeti"dir.

Gösterim. s_{-i}: i dışındaki oyuncuların stratejileri. $C_i(s_i^{OPT}, s_{-i})$: i'nin deneme maliyeti. $C(s) = \Sigma_i C_i(s)$: toplam maliyet.

Tanım (Roughgarden, 2009). $\lambda > 0$ ve $0 \le \mu < 1$ iki sabit olsun. Her s ve s^{OPT} için $\Sigma_i C_i(s_i^{OPT}, s_{-i}) \le \lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$ ise oyun **(λ, μ)-pürüzsüz**dür.

Tek Bir Reçete: Pürüzsüzlük (Smoothness)

Düşünce deneyi. s bir durum (her oyuncunun bir stratejisi var), s^{OPT} optimum. Ötekiler s'de kalırken *yalnız* oyuncu i, s^{OPT}'deki stratejisine geçseydi ne öderdi? Bu, i'nin "deneme maliyeti"dir.

Gösterim. s_{-i}: i dışındaki oyuncuların stratejileri. $C_i(s_i^{OPT}, s_{-i})$: i'nin deneme maliyeti. $C(s) = \Sigma_i C_i(s)$: toplam maliyet.

Tanım (Roughgarden, 2009). $\lambda > 0$ ve $0 \le \mu < 1$ iki sabit olsun. Her s ve s^{OPT} için $\Sigma_i C_i(s_i^{OPT}, s_{-i}) \le \lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$ ise oyun **(λ, μ)-pürüzsüz**dür.

Sözle: deneme maliyetlerinin toplamı, optimumun maliyetinin λ katı ile s'nin maliyetinin μ katının toplamını aşmaz. λ ve μ küçüldükçe elde edilen sınır iyileşir.

Pürüzsüzlükten PoA Sınırına: Dört Adım

1. Denge. s bir Nash dengesi: kimse tek başına sapmakla kazanmaz. Her i için $C_i(s) \le C_i(s_i^{OPT}, s_{-i})$.

Pürüzsüzlükten PoA Sınırına: Dört Adım

1. Denge. s bir Nash dengesi: kimse tek başına sapmakla kazanmaz. Her i için $C_i(s) \le C_i(s_i^{OPT}, s_{-i})$.

2. Topla. Bu eşitsizlikleri tüm oyuncular için topla: $C(s) \le \Sigma_i\, C_i(s_i^{OPT}, s_{-i})$.

Pürüzsüzlükten PoA Sınırına: Dört Adım

1. Denge. s bir Nash dengesi: kimse tek başına sapmakla kazanmaz. Her i için $C_i(s) \le C_i(s_i^{OPT}, s_{-i})$.

2. Topla. Bu eşitsizlikleri tüm oyuncular için topla: $C(s) \le \Sigma_i\, C_i(s_i^{OPT}, s_{-i})$.

3. Pürüzsüzlüğü uygula. Sağ yan en çok $\lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$ olduğundan $C(s) \le \lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$.

Pürüzsüzlükten PoA Sınırına: Dört Adım

1. Denge. s bir Nash dengesi: kimse tek başına sapmakla kazanmaz. Her i için $C_i(s) \le C_i(s_i^{OPT}, s_{-i})$.

2. Topla. Bu eşitsizlikleri tüm oyuncular için topla: $C(s) \le \Sigma_i\, C_i(s_i^{OPT}, s_{-i})$.

3. Pürüzsüzlüğü uygula. Sağ yan en çok $\lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$ olduğundan $C(s) \le \lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$.

4. Düzenle. $\mu \cdot C(s)$'yi sola geçir: $(1 - \mu) \cdot C(s) \le \lambda \cdot C(s^{OPT})$. İki yanı $(1 - \mu) \cdot C(s^{OPT})$'ye böl: $C(s) / C(s^{OPT}) \le \lambda / (1 - \mu)$. s herhangi bir denge olduğundan **PoA ≤ λ / (1 − μ)**.

Pürüzsüzlükten PoA Sınırına: Dört Adım

1. Denge. s bir Nash dengesi: kimse tek başına sapmakla kazanmaz. Her i için $C_i(s) \le C_i(s_i^{OPT}, s_{-i})$.

2. Topla. Bu eşitsizlikleri tüm oyuncular için topla: $C(s) \le \sum_i C_i(s_i^{OPT}, s_{-i})$.

3. Pürüzsüzlüğü uygula. Sağ yan en çok $\lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$ olduğundan $C(s) \le \lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$.

4. Düzenle. $\mu \cdot C(s)$'yi sola geçir: $(1 - \mu) \cdot C(s) \le \lambda \cdot C(s^{OPT})$. İki yanı $(1 - \mu) \cdot C(s^{OPT})$'ye böl: $C(s) / C(s^{OPT}) \le \lambda / (1 - \mu)$. s herhangi bir denge olduğundan **PoA ≤ λ / (1 − μ)**.

Tanıdık örnekler. 4/3 ispatının kenar eşitsizliği $\lambda = 1$, $\mu = ¼$ demektir: $1 / (1 - ¼) =$ **4/3**. Atomik (bölünemez oyunculu) doğrusal tıkanıklık oyunlarında $\lambda = 5/3$, $\mu = 1/3$: $(5/3) / (2/3) =$ **5/2**.

Pürüzsüzlükten PoA Sınırına: Dört Adım

1. Denge. s bir Nash dengesi: kimse tek başına sapmakla kazanmaz. Her i için $C_i(s) \le C_i(s_i^{OPT}, s_{-i})$.

2. Topla. Bu eşitsizlikleri tüm oyuncular için topla: $C(s) \le \sum_i C_i(s_i^{OPT}, s_{-i})$.

3. Pürüzsüzlüğü uygula. Sağ yan en çok $\lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$ olduğundan $C(s) \le \lambda \cdot C(s^{OPT}) + \mu \cdot C(s)$.

4. Düzenle. $\mu \cdot C(s)$'yi sola geçir: $(1 - \mu) \cdot C(s) \le \lambda \cdot C(s^{OPT})$. İki yanı $(1 - \mu) \cdot C(s^{OPT})$'ye böl: $C(s) / C(s^{OPT}) \le \lambda / (1 - \mu)$. s herhangi bir denge olduğundan **PoA ≤ λ / (1 − μ)**.

Tanıdık örnekler. 4/3 ispatının kenar eşitsizliği $\lambda = 1$, $\mu = ¼$ demektir: $1 / (1 - ¼) =$ **4/3**. Atomik (bölünemez oyunculu) doğrusal tıkanıklık oyunlarında $\lambda = 5/3$, $\mu = 1/3$: $(5/3) / (2/3) =$ **5/2**.

Neden önemli? İspat yalnız “kimse tek başına sapmaz” koşulunu kullanır; bu yüzden aynı sınır daha gevşek denge türlerinde de (ör. karışık dengelerde) geçerlidir: **sağlam PoA** (robust PoA).

Atomik Tıkanıklık Oyunları: PoA ≤ 5/2

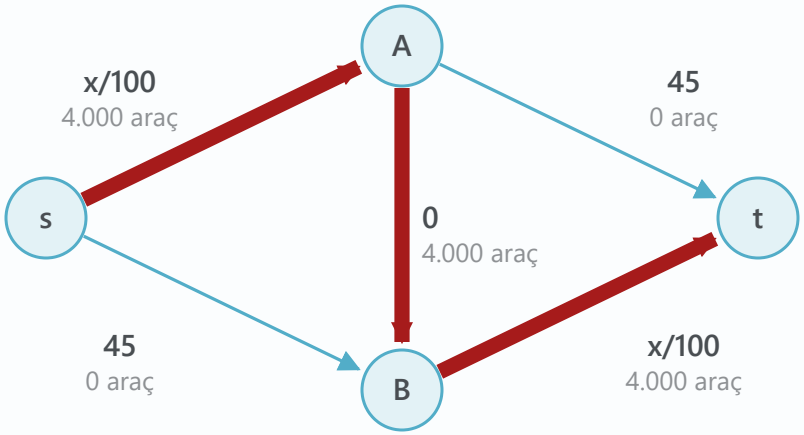
y (s^OPT) \ z (dengedeki yük)	0	1	2	3	4	5
0	**0**	1	4	9	16	25
1	2	**0**	**0**	2	6	12
2	14	9	6	5	6	9
3	36	28	22	18	16	16
4	68	57	48	41	36	33
5	110	96	84	74	66	60

Hücre: 3 × (sağ yan − sol yan), kesirsiz olsun diye. Negatif hücre yok: eşitsizlik doğru (betik y, z ≤ 60 denedi); turuncu 0'larda (ör. y = 1, z = 1 ya da 2) iki yan eşit.

- Her oyuncu bir **birim** yük; kenar maliyeti $c_e(x) = a_e x + b_e$.
- z_e: dengede e kenarını kullanan kişi sayısı; y_e: optimumda kullanan kişi sayısı. Tek başına optimumdaki yoluna geçen oyuncu e'de en çok $z_e + 1$ kişi bulur.
- Anahtar eşitsizlik (tamsayılar): $\mathbf{y(z + 1) \le 5/3 \cdot y^2 + 1/3 \cdot z^2}$. c(x) = x iken sol yan e'deki deneme maliyetleri; y^2 ve z^2 e'nin optimum ve denge maliyeti.
- ⇒ (5/3, 1/3)-pürüzsüz ⇒ PoA ≤ (5/3)/(2/3) = **5/2**; sınır sıkıdır (Awerbuch–Azar–Epstein; Christodoulou–Koutsoupias, 2005).

Bölünebilir trafikte 4/3, bölünemez oyunculara 5/2: oyuncunun kendi eklediği gecikme (z + 1'deki "+1") kaybı büyütür.

Adım Adım: Braess Ağında PoA



Yeni yollu ağda denge

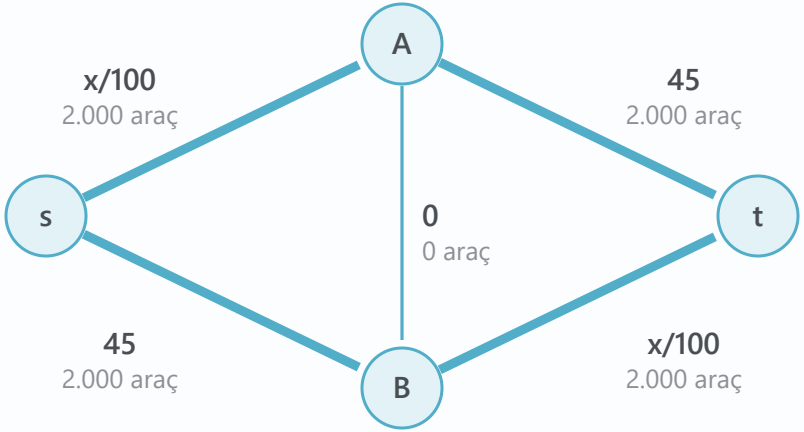
herkes s→A→B→t: 80 dk

C(denge) = 4.000 · 80

= 320.000 araç·dk

A→B yolu açılınca herkes zikzağa geçer, kişi başı 80 dk. Toplam 320.000 araç·dk.

Adım Adım: Braess Ağında PoA



Yeni yol öncesi: 65 dk

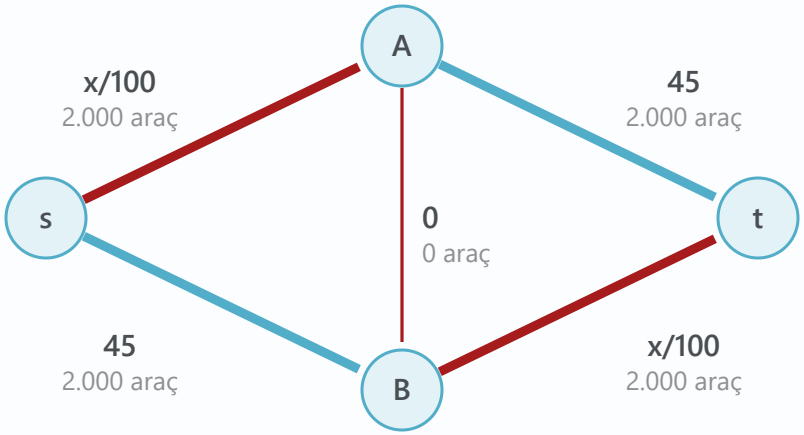
80 / 65 ≈ 1,231

Bu “Braess kaybı”dır;

PoA değil: payda başka ağ

Sık yapılan kıyas 80/65 ≈ 1,23. Ama PoA'nın paydası **aynı ağın** optimumudur; yeni ağda 65'ten iyisi olabilir mi?

Adım Adım: Braess Ağında PoA



z = 0'da marjinal maliyetler:

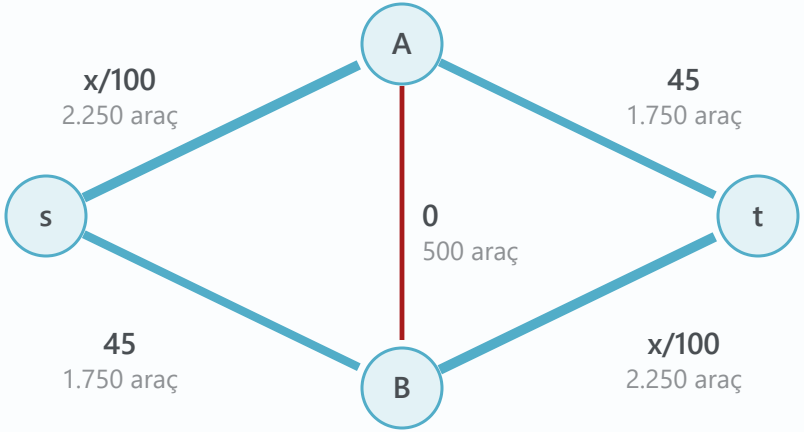
zikzak: 2·20 + 2·20 = 80

eski yol: 2·20 + 45 = 85

→ biraz zikzak iyi!

s→A'da 2.000 araçla gecikme 20 dk; marjinal maliyet $\ell + x \cdot \ell' = 20 + 2.000 \cdot (1/100) = 40$ dk. Zikzak bu türden iki kenar kullanır: 40 + 0 + 40 = 80; eski yol 40 + 45 = 85. 80 < 85: planlayıcı bile birkaç aracı zikzağa yollar.

Adım Adım: Braess Ağında PoA



Simetrik: a araç her eski yolda

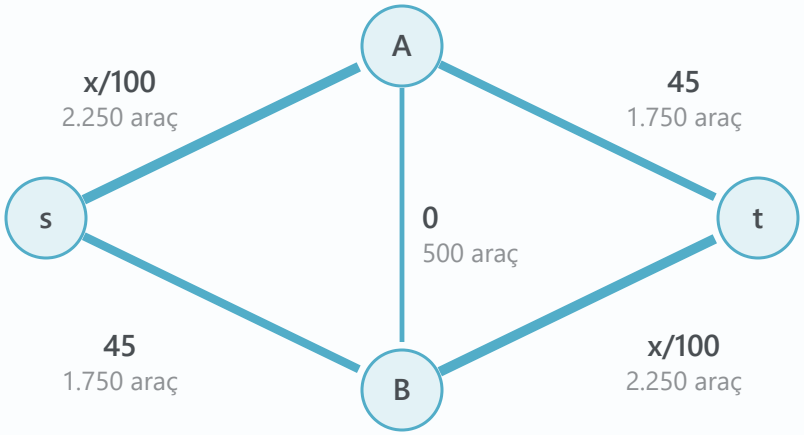
$C(a) = 2(4000 - a)^2/100 + 90a$

$C' = 0 \rightarrow a = 1.750, z = 500$

$C(opt) = 258.750$

Optimum: her eski yolda 1.750, zikzakta 500 araç (marjinal maliyetler 90'da eşit). Ortalama ≈ 64,69 dk; betik tüm dağılımları taradı.

Adım Adım: Braess Ağında PoA



PoA = 320.000 / 258.750

≈ 1,237

≤ 4/3 ✓

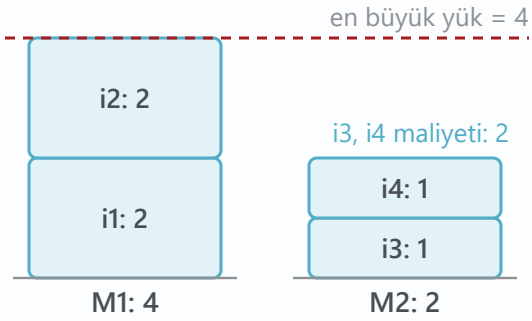
normalleştirilmiş: tam 4/3

PoA = 320.000 / 258.750 ≈ 1,237 ≤ 4/3. Klasik normalleştirilmiş Braess ağında (x, 1, 1, x, 0) oran tam 2 / (3/2) = 4/3.

YÜK DENGELEME

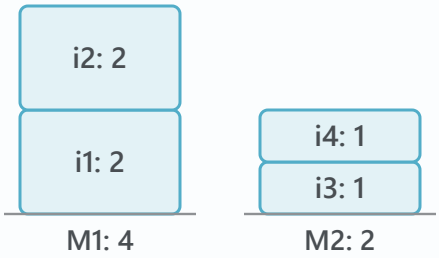
Yük Dengeleme (Load Balancing) Oyunu

- n iş (oyuncu), m özdeş makine; iş j'nin boyu w_j.
- Her iş bir makine seçer; **maliyeti** o makinenin toplam yüküdür (bitiş süresi).
- Sosyal maliyet: **en büyük yük (makespan)**, yani tüm işlerin bitiş anı.
- Uygulama: sunucu kümeleri, bulut görev yerleştirme, işlemci çekirdekleri. Koutsoupias–Papadimitriou PoA'yı ilk kez bu modelde (1999) tanımladı.



Bu oyunda saf Nash dengesi her zaman vardır: bir iş daha az yüklü makineye geçtiğinde, yükler büyükten küçüğe sıralanınca liste “küçülür” (ilk farklı yerde daha küçük sayı gelir). Bu sonsuza dek süremez; süreç bir Nash dengesinde durur.

Adım Adım: Yük Dengeleme Oyununda PoA



4 iş: boylar 2, 2, 1, 1

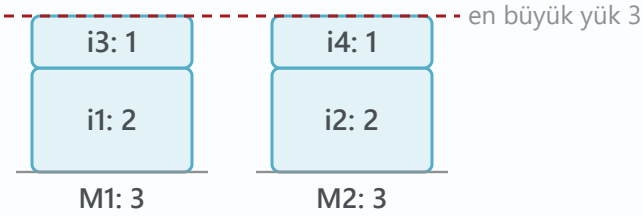
2 özdeş makine

İşin maliyeti: makinesinin yükü

Sosyal maliyet: en büyük yük

Yük dengeleme oyunu: her iş (oyuncu) bir makine seçer; kendi bitiş süresini, yani makinesinin toplam yükünü küçültmek ister.

Adım Adım: Yük Dengeleme Oyununda PoA



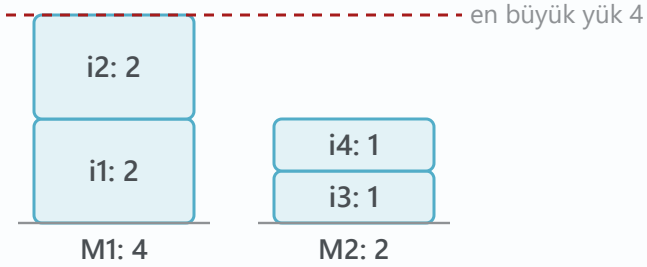
Optimum: {2, 1} | {2, 1}

yükler 3 ve 3

en büyük yük = 3

Sosyal optimum: toplam iş 6, iki makine → en iyi olasılık 3 ve 3. En büyük yük 3.

Adım Adım: Yük Dengeleme Oyununda PoA



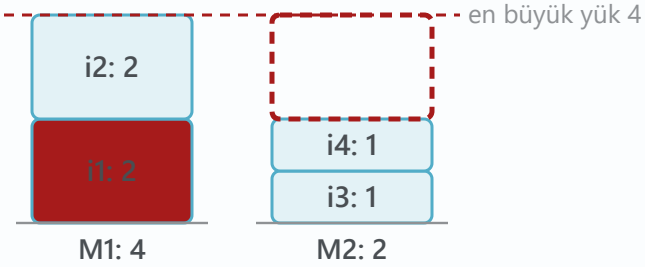
Atama: {2, 2} | {1, 1}

yükler 4 ve 2

en büyük yük = 4

Başka bir atama: iki büyük iş M1'de (yük 4), iki küçük iş M2'de (yük 2). Bu bir denge olabilir mi?

Adım Adım: Yük Dengeleme Oyununda PoA



i1 (2) M2'ye geçerse:

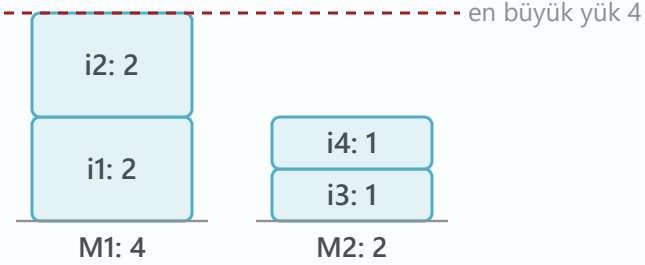
2 + 2 = 4, şimdi 4

→ kazanmaz

i3 (1) M1'e geçerse: 4 + 1 = 5 > 2

Sapma kontrolü: büyük iş M2'ye geçse yine 4 bekler; küçük iş M1'e geçse 5 bekler. **Kimse kazanmaz → Nash dengesi.**

Adım Adım: Yük Dengeleme Oyununda PoA



En kötü denge: 4

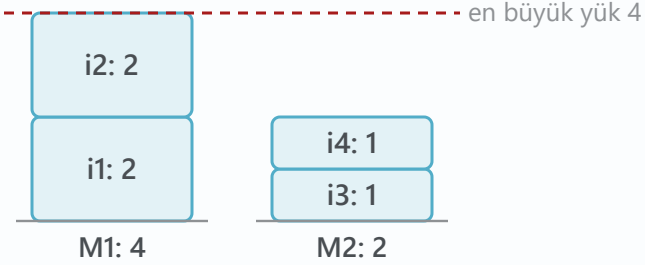
En iyi denge: 3 (optimum da denge)

PoA = 4/3, PoS = 1

Betik: 16 atamanın 6'sı denge

Bu örnekte **PoA = 4/3** ve PoS = 1: optimum da bir denge, ama kötü bir denge de var.

Adım Adım: Yük Dengeleme Oyununda PoA



Özdeş m makine, saf dengeler:

PoA = 2 − 2/(m + 1)

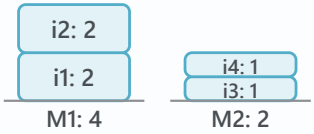
m = 2 → 4/3 (bu örnek sıkı)

Karışık dengeler: m ile çok yavaş büyür

Genel sonuç (Finn–Horowitz, 1979; Koutsoupias–Papadimitriou, 1999): saf dengede PoA = 2 − 2/(m+1) < 2. Karışık dengede PoA m ile çok yavaş ama sınırsız büyür (Koutsoupias–Mavronicolas–Spirakis, 2003; Czumaj–Vöcking, 2002).

Kod İzleme: `nash_mi(boy, atama, m)`

i1, i2 → M1; i3, i4 → M2



Çağrı: iş boyları ve her işin makinesi (0 = M1, 1 = M2).

Adım 1 / 11

```
▶ def nash_mi(boy, atama, m):
2   yuk = [0] * m
3   for j, k in enumerate(atama):
4     yuk[k] += boy[j]
5   for j, k in enumerate(atama):
6     for h in range(m):
7       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

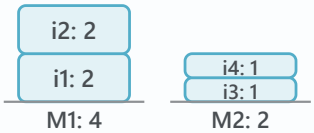
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

yuk = [0, 0]



Yük dizisi sıfırlanır.

Adım 2 / 11

```
1 def nash_mi(boy, atama, m):
▶   yuk = [0] * m
3   for j, k in enumerate(atama):
4     yuk[k] += boy[j]
5   for j, k in enumerate(atama):
6     for h in range(m):
7       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

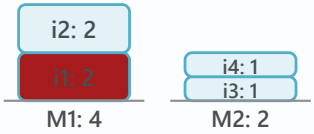
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [0, 0]`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

yuk = [2, 0]



j = 0: i1 (boy 2) M1'e eklenir → yuk = [2, 0].


```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
▶   for j, k in enumerate(atama):
▶     yuk[k] += boy[j]
5   for j, k in enumerate(atama):
6     for h in range(m):
7       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

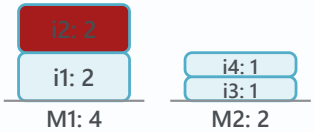
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [2, 0]` `j = 0` `k = 0`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

yuk = [4, 0]



j = 1: i2 (boy 2) M1'e eklenir → yuk = [4, 0].

Adım 4 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
▶   for j, k in enumerate(atama):
▶     yuk[k] += boy[j]
5   for j, k in enumerate(atama):
6     for h in range(m):
7       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

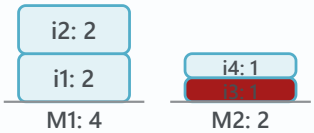
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 0]` `j = 1` `k = 0`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

yuk = [4, 1]



j = 2: i3 (boy 1) M2'ye eklenir → yuk = [4, 1].

Adım 5 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
▶   for j, k in enumerate(atama):
▶     yuk[k] += boy[j]
5   for j, k in enumerate(atama):
6     for h in range(m):
7       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

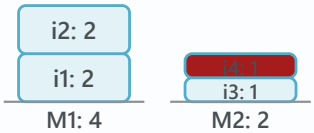
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 1]` `j = 2` `k = 1`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

yuk = [4, 2]



j = 3: i4 (boy 1) M2'ye eklenir → yuk = [4, 2].

Adım 6 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
▶   for j, k in enumerate(atama):
▶     yuk[k] += boy[j]
5   for j, k in enumerate(atama):
6     for h in range(m):
7       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

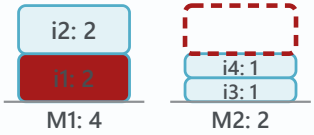
DEĞİŞKENLER

`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 2]` `j = 3` `k = 1`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`



yuk[1] + boy[0] = 4
yuk[0] = 4 → kaçmaz

j = 0, h = 1: i1 M2'ye geçerse 2 + 2 = 4; şimdiki 4. 4 < 4? **Hayır**.

Adım 7 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
3   for j, k in enumerate(atama):
4     yuk[k] += boy[j]
▶   for j, k in enumerate(atama):
▶     for h in range(m):
▶       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

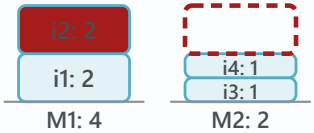
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 2]` `j = 0` `k = 0` `h = 1`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

yuk[1] + boy[1] = 4
yuk[0] = 4 → kaçmaz



j = 1, h = 1: i2 M2'ye geçerse 2 + 2 = 4; şimdiki 4. 4 < 4? **Hayır**.

Adım 8 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
3   for j, k in enumerate(atama):
4     yuk[k] += boy[j]
▶   for j, k in enumerate(atama):
▶     for h in range(m):
▶       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

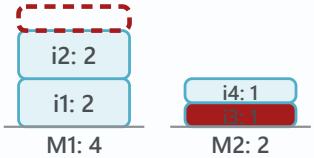
DEĞİŞKENLER

`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 2]` `j = 1` `k = 0` `h = 1`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`



yuk[0] + boy[2] = 5
yuk[1] = 2 → kaçmaz

j = 2, h = 0: i3 M1'e geçerse 4 + 1 = 5; şimdiki 2. 5 < 2? **Hayır**.

Adım 9 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
3   for j, k in enumerate(atama):
4     yuk[k] += boy[j]
▶   for j, k in enumerate(atama):
▶     for h in range(m):
▶       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

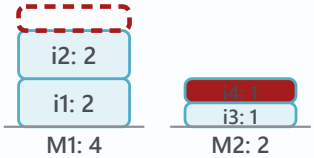
DEĞİŞKENLER

`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 2]` `j = 2` `k = 1` `h = 0`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`



yuk[0] + boy[3] = 5
yuk[1] = 2 → kaçmaz

j = 3, h = 0: i4 M1'e geçerse 4 + 1 = 5; şimdiki 2. 5 < 2? **Hayır**.

Adım 10 / 11

```
1 def nash_mi(boy, atama, m):
2   yuk = [0] * m
3   for j, k in enumerate(atama):
4     yuk[k] += boy[j]
▶   for j, k in enumerate(atama):
▶     for h in range(m):
▶       if yuk[h] + boy[j] < yuk[k]:
8         return False      # j kaçar
9   return True             # kimse kaçmaz
```

DEĞİŞKENLER

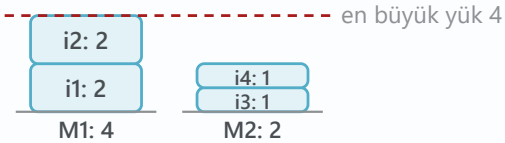
`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 2]` `j = 3` `k = 1` `h = 0`

ÇIKTI

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

Kod İzleme: `nash_mi(boy, atama, m)`

nash_mi → True



Hiçbir iş kârlı bir makineye kaçamadı → **True**: {2, 2} | {1, 1} bir Nash dengesidir (en büyük yük 4).

Adım 11 / 11

```
1  def nash_mi(boy, atama, m):
2    yuk = [0] * m
3    for j, k in enumerate(atama):
4      yuk[k] += boy[j]
5    for j, k in enumerate(atama):
6      for h in range(m):
7        if yuk[h] + boy[j] < yuk[k]:
8          return False      # j kaçar
▶    return True             # kimse kaçmaz
```

DEĞİŞKENLER

`boy = [2, 2, 1, 1]` `atama = [0, 0, 1, 1]` `m = 2` `yuk = [4, 2]`

ÇIKTI

```
True
```

h = k (kendi makinesi) için koşul hep yanlıştır; adımlarda yalnız h ≠ k gösterildi.

YÜK DENGELEME

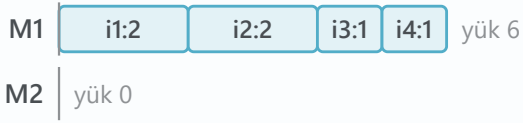
Kaba Kuvvet: 16 Atamanın Hepsi

Bölüşüm (M1 | M2, simetrik)	Atama sayısı	En büyük yük	Nash?
{2, 1} | {2, 1}	4	3	✓ denge
{2, 1, 1} | {2}	4	4	✗
{2, 2} | {1, 1}	2	4	✓ denge
{2, 2, 1} | {1}	4	5	✗
{2, 2, 1, 1} | { }	2	6	✗

Betik $2^4 = 16$ atamanın hepsini tarayıp her birinde her işin sapmasını denedi. Dengeler en büyük yükü 3 ve 4 olanlardır → **PoA = 4/3** (en kötü denge), **PoS = 1** (en iyi denge = optimum).

En iyi tepki dinamiği bu oyunda her zaman bir dengede durur: her kârlı geçiş, sıralanmış yük listesini küçültür. Varılan denge başlangıca ve hamle sırasına bağlıdır.

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



1/16 atama = (0, 0, 0, 0)

en büyük yük 6 · Nash? ✗

opt = 6 · dengeler = []

(0, 0, 0, 0): hepsi M1'de, yükler [6, 0]; i1 M2'ye geçince 2 < 6 ⇒ denge değil. opt şimdilik 6.

Adım 1 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

`atama = (0, 0, 0, 0)`

`yuk = [6, 0]` `opt = 6`

`dengeler = []`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



2/16 atama = (0, 0, 0, 1)

en büyük yük 5 · Nash? ✗

opt = 5 · dengeler = []

(0, 0, 0, 1): yükler [5, 1]; i1 öteki makineye geçince 3 < 5 ⇒ denge değil.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

`atama = (0, 0, 0, 1)`

`yuk = [5, 1]` `opt = 5`

`dengeler = []`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



3/16 atama = (0, 0, 1, 0)

en büyük yük 5 · Nash? ✗

opt = 5 · dengeler = []

(0, 0, 1, 0): yükler [5, 1]; i1 öteki makineye geçince 3 < 5 ⇒ denge değil.

Adım 3 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

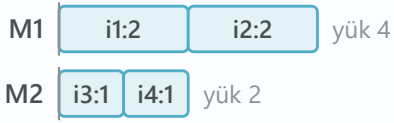
`atama = (0, 0, 1, 0)`

`yuk = [5, 1]` `opt = 5`

`dengeler = []`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



4/16 atama = (0, 0, 1, 1)

en büyük yük 4 · Nash? ✓

opt = 4 · dengeler = [4]

(0, 0, 1, 1): yükler [4, 2], kimse kaçamaz ⇒ **denge**, en büyük yük 4 listeye eklenir.

Adım 4 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
▶        dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

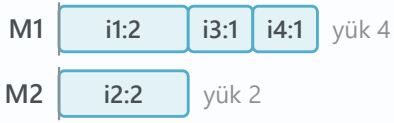
`atama = (0, 0, 1, 1)`

`yuk = [4, 2]` `opt = 4`

`dengeler = [4]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



5/16 atama = (0, 1, 0, 0)

en büyük yük 4 · Nash? ✗

opt = 4 · dengeler = [4]

(0, 1, 0, 0): yükler [4, 2]; i3 öteki makineye geçince 3 < 4 ⇒ denge değil.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

atama = (0, 1, 0, 0)

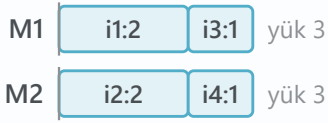
yuk = [4, 2]

opt = 4

dengeler = [4]

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



6/16 atama = (0, 1, 0, 1)

en büyük yük 3 · Nash? ✓

opt = 3 · dengeler = [4, 3]

(0, 1, 0, 1): yükler [3, 3], kimse kaçamaz ⇒ **denge**, en büyük yük 3 listeye eklenir.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
▶        dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

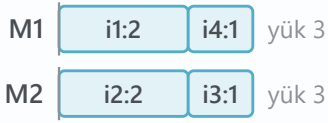
`atama = (0, 1, 0, 1)`

`yuk = [3, 3]` `opt = 3`

`dengeler = [4, 3]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



7/16 atama = (0, 1, 1, 0)

en büyük yük 3 · Nash? ✓

opt = 3 · dengeler = [4, 3, 3]

(0, 1, 1, 0): yükler [3, 3], kimse kaçamaz ⇒ **denge**, en büyük yük 3 listeye eklenir.

Adım 7 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
▶        dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

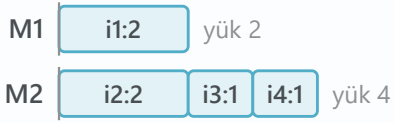
`atama = (0, 1, 1, 0)`

`yuk = [3, 3]` `opt = 3`

`dengeler = [4, 3, 3]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



8/16 atama = (0, 1, 1, 1)

en büyük yük 4 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3]

(0, 1, 1, 1): yükler [2, 4]; i3 öteki makineye geçince 3 < 4 ⇒ denge değil.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

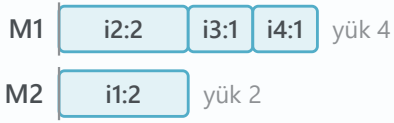
atama = (0, 1, 1, 1)

yuk = [2, 4] opt = 3

dengeler = [4, 3, 3]

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



9/16 atama = (1, 0, 0, 0)

en büyük yük 4 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3]

(1, 0, 0, 0): yükler [4, 2]; i3 öteki makineye geçince 3 < 4 ⇒ denge değil.

Adım 9 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

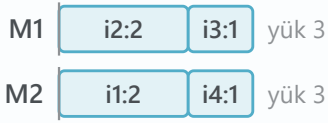
atama = (1, 0, 0, 0)

yuk = [4, 2] opt = 3

dengeler = [4, 3, 3]

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



10/16 atama = (1, 0, 0, 1)

en büyük yük 3 · Nash? ✓

opt = 3 · dengeler = [4, 3, 3, 3]

(1, 0, 0, 1): yükler [3, 3], kimse kaçamaz ⇒ **denge**, en büyük yük 3 listeye eklenir.

Adım 10 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
▶        dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

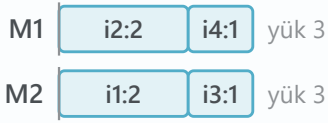
`atama = (1, 0, 0, 1)`

`yuk = [3, 3]` `opt = 3`

`dengeler = [4, 3, 3, 3]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



11/16 atama = (1, 0, 1, 0)

en büyük yük 3 · Nash? ✓

opt = 3 · dengeler = [4, 3, 3, 3, 3]

(1, 0, 1, 0): yükler [3, 3], kimse kaçamaz ⇒ **denge**, en büyük yük 3 listeye eklenir.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
▶        dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

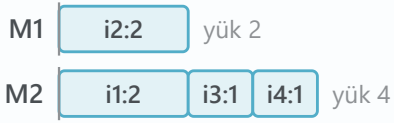
`atama = (1, 0, 1, 0)`

`yuk = [3, 3]` `opt = 3`

`dengeler = [4, 3, 3, 3, 3]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



12/16 atama = (1, 0, 1, 1)

en büyük yük 4 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3, 3, 3]

(1, 0, 1, 1): yükler [2, 4]; i3 öteki makineye geçince 3 < 4 ⇒ denge değil.

Adım 12 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

atama = (1, 0, 1, 1)

yuk = [2, 4] opt = 3

dengeler = [4, 3, 3, 3, 3]

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



13/16 atama = (1, 1, 0, 0)

en büyük yük 4 · Nash? ✓

opt = 3 · dengeler = [4, 3, 3, 3, 3, 4]

(1, 1, 0, 0): yükler [2, 4], kimse kaçamaz ⇒ **denge**, en büyük yük 4 listeye eklenir.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
▶        dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

`atama = (1, 1, 0, 0)`

`yuk = [2, 4]` `opt = 3`

`dengeler = [4, 3, 3, 3, 3, 4]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



14/16 atama = (1, 1, 0, 1)

en büyük yük 5 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3, 3, 3, 4]

(1, 1, 0, 1): yükler [1, 5]; i1 öteki makineye geçince 3 < 5 ⇒ denge değil.

Adım 14 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

atama = (1, 1, 0, 1)

yuk = [1, 5] opt = 3

dengeler = [4, 3, 3, 3, 3, 4]

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



15/16 atama = (1, 1, 1, 0)

en büyük yük 5 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3, 3, 3, 4]

(1, 1, 1, 0): yükler [1, 5]; i1 öteki makineye geçince 3 < 5 ⇒ denge değil.


```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

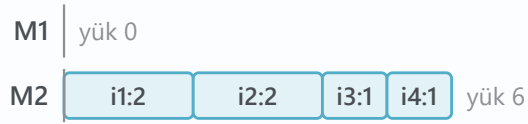
`atama = (1, 1, 1, 0)`

`yuk = [1, 5]` `opt = 3`

`dengeler = [4, 3, 3, 3, 3, 4]`

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



16/16 atama = (1, 1, 1, 1)

en büyük yük 6 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3, 3, 3, 4]

(1, 1, 1, 1): yükler [0, 6]; i1 öteki makineye geçince 2 < 6 ⇒ denge değil.

Adım 16 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
▶    for atama in product(range(m), repeat=len(boy)):
▶      yuk = [0] * m
▶      for j, k in enumerate(atama):
▶        yuk[k] += boy[j]
▶      opt = min(opt, max(yuk))
▶      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
11   return max(dengeler) / opt, min(dengeler) / opt
12 r = poa_pos([2, 2, 1, 1], 2)
13 print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

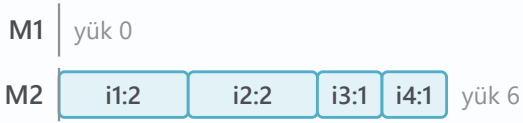
atama = (1, 1, 1, 1)

yuk = [0, 6] opt = 3

dengeler = [4, 3, 3, 3, 3, 4]

ÇIKTI

Kod İzleme: Tüm Atamaları Tarayarak PoA ve PoS



16/16 atama = (1, 1, 1, 1)

en büyük yük 6 · Nash? ✗

opt = 3 · dengeler = [4, 3, 3, 3, 3, 4]

Dengeler [4, 3, 3, 3, 3, 4] ⇒ PoA = 4/3, PoS = 3/3 = 1. m^n atama: büyük oyunlarda üstel.

Adım 17 / 17

```
1  from itertools import product
2  def poa_pos(boy, m):
3    opt, dengeler = float("inf"), []
4    for atama in product(range(m), repeat=len(boy)):
5      yuk = [0] * m
6      for j, k in enumerate(atama):
7        yuk[k] += boy[j]
8      opt = min(opt, max(yuk))
9      if nash_mi(boy, atama, m):       # denge testi
10       dengeler.append(max(yuk))
▶    return max(dengeler) / opt, min(dengeler) / opt
▶  r = poa_pos([2, 2, 1, 1], 2)
▶  print(round(r[0], 3), r[1])
```

DEĞİŞKENLER

`opt = 3`

`dengeler = [4, 3, 3, 3, 3, 4]`

ÇIKTI

```
1.333 1.0
```

Farklı Hızlı (İlişkili) Makineler

- Makine i'nin hızı s_i; bir işin bitiş süresi, makinesinin toplam yükünün s_i'ye bölümüdür. Saf Nash dengesi yine her zaman vardır.
- Örnek: hızlar A = 2, B = 1, C = 1; iş boyları 2, 1, 1. Sosyal maliyet: en büyük bitiş süresi.

Atama	Süreler (A, B, C)	En büyük	Nash?
A {2}, B {1}, C {1}	1, 1, 1	1	evet (optimum)
A {1, 1}, B {2}, C { }	1, 2, 0	**2**	evet

İkinci atamada 2'lik iş A'ya geçse (2 + 2)/2 = 2, C'ye geçse 2/1 = 2 bekler; 1'lik iş C'ye geçse yine 1 bekler: kimse kazanmaz. Oran 2/1 = 2, özdeş üç makinenin sınırı 2 − 2/4 = 1,5'ten büyük.

Özdeş makinelerde saf dengenin kaybı hep 2 katın altında kalır (2 − 2/(m+1)). Hızlar farklıysa kayıp, makine sayısı arttıkça (çok yavaş da olsa) sınırsız büyür (Czumaj–Vöcking, 2002). Yük Dengeleme oyununun 2. ve 3. seviyeleri bu modeli kullanır.

Oyuna Dönüş: Yük Dengeleme Oyunu

Oyunun ölçüsü farklı

Oyunda sosyal maliyet en büyük yük değil, tüm işlerin bitiş sürelerinin toplamıdır. Seviye 2, CPU Scheduling: 7 birim iş, hızlar 2, 1, 1.

Yükler (A, B, C)	Toplam süre	Nash?
(3, 2, 2)	3·1,5 + 2·2 + 2·2 = **12,5**	evet (optimum)
(4, 2, 1)	4·2 + 2·2 + 1·1 = **13**	evet (en kötü)

PoA = 13 / 12,5 = **1,04**. (4, 2, 1)'de A'daki iş C'ye geçse yine 2 bekler: kimse kazanmaz. Simetriği (4, 1, 2) de aynı maliyetle (13) bir dengedir.

Bencil yerleşim nereye varır?

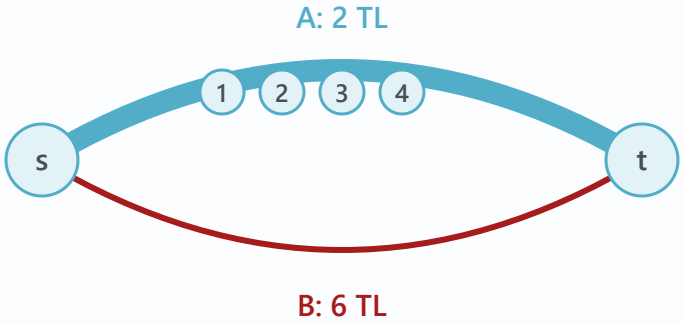
Her iş geldiği anda, kendi bitiş süresi en kısa olacak sunucuyu seçerse (eşitlikte alfabede önce gelen sunucu) sıra A, A, B, C, A, A, B olur ve sonuç tam olarak en kötü dengelerden (4, 2, 1) çıkar. Oyundaki bencil denge gösterimi (Selfish equilibrium) bu açgözlü yerleşimle kurulur.

Seviye 1'de (6 iş, özdeş sunucular) tek denge (2, 2, 2) optimumdur: PoA = 1. Kayıp, sunucular farklılaşınca ortaya çıkar.

{2, 2, 1, 1} örneğindeki 4/3 en büyük yükü, oyun toplam süreyi ölçer: PoA'nın değeri ölçülen sosyal maliyete bağlıdır.

İSTİKRARIN BEDELİ

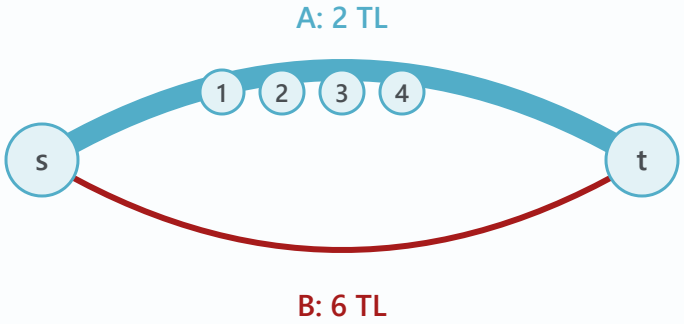
Aynı Oyunda PoA ≠ PoS



bağlantı bedeli kullananlarca eşit bölünür

4 oyuncu s'den t'ye bağlantı kiralar: A 2 TL, B 6 TL; bedeli kullananlar **eşit böler** (adil maliyet paylaşımı). Kalınlık: bağlantıdaki oyuncu sayısı.

Aynı Oyunda PoA ≠ PoS



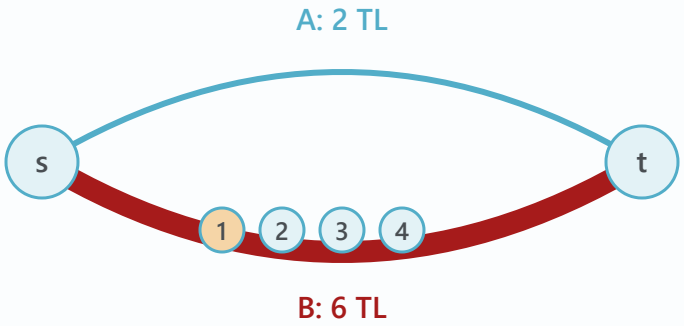
herkes A: kişi başı 0,5 TL

B'ye tek başına geçen: 6 TL

toplam 2 = optimum

Herkes A'da: her biri 0,5 öder; B'ye tek başına geçen 6 öderdi ⇒ kimse geçmez. **Denge, toplam 2 = optimum.**

Aynı Oyunda PoA ≠ PoS



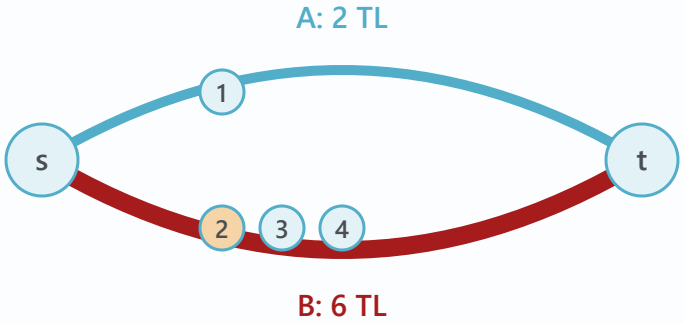
herkes B: kişi başı 1,5 TL

A'ya tek başına geçen: 2 TL

toplam 6

Herkes B'de: her biri 1,5 öder; A'ya tek başına geçen 2 öderdi (2 > 1,5) ⇒ kimse geçmez.
Bu da denge, toplam 6.

Aynı Oyunda PoA ≠ PoS



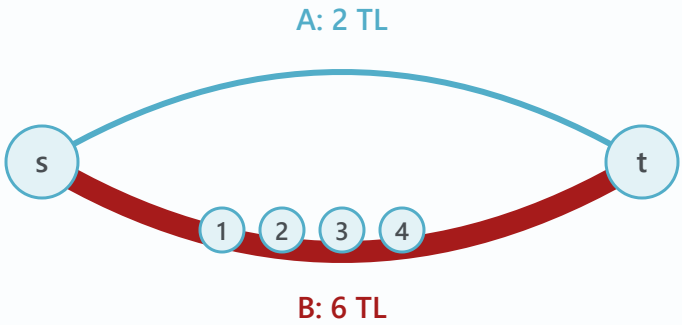
A'daki 1 kişi: 2 TL

B'deki 3 kişi: 2 TL

B'den A'ya geçen: 1 TL ⇒ kaçar

Bölünmüş profil (1 A, 3 B): B'deki bir oyuncu A'ya geçerse 1 öder < 2 ⇒ denge değil. Betik 16 profilin hepsini taradı: yalnız "hepsi A" ve "hepsi B" denge.

Aynı Oyunda PoA ≠ PoS

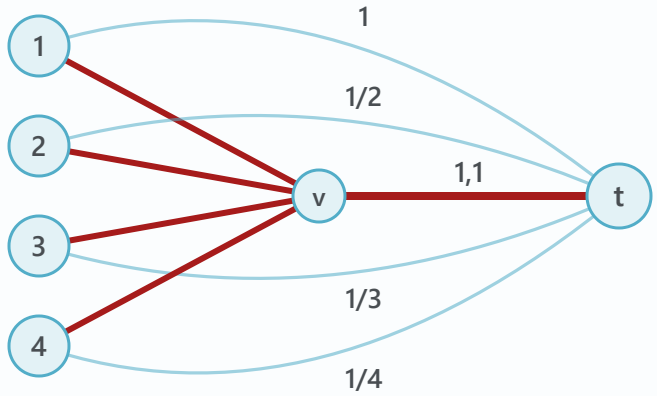


PoA = 6/2 = 3

PoS = 2/2 = 1

PoA = 6/2 = 3 (en kötü denge), **PoS = 1** (en iyi denge). Koordinasyon sorunu: oyuncular iyi dengeye yönlendirilebilirse kayıp yok; PoS bu iyimser senaryonun ölçüsüdür.

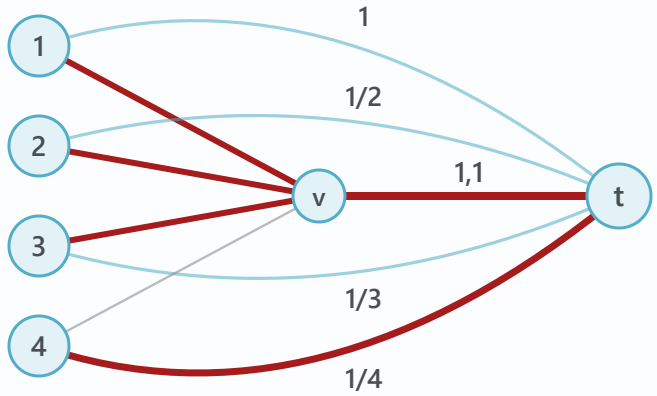
Adım Adım: İstikrarın Bedeli (H_n Örneği)



oyuncu	yol	öder
1	ortak	0,275
2	ortak	0,275
3	ortak	0,275
4	ortak	0,275
toplam		1,1

Her oyuncu t'ye iki yoldan biriyle gider: yalnız kendisinin kullandığı 1/i bedelli yol ya da herkesin ortak kullandığı, bedeli 1 + ε = 1,1 olan yol (ε: küçük bir ek bedel); ortak yolun bedeli kullananlarca eşit bölünür. Optimum: herkes ortakta, toplam **1,1**, kişi başı 0,275.

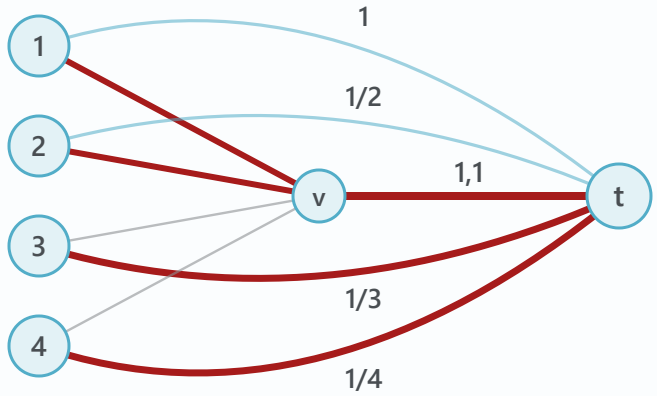
Adım Adım: İstikrarın Bedeli (H_n Örneği)



oyuncu	yol	öder
1	ortak	0,367
2	ortak	0,367
3	ortak	0,367
4	kendi	0,25
toplam		1,35

Oyuncu 4: kendi yolu 1/4 = 0,25 < 0,275 → **ayrılır**. Kalan üç kişi artık 1,1/3 ≈ 0,367 öder.

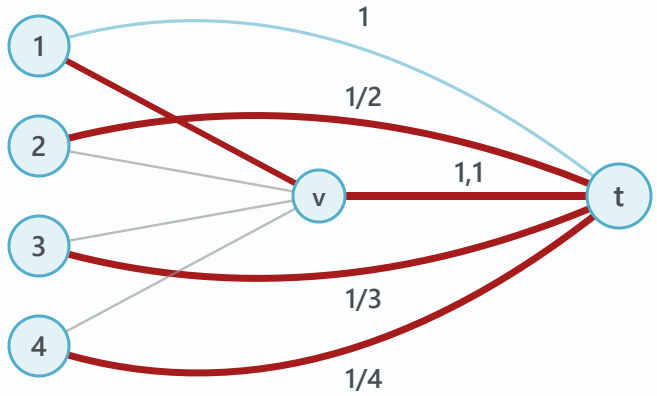
Adım Adım: İstikrarın Bedeli (H_n Örneği)



oyuncu	yol	öder
1	ortak	0,55
2	ortak	0,55
3	kendi	0,333
4	kendi	0,25
toplam		1,683

Oyuncu 3: 1/3 ≈ 0,333 < 0,367 → ayrılır. Kalan iki kişi 0,55 öder.

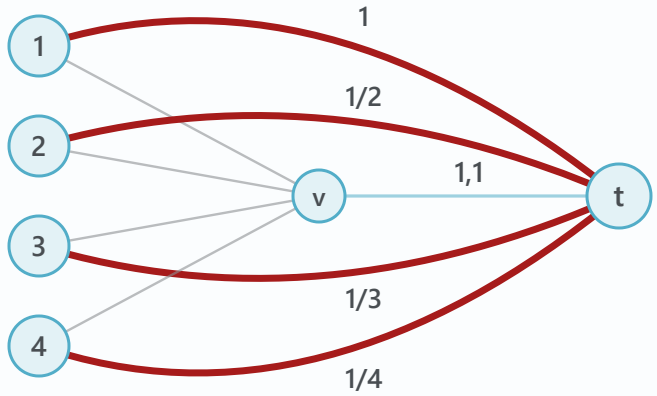
Adım Adım: İstikrarın Bedeli (H_n Örneği)



oyuncu	yol	öder
1	ortak	1,1
2	kendi	0,5
3	kendi	0,333
4	kendi	0,25
toplam		2,183

Oyuncu 2: 1/2 = 0,5 < 0,55 → ayrılır. Oyuncu 1 ortak yolun bedelini tek başına öder: 1,1.

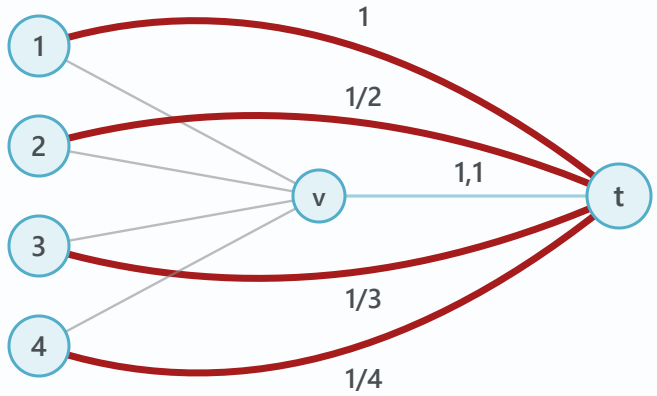
Adım Adım: İstikrarın Bedeli (H_n Örneği)



oyuncu	yol	öder
1	kendi	1
2	kendi	0,5
3	kendi	0,333
4	kendi	0,25
toplam		2,083

Oyuncu 1: kendi yolu 1 < 1,1 → ayrılır. Herkes kendi yolunda: 1 + 1/2 + 1/3 + 1/4 = **25/12 ≈ 2,083**.

Adım Adım: İstikrarın Bedeli (H_n Örneği)



oyuncu	yol	öder
1	kendi	1
2	kendi	0,5
3	kendi	0,333
4	kendi	0,25
toplam		2,083

Betik: 16 profilin **tek** dengesi bu → PoS = PoA = (25/12) / 1,1 = 125/66 ≈ 1,894. ε → 0 iken n oyuncuda PoS → $H_n = 1 + 1/2 + \ldots + 1/n$.

Neden PoS ≤ H_n? Potansiyel ve Maliyet

Gösterim. c_e: e bağlantısının bedeli; n_e: e'yi kullanan oyuncu sayısı (her biri c_e/n_e öder). $H(k) = 1 + 1/2 + \ldots + 1/k$; ör. $H(1) = 1$, $H(4) = 25/12 \approx 2{,}08$. $H_n = H(n)$. Toplam maliyet $C(s)$: kullanılan bağlantıların bedelleri toplamı.

Neden PoS ≤ H_n? Potansiyel ve Maliyet

Gösterim. c_e: e bağlantısının bedeli; n_e: e'yi kullanan oyuncu sayısı (her biri c_e/n_e öder). $H(k) = 1 + 1/2 + \ldots + 1/k$; ör. $H(1) = 1$, $H(4) = 25/12 \approx 2{,}08$. $H_n = H(n)$. Toplam maliyet $C(s)$: kullanılan bağlantıların bedelleri toplamı.

Rosenthal potansiyeli. Oyuncular bağlantıya tek tek eklenmiş gibi düşün: 1. kişi c_e, 2. kişi $c_e/2$, …, k. kişi c_e/k yazar. Bağlantının katkısı $c_e \cdot H(n_e)$ olur: $\mathbf{\Phi(s) = \Sigma_e\, c_e \cdot H(n_e)}$.

Neden PoS ≤ H_n? Potansiyel ve Maliyet

Gösterim. c_e: e bağlantısının bedeli; n_e: e'yi kullanan oyuncu sayısı (her biri c_e/n_e öder). $H(k) = 1 + 1/2 + \ldots + 1/k$; ör. $H(1) = 1$, $H(4) = 25/12 \approx 2{,}08$. $H_n = H(n)$. Toplam maliyet $C(s)$: kullanılan bağlantıların bedelleri toplamı.

Rosenthal potansiyeli. Oyuncular bağlantıya tek tek eklenmiş gibi düşün: 1. kişi c_e, 2. kişi $c_e/2$, …, k. kişi c_e/k yazar. Bağlantının katkısı $c_e \cdot H(n_e)$ olur: $\mathbf{\Phi(s) = \Sigma_e\, c_e \cdot H(n_e)}$.

Örnek (H_n örneği, dört oyuncu). Herkes ortak yolda: $C = 1{,}1$ ve $\Phi = 1{,}1 \cdot H(4) = 1{,}1 \cdot 25/12 \approx 2{,}29$. Herkes kendi yolunda: her yolu 1 kişi kullanır, $H(1) = 1$, yani $\Phi = C = 25/12 \approx 2{,}08$.

İSTİKRARIN BEDELİ

Neden PoS ≤ H_n? Potansiyel ve Maliyet

Gösterim. c_e: e bağlantısının bedeli; n_e: e'yi kullanan oyuncu sayısı (her biri c_e/n_e öder). $H(k) = 1 + 1/2 + \ldots + 1/k$; ör. $H(1) = 1$, $H(4) = 25/12 \approx 2{,}08$. $H_n = H(n)$. Toplam maliyet $C(s)$: kullanılan bağlantıların bedelleri toplamı.

Rosenthal potansiyeli. Oyuncular bağlantıya tek tek eklenmiş gibi düşün: 1. kişi c_e, 2. kişi $c_e/2$, …, k. kişi c_e/k yazar. Bağlantının katkısı $c_e \cdot H(n_e)$ olur: $\mathbf{\Phi(s) = \Sigma_e\, c_e \cdot H(n_e)}$.

Örnek (H_n örneği, dört oyuncu). Herkes ortak yolda: $C = 1{,}1$ ve $\Phi = 1{,}1 \cdot H(4) = 1{,}1 \cdot 25/12 \approx 2{,}29$. Herkes kendi yolunda: her yolu 1 kişi kullanır, $H(1) = 1$, yani $\Phi = C = 25/12 \approx 2{,}08$.

Potansiyel, maliyetin en az 1, en çok H_n katıdır. Kullanılan her bağlantıda $1 \le H(n_e) \le H_n$ (en çok n kullanıcı var). Her terimi c_e ile çarpıp topla: her s için $\mathbf{C(s) \le \Phi(s) \le H_n \cdot C(s)}$.

İSTİKRARIN BEDELİ

PoS ≤ H_n: Optimumdan Bir Dengeye Yürümek

1. Optimumdan başla. s^{OPT} profilinde en iyi tepki dinamiğini çalıştır: her adımda bir oyuncu kendisi için daha ucuz bir yola geçer.

PoS ≤ H_n: Optimumdan Bir Dengeye Yürümek

1. Optimumdan başla. s^{OPT} profilinde en iyi tepki dinamiğini çalıştır: her adımda bir oyuncu kendisi için daha ucuz bir yola geçer.

2. Φ her hamlede azalır. Potansiyel oyunda sapan oyuncunun maliyet değişimi Φ'nin değişimine eşittir; oyuncu kazanıyorsa Φ düşer. Profil sayısı sonlu olduğundan dinamik bir s dengesinde durur ve **Φ(s) ≤ Φ(OPT)**.

İSTİKRARIN BEDELİ

PoS ≤ H_n: Optimumdan Bir Dengeye Yürümek

1. Optimumdan başla. s^{OPT} profilinde en iyi tepki dinamiğini çalıştır: her adımda bir oyuncu kendisi için daha ucuz bir yola geçer.

2. Φ her hamlede azalır. Potansiyel oyunda sapan oyuncunun maliyet değişimi Φ'nin değişimine eşittir; oyuncu kazanıyorsa Φ düşer. Profil sayısı sonlu olduğundan dinamik bir s dengesinde durur ve **Φ(s) ≤ Φ(OPT)**.

3. Zinciri kur. C(s) ≤ Φ(s) (potansiyel maliyetten küçük değil) ≤ Φ(OPT) (2. adım) ≤ H_n·C(OPT) (potansiyel maliyetin en çok H_n katı). Örnekte: 2,08 ≤ 2,08 ≤ 2,29 ≤ 2,29.

İSTİKRARIN BEDELİ

PoS ≤ H_n: Optimumdan Bir Dengeye Yürümek

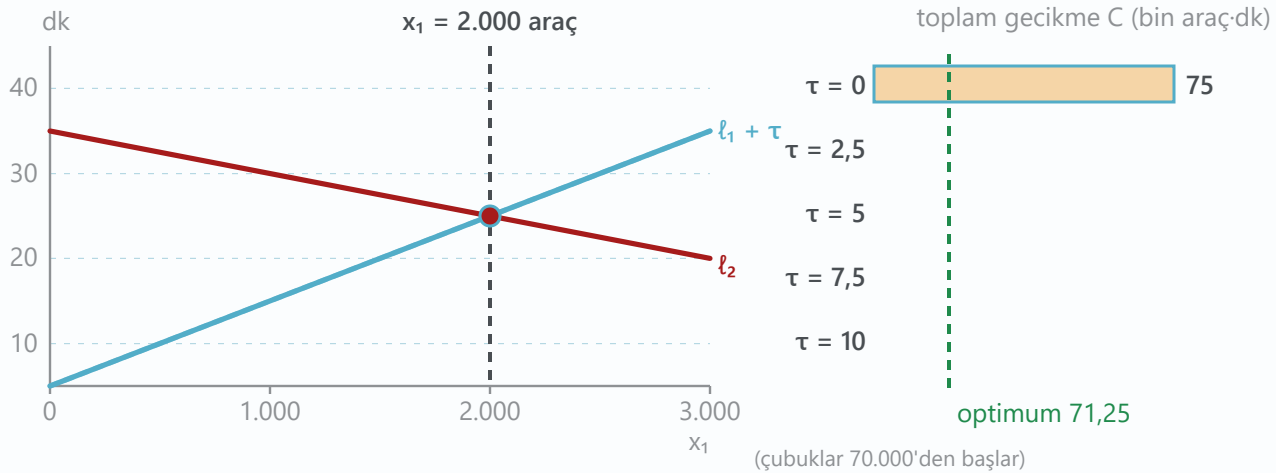
1. Optimumdan başla. s^{OPT} profilinde en iyi tepki dinamiğini çalıştır: her adımda bir oyuncu kendisi için daha ucuz bir yola geçer.

2. Φ her hamlede azalır. Potansiyel oyunda sapan oyuncunun maliyet değişimi Φ'nin değişimine eşittir; oyuncu kazanıyorsa Φ düşer. Profil sayısı sonlu olduğundan dinamik bir s dengesinde durur ve **Φ(s) ≤ Φ(OPT)**.

3. Zinciri kur. C(s) ≤ Φ(s) (potansiyel maliyetten küçük değil) ≤ Φ(OPT) (2. adım) ≤ H_n·C(OPT) (potansiyel maliyetin en çok H_n katı). Örnekte: 2,08 ≤ 2,08 ≤ 2,29 ≤ 2,29.

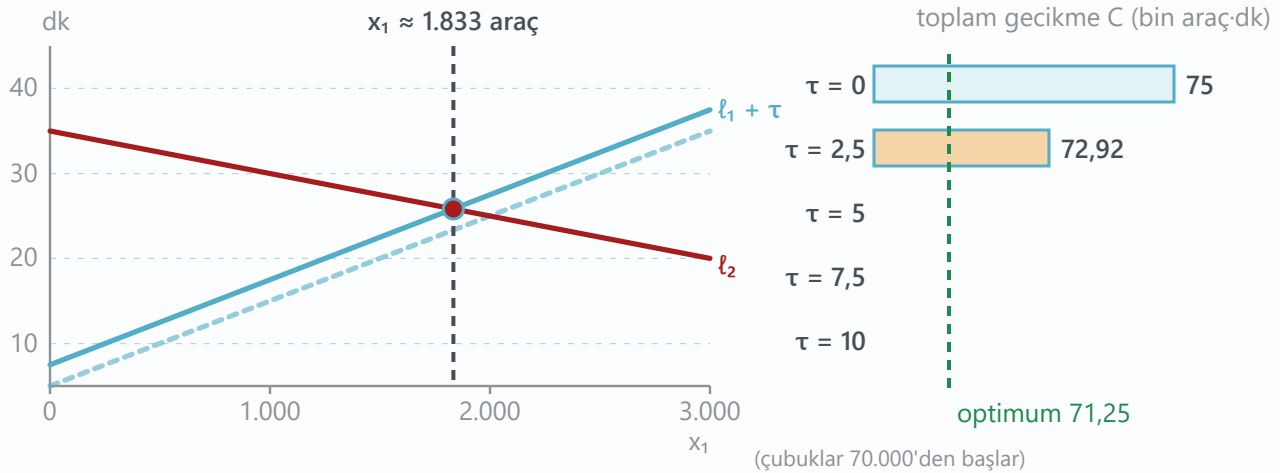
En az bir dengenin maliyeti optimumun en çok H_n katıdır: **PoS ≤ H_n** (Anshelevich vd., 2004). H_n çok yavaş büyür (H_{100} ≈ 5,19; yaklaşık ln n). H_n örneği sınırın sıkı olduğunu gösterir. PoA ise n kadar büyük olabilir.

Adım Adım: Tıkanıklık Ücreti Dengeyi Kaydırır



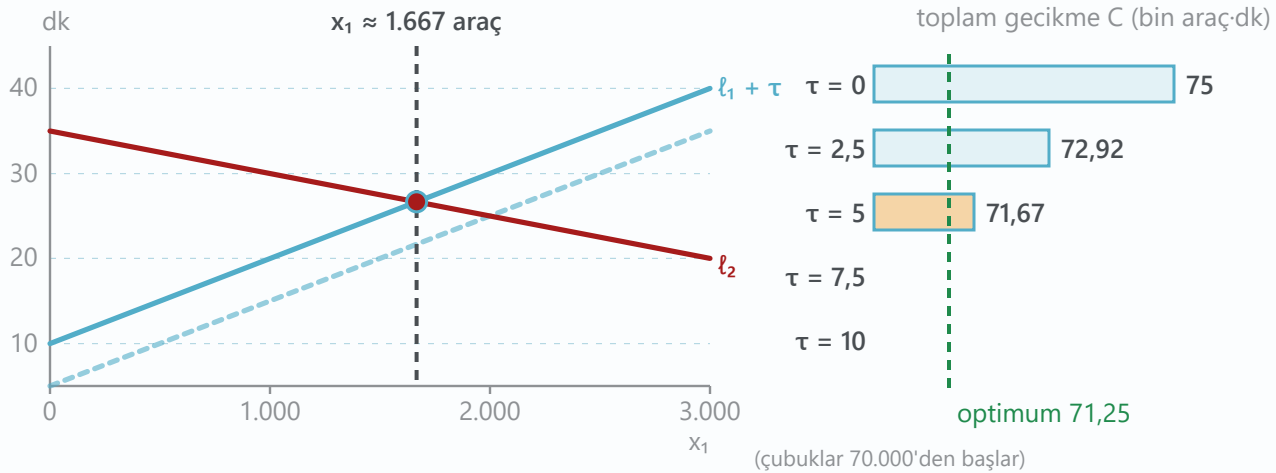
İki yollu ağ: $\ell_1 = 5 + x_1/100$, $\ell_2 = 20 + x_2/200$, toplam 3.000 araç. Ücret yok ($\tau = 0$): denge $\ell_1 = \ell_2 \rightarrow x_1 = 2.000$, toplam 75.000 araç·dk. Ücret sürücü için dakika cinsinden ek maliyet sayılır.

Adım Adım: Tıkanıklık Ücreti Dengeyi Kaydırır



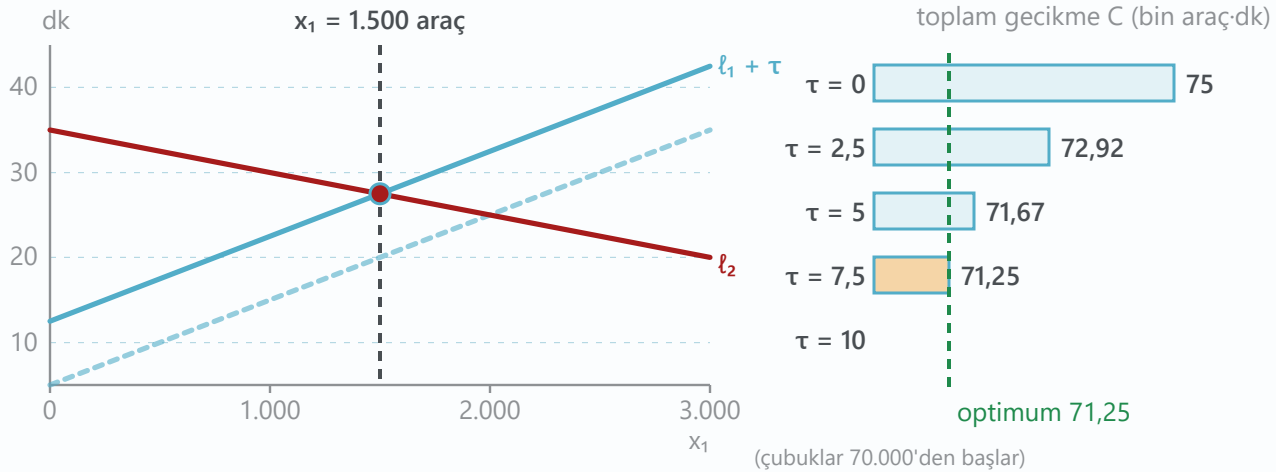
τ = 2,5: Yol 1 eğrisi yukarı kayar; kesişim sola gider: $x_1 \approx 1.833$, toplam ≈ 72.917. Ücret bir *transfer*dir, toplam gecikmeye eklenmez.

Adım Adım: Tıkanıklık Ücreti Dengeyi Kaydırır



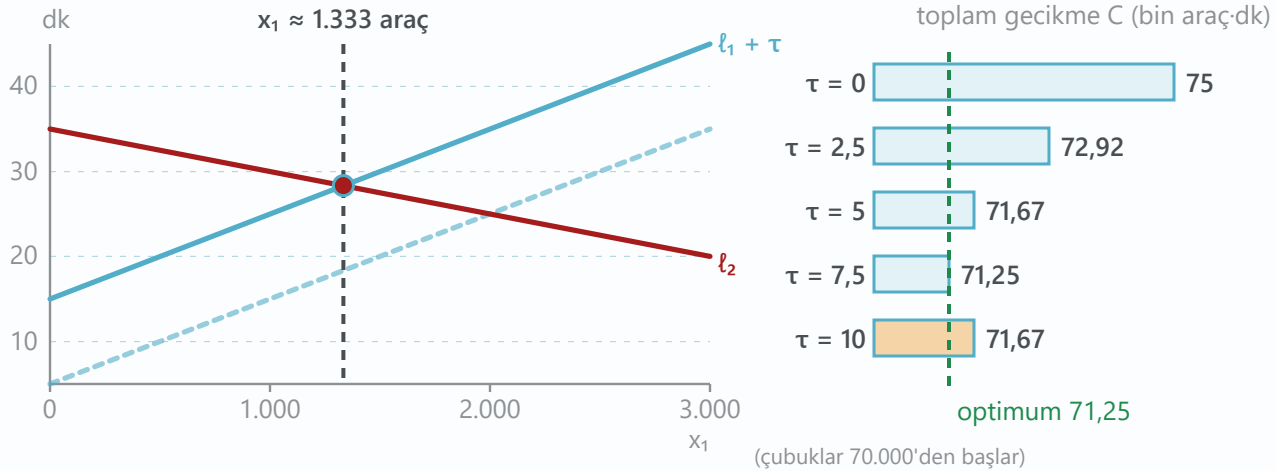
$\tau = 5$: $x_1 \approx 1.667$, toplam ≈ 71.667. Her artış bir kısım sürücüyü geniş Yol 2'ye iter.

Adım Adım: Tıkanıklık Ücreti Dengeyi Kaydırır



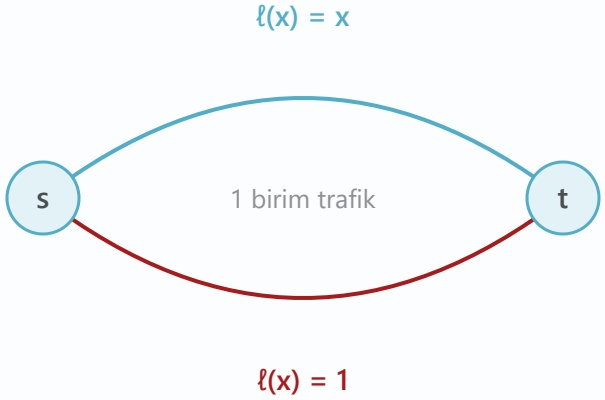
$\tau = 7{,}5$: $x_1 = 1.500$ → **sistem optimumu** (71.250) artık bir denge. Marjinal maliyet ücretleri Yol 1'e $x_1\ell'_1 = 15$, Yol 2'ye $x_2\ell'_2 = 7{,}5$ dk; önemli olan fark: 7,5.

Adım Adım: Tıkanıklık Ücreti Dengeyi Kaydırır



τ = 10: fazla ücret aşırıya kaçar: $x_1 \approx 1.333$, toplam yine ≈ 71.667. Doğru ücret dışsallığa eşit olandır, “ne kadar çok o kadar iyi” değil.

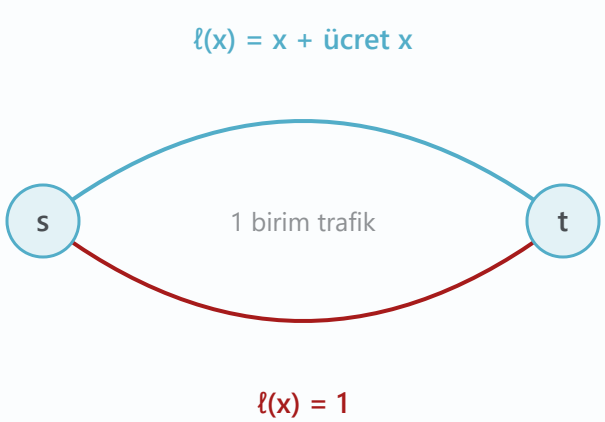
Adım Adım: Tıkanıklık Ücreti Pigou'yu Düzeltir

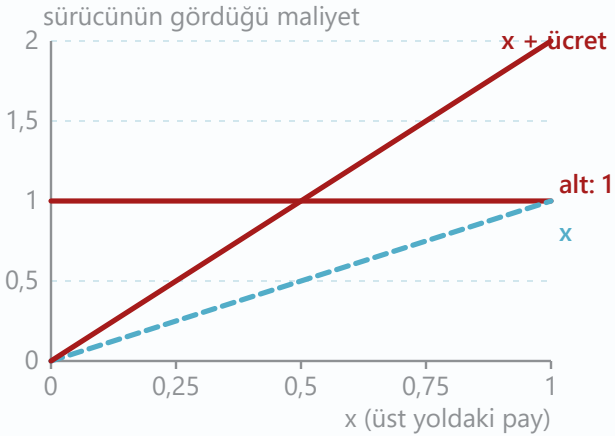




Başlangıç: ücret yok, herkes üst yolda (x = 1), toplam gecikme 1. Amaç: optimumu (x = 1/2) denge yapmak.

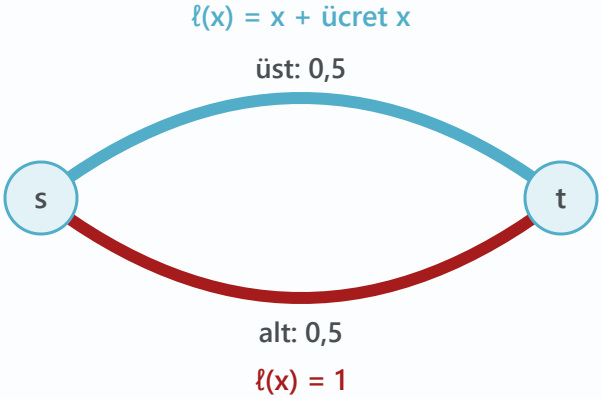
Adım Adım: Tıkanıklık Ücreti Pigou'yu Düzeltir

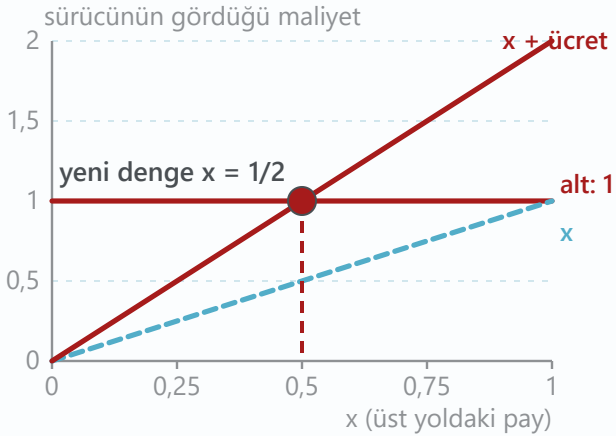




Marjinal maliyet ücreti $\tau_e = f_e \cdot \ell'_e(f_e)$: üst yolda $\ell' = 1 \rightarrow \boldsymbol{\tau = x}$; alt yolda $\ell' = 0 \rightarrow$ ücret yok. Sürücü üstte $x + x = 2x$ görür.

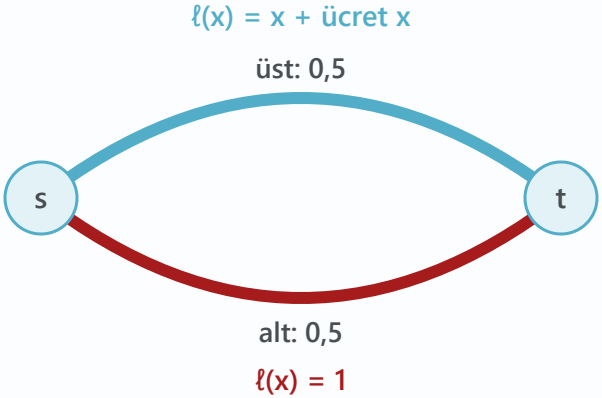
Adım Adım: Tıkanıklık Ücreti Pigou'yu Düzeltir

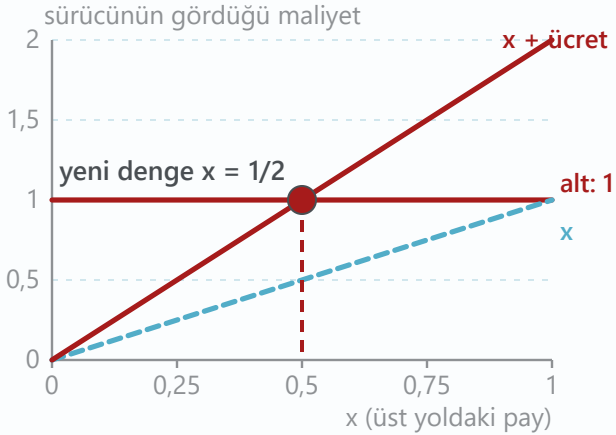




Yeni denge: 2x = 1 → **x = 1/2**. Bencil sürücüler kendiliğinden optimum dağılımı seçer; gecikme 3/4.

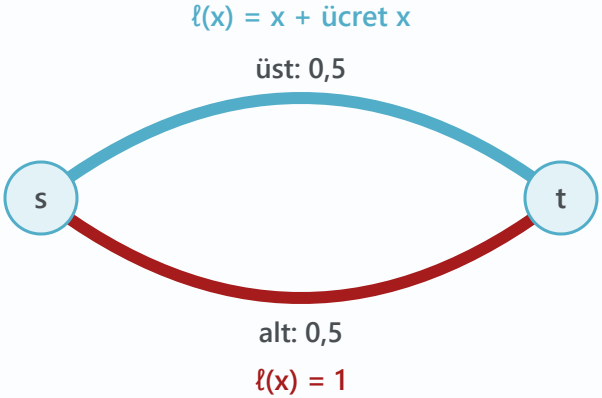
Adım Adım: Tıkanıklık Ücreti Pigou'yu Düzeltir

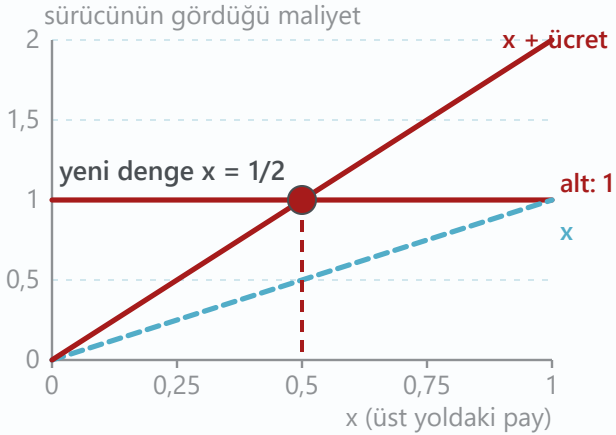




Toplanan ücret: 1/2 · 1/2 = 1/4. Sürücülerin ödediği toplam 3/4 + 1/4 = 1: gelir geri dağıtılmazsa sürücüler kazanmaz, kazanç topluma geçer.

Adım Adım: Tıkanıklık Ücreti Pigou'yu Düzeltir





Aynı sonucu sabit ücret de verir: üst yola τ = 1/2 → x + 1/2 = 1 → x = 1/2. Genel teorem: marjinal maliyet ücreti *her* ağda optimumu denge yapar.

POA'YI AZALTMA

PoA'yı Azaltmanın Diğer Yolları

Tıkanıklık fiyatlandırması

Kullanıcıların başkalarına yüklediği dışsal maliyeti karara dahil eder (Londra, Stockholm, Singapur uygulamaları).

Merkezî (Stackelberg) yönlendirme

Trafiğin bir kısmını merkez yönlendirir, gerisi bencil seçer; kontrol edilen pay arttıkça oran 1'e yaklaşır.

Kapasite planlama

Roughgarden–Tardos: r birim trafiğin bencil dengesi, 2r birim trafiğin en iyi yönlendirilmesinden daha pahalı değildir. Kapasiteyi artırmak ücretin yerini tutabilir.

Rota önerisi ve bilgi

Öneriler dengeyi daha düşük toplam maliyete yönlendirebilir; ama uyum gönüllüdür.

Mekanizma tasarımı

Kuralları, iyi sonuç tek denge olacak biçimde kurmak.

Maliyeti hesaba katmak

Müdahalenin bilgi, uygulama ve adalet maliyetleri elde edilen verimlilik kazancıyla birlikte değerlendirilmelidir.

Gerçek Hayat ve Tartışma

Trafik ağlarında ücretli şeritler, internette ağlar arası rota seçimi, enerji şebekesinde zamana bağlı tarifeler: üçünde de soru aynıdır. Bencil denge ne kadar kötü, hangi müdahale onu optimuma yaklaştırır?

Tartışma soruları

- Bireysel çıkarlar neden her zaman sosyal optimuma ulaşmaz? Pigou örneğinde hangi dışsallık var?
- PoA'yı azaltmak için hangi teşvikler uygulanabilir (ücret, bonus, merkezî yönlendirme)? Hangisi daha adil?
- Braess paradoksu günlük hayatta nerede görülebilir? Bir yolu kapatmak ne zaman mantıklıdır?

DEĞERLENDİRME

Özet

Örnek	Denge	Optimum	Oran
İki yol, 3.000 araç	75.000	71.250	PoA ≈ 1,053
Pigou (x ve 1)	1	3/4	PoA = 4/3
Doğrusal gecikme, her ağ	—	—	PoA ≤ 4/3 (sıkı)
Pigou, $\ell = x^p$	1	→ 0	PoA → ∞
Atomik tıkanıklık, doğrusal	—	—	PoA ≤ 5/2 (sıkı)
Braess, 4.000 araç	320.000	258.750	PoA ≈ 1,237
Yük dengeleme {2, 2, 1, 1}	4 (en kötü)	3	PoA = 4/3, PoS = 1
Adil paylaşım, H_n örneği	25/12	1,1	PoS ≈ 1,894 (≤ H_n)
Pigou + ücret $\tau = x$	3/4	3/4	1

DEĞERLENDİRME

Örnek Problemler

1. Pigou ağında üst yolun gecikmesi x^2 olsun. Denge maliyeti, optimum ve PoA nedir?
2. İşler {3, 3, 2, 2, 2}, 2 özdeş makine. Optimum en büyük yük nedir? En büyük yükü 7 olan bir Nash dengesi bulun.
3. Normalleştirilmiş Braess ağında (s→A: x, A→t: 1, s→B: 1, B→t: x, A→B: 0; 1 birim trafik) PoA nedir?
4. H_n örneğini $n = 3$ için kurun (ε küçük). PoS kaça yaklaşır?
5. Üst yol $2x$, alt yol 1 olan Pigou ağında PoA ve optimumu denge yapan sabit ücret nedir?

▶ **Cevapları göster**

DEĞERLENDİRME

Alıştırmalar: İzlemeler ve Yeni Konular

1. `pigou_poa(3)` kaç döndürür? ($x^{OPT} = 4^{-1/3}$)
2. Dört iş (4, 3, 3, 2) başta hepsi M1'de; her adımda kârlı sapan ilk iş en az yüklü makineye geçer. Bu en iyi tepki dinamiği hangi en büyük yükte durur? Optimum kaçtır?
3. Bir oyun (1, ½)-pürüzsüz ise PoA için ne söylenebilir? Pigou'nun (1, ¼) değeriyle karşılaştırın.
4. PoA ≠ PoS oyununda A 3 TL, B 8 TL olsun (4 oyuncu). Dengeler hangileri, PoA ve PoS kaçtır?
5. $y = 2$, $z = 3$ için $y(z + 1) \le 5/3 \cdot y^2 + 1/3 \cdot z^2$ eşitsizliğini doğrulayın.
6. `poa_pos([3, 2, 2, 1], 2)` ne döndürür?

▶ **Cevapları göster**

DEĞERLENDİRME

Etkileşimli Oyunlar

Yük Dengeleme ve PoA

İşlerin makinelere dağılımı: denge, optimum ve aradaki oran.

Bencil Yönlendirme

Wardrop dengesi, sistem optimumu ve aradaki oran (PoA).

Kaynaklar

- T. Roughgarden, *Twenty Lectures on Algorithmic Game Theory*, Cambridge University Press, 2016 (Ders 11–15).
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (ed.), *Algorithmic Game Theory*, Cambridge University Press, 2007 (Bölüm 17–20).
- E. Koutsoupias, C. Papadimitriou, "Worst-case Equilibria", *STACS*, 1999.
- T. Roughgarden, É. Tardos, "How Bad Is Selfish Routing?", *Journal of the ACM*, 2002.
- T. Roughgarden, "The Price of Anarchy Is Independent of the Network Topology", *J. Computer and System Sciences*, 2003.
- E. Anshelevich vd., "The Price of Stability for Network Design with Fair Cost Allocation", *FOCS*, 2004.
- T. Roughgarden, "Intrinsic Robustness of the Price of Anarchy", *STOC*, 2009.
- G. Christodoulou, E. Koutsoupias, "The Price of Anarchy of Finite Congestion Games", *STOC*, 2005.
- B. Awerbuch, Y. Azar, A. Epstein, "The Price of Routing Unsplittable Flow", *STOC*, 2005.
- A. C. Pigou, *The Economics of Welfare*, Macmillan, 1920.
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017 (Bölüm 8: Anarşinin Bedeli); D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, Cambridge, 2010 (Bölüm 8: trafik ve Braess).
- Ders sayfası: T. Roughgarden, CS364A (Stanford, 2013), Ders 11–15 notları.

Kaynaklar (2/2)

- C. Papadimitriou, “Algorithms, Games, and the Internet”, *STOC*, 2001.
- R. L. Graham, “Bounds for Certain Multiprocessing Anomalies”, *Bell System Technical Journal* 45(9), 1966.
- G. Finn, E. Horowitz, “A Linear Time Approximation Algorithm for Multiprocessor Scheduling”, *BIT* 19, 1979.
- E. Koutsoupias, M. Mavronicolas, P. Spirakis, “Approximate Equilibria and Ball Fusion”, *Theory of Computing Systems* 36(6), 2003.
- A. Czumaj, B. Vöcking, “Tight Bounds for Worst-Case Equilibria”, *SODA*, 2002.
- T. Roughgarden, “Stackelberg Scheduling Strategies”, *SIAM Journal on Computing* 33(2), 2004.

Son

Bölüm 9: Anarşinin Bedeli

Sonraki bölüm: Bölüm 10: Çoklu Ajan Sistemleri

Çalışma soruları · Terim sözlüğü