Bölüm 14: Gelecek Yönelimler

Algoritmik Oyun Kuramı

Yapay zekâ ajanları | Öğrenen mekanizmalar | Blokzincir mekanizmaları | Hesaplama ve ölçek | Dersin haritası

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_14_Gelecek_Yonelimler.html

GİRİŞ

Oyunla Başlangıç: Yapay Zekâ Ajan Turnuvası

Oyunun kuralı

Ajanlar seçilen oyunu 60 ya da 120 tur oynar; tam turnuvada her ajan her ajanla bir kez karşılaşır ve toplam kazanca göre sıralanır.

Merak sorusu

- Hep işbirliği yapan, hep ihanet eden, rastgele oynayan ve rakibin en sık hamlesine en iyi tepki veren ajanlardan hangisi birinci olur?
- Bir oyunda birinci olan ajan başka bir oyunda da birinci olur mu?



Mahkûmun İkilemi, Seviye 1: her maç 60 tur; her ajan diğer üçüyle birer kez oynar (6 maç).

[Yapay Zekâ Ajan Turnuvası →]

GİRİŞ

Öğrenme Çıktıları

Yeni oyuncular

Yapay zekâ ajanlarının pazarlık ve fiyatlandırmada yarattığı oyun kuramı sorularını tanımlamak.

Veriden tasarım

Mekanizmanın veriden öğrenilmesini ve örneklem karmaşıklığı kavramını sayısal örnekle açıklamak.

Zincir üstü mekanizmalar

İşlem ücreti açık artırmalarını, taban ücret kuralını ve MEV (işlem sıralama gücünden elde edilen değer) kavramını yorumlamak.

Dersi birleştirmek

Dersin bütün bölümlerini tek haritada ilişkilendirmek.

GİRİŞ

Algoritmik Oyun Kuramı: Gelişen Bir Alan

- Endüstri ve akademide uygulamalar artıyor: otonom araçlar, çevrim içi platformlar, enerji ağları.
- Teknolojik eğilimler: yapay zekâ ve makine öğrenmesi, büyük veri ve gerçek zamanlı karar, blokzincir ve akıllı sözleşmeler.

Adalet ve etik
Adil ve şeffaf mekanizmalar.

Dinamik ve öğrenen sistemler
Uyarlanabilir otonom ajanlar.

Büyük ağlar
Milyonlarca oyunculu stratejik etkileşim.

Teşvik ve politika
Sosyal refah ve sürdürülebilirlik için kural tasarımı.

Bu bölüm, dersin araçlarının bu yeni alanlarda nasıl kullanıldığını ve nerede yetersiz kaldığını gösterir.

Uygulama Alanları

Finans

Algoritmik ticaret, portföy yönetimi; alım-satım algoritmaları birbirine karşı oynar.

Trafik ve lojistik

Otonom araç ağları, rota seçimi; bencil yönlendirme ve Braess.

Çevrim içi platformlar

Öneri sistemleri, reklam açık artırmaları, sosyal ağ analizi.

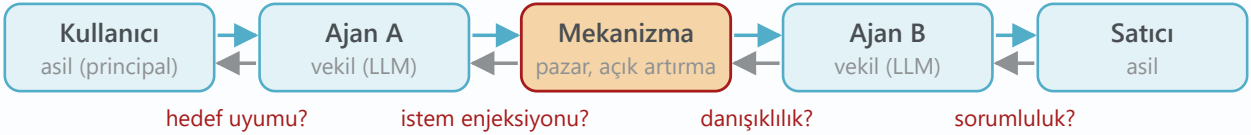
Enerji şebekeleri

Yük dengeleme, talep yanıtı, elektrik piyasaları.

YAPAY ZEKÂ AJANLARI

Stratejik Yapay Zekâ Ajanları

- Büyük dil modeli (LLM) tabanlı ajanlar kullanıcı adına alışveriş, pazarlık, planlama ve kaynak paylaşımı üstlenebilir.
- Ajan artık bir **oyuncu**: hedefi, bilgisi ve stratejisi olan bir aktör. Oyun kuramının soruları yeniden gündemde.



Hedef uyumu
Ajanın hedefi kullanıcıyla aynı mı? Asil–vekil (principal–agent) sorunu.

Rasyonellik varsayımı
LLM davranışı çerçevelemeye (framing) duyarlı olabilir; Nash tahmininden sapabilir.

Manipülasyon
Karşı ajanın mesajına gizlenen talimatlar (istem enjeksiyonu: prompt injection).

Kimlik ve sorumluluk
Kim adına konuşuyor, hatadan kim sorumlu, sözleşme nasıl doğrulanır?

Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	1	2	3	4	5	6	7	8	∞
A'nın payı									

100 TL paylaşılacak. A tek, B çift turlarda önerir; her gecikme değeri %10 azaltır (δ = 0,9). Tur 4 son tur. Geriye doğru çıkarım.

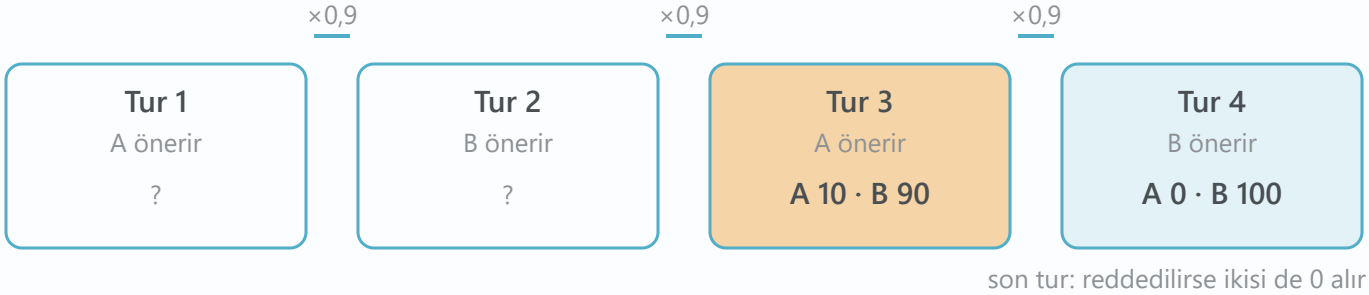
Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**∞**
A'nın payı									

Tur 4: B önerir. A reddederse ikisi de 0 alır → A her şeyi kabul eder (kayıtsız kalan taraf kabul eder varsayılır): **B 100, A 0**.

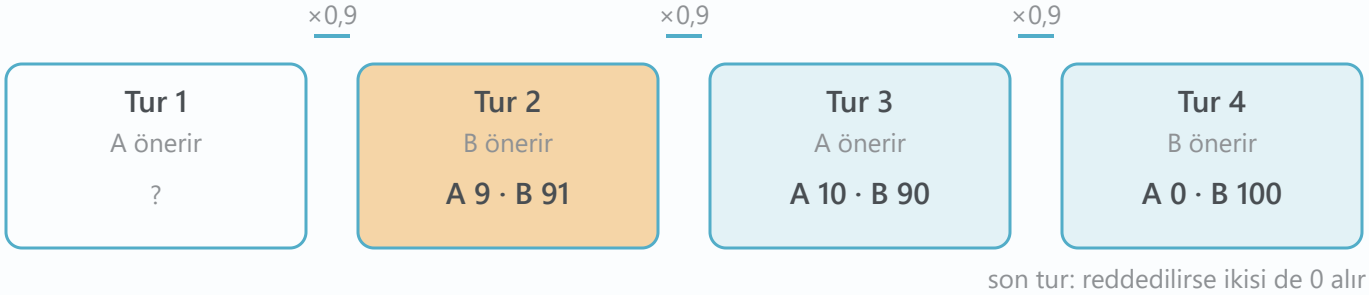
Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**∞**
A'nın payı									

Tur 3: B reddederse tur 4'te 100 alır; bugünkü değeri 0,9·100 = 90. A, B'ye 90 önerir: **A 10, B 90**.

Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	1	2	3	4	5	6	7	8	∞
A'nın payı									

Tur 2: A'nın bekleme değeri 0,9·10 = 9. B, A'ya 9 önerir: **A 9, B 91**.

Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	1	2	3	4	5	6	7	8	∞
A'nın payı									

Tur 1: B'nin bekleme değeri 0,9·91 = 81,9. A 18,1 alır, anlaşma **hemen** olur (alt oyun mükemmel denge).

Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**∞**
A'nın payı	100	10	91	18,1	83,7	24,7	77,8	30	52,6

Tur sayısı değişince pay salınır; T = 1 ültimatom oyunudur (A hepsini alır). Sonsuz ufukta (Rubinstein, 1982) önerenin payı 1/(1 + δ) = **52,6**.

Adım Adım: Ajanlar Arası Pazarlık



Tur sayısı T	1	2	3	4	5	6	7	8	∞
A'nın payı	100	10	91	18,1	83,7	24,7	77,8	30	52,6

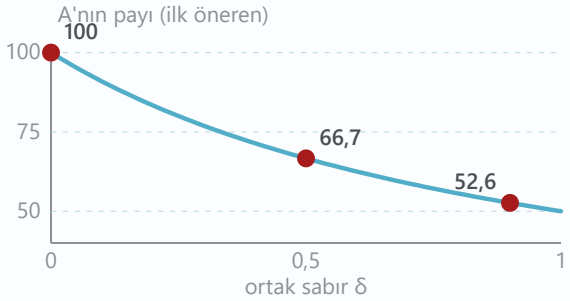
Ajanlar için ders: **sabır (δ) ve son söz hakkı pazarlık gücüdür**. Son tarih, son teklifin kimde olduğunu belirlediği için ufuk bilgisi de stratejiktir.

Sonsuz Ufuk: Rubinstein Çözümü

- Son tur yoksa geriye doğru çıkarım başlayamaz; çözüm **durağan** stratejilerle (her turda aynı teklifi yapmak) bulunur: A önerirken kendine hep x*, B önerirken A'ya hep y* bırakır.
- B kabul ile reddetme arasında kayıtsız olmalı: $1 - x^* = \delta_B \cdot (1 - y^*)$. A da öyle: $y^* = \delta_A \cdot x^*$.
- İkinci denklemi birinciye koyalım: $1 - x^* = \delta_B \cdot (1 - \delta_A \cdot x^*) \Rightarrow x^* \cdot (1 - \delta_A \delta_B) = 1 - \delta_B$:

$$x^* = (1 - \delta_B) / (1 - \delta_A \delta_B)$$

Simetrik sabırda ($\delta_A = \delta_B = \delta$) $x^* = 1/(1 + \delta)$: $\delta = 0{,}9$ için $100/1{,}9 \approx$ **52,6**. Öneri ilk turda kabul edilir; gecikme olmadığı için hiçbir değer kaybolmaz.



δ_A	δ_B	A	B
0,9	0,9	52,6	47,4
0,9	0,8	71,4	28,6
0,8	0,9	35,7	64,3
0,99	0,9	91,7	8,3

Ajan tasarımına yansıması: **sabır güçtür**. Gerçekten aceleci (δ'sı düşük) ajan daha az pay alır. Eksik bilgide aceleci *görünmek* de rakibin inancını değiştirerek benzer etki yapar.

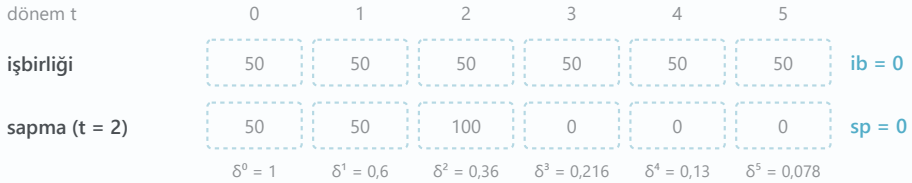
Fiyatlandırma Algoritmaları Danışabilir mi?

- İki firma aynı ürünü satıyor; tekel fiyatında toplam kâr 100 (her birine 50). Fiyat kırılırsa kâr sıfırlanır (Bertrand rekabeti: ucuz olan bütün pazarı alır).
- Kinci tetik stratejisi (grim trigger): "Rakip fiyat kırana kadar yüksek fiyat, kırarsa sonsuza dek rekabet."
- Yapı tekrarlanan Mahkûmun İkilemi ile aynıdır: sapan bir dönem 100 alır, sonra 0; uyan her dönem 50 alır. Tetik koşulu $50/(1 - \delta) \geq 100 \iff$ **$\delta \geq 1/2$**.

Simülasyonlarda pekiştirmeli öğrenen fiyatlandırma ajanlarının, açıkça iletişim kurmadan rekabetçi düzeyin üstünde fiyatlar öğrenebildiği gösterilmiştir (Calvano ve diğ., 2020).

Düzenleme sorusu: açık bir anlaşma yoksa **algoritmik danışıklılık** (algorithmic collusion) var mıdır? Rekabet hukuku geleneksel olarak "insanlar arası anlaşma" üzerine kuruludur; açıkça iletişim kurmadan yüksek fiyatı öğrenen algoritmalar bu çerçeveyi zorlar.

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



İki yol karşılaştırılır: her dönem 50 ile **işbirliği** ve t = 2'de fiyat kırıp o dönem 100 alan **sapma**. Tetik stratejisi gereği sapmadan sonra kâr 0. δ = 0,6, 6 dönem.

Adım 1 / 9

```
1  def karsilastir(delta, s, T):
▶       ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
3      for t in range(T):
4          ib += delta**t * 50
5          if t < s:
6              sp += delta**t * 50
7          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
9          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`delta = 0.6` `s = 2` `T = 6` `ib = 0.0` `sp = 0.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?

dönem t	0	1	2	3	4	5	
işbirliği	50	50	50	50	50	50	ib = 50
sapma (t = 2)	50	50	100	0	0	0	sp = 50
	$\delta^0 = 1$	$\delta^1 = 0{,}6$	$\delta^2 = 0{,}36$	$\delta^3 = 0{,}216$	$\delta^4 = 0{,}13$	$\delta^5 = 0{,}078$	

t = 0: iki yol da 50 kazanır ($\delta^0 = 1$).

Adım 2 / 9

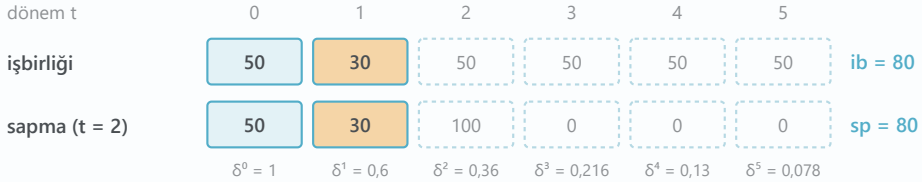
```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
▶      for t in range(T):
▶          ib += delta**t * 50
▶          if t < s:
▶              sp += delta**t * 50
7          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
9          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`t = 0` `ib = 50.0` `sp = 50.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



t = 1: 0,6·50 = 30 eklenir; iki yol hâlâ eşit: 80.

Adım 3 / 9

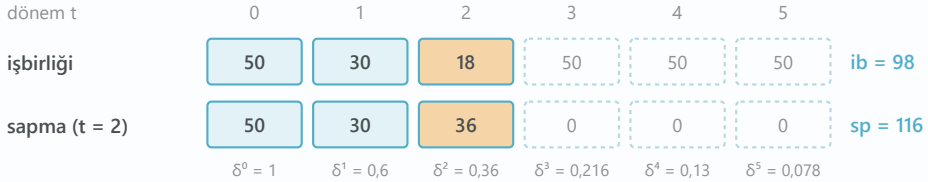
```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
▶      for t in range(T):
▶          ib += delta**t * 50
▶          if t < s:
▶              sp += delta**t * 50
7          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
9          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`t = 1` `ib = 80.0` `sp = 80.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



t = 2: sapma anı. A pazarın tümünü alır: 0,36·100 = 36; işbirliği yolu yalnızca 0,36·50 = 18 ekler. Sapma **18 önde**.

Adım 4 / 9

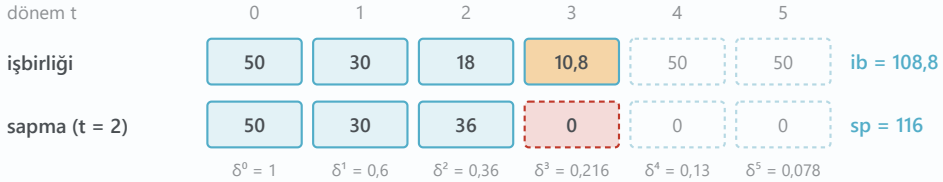
```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
▶      for t in range(T):
▶          ib += delta**t * 50
5          if t < s:
6              sp += delta**t * 50
▶          elif t == s:
▶              sp += delta**t * 100   # A fiyat kırar
9          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`t = 2` `ib = 98.0` `sp = 116.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



t = 3: tetik devrede, rakip fiyatı kalıcı kırar; sapma yolu 0 alır, işbirliği 10,8 ekler.

Adım 5 / 9

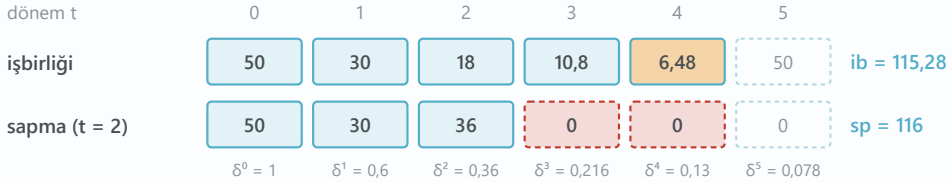
```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
▶      for t in range(T):
▶          ib += delta**t * 50
▶          if t < s:
6              sp += delta**t * 50
▶          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
▶          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`t = 3` `ib = 108.8` `sp = 116.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



t = 4: işbirliği 6,48 ekler: 115,28. Fark 0,72'ye indi.

Adım 6 / 9

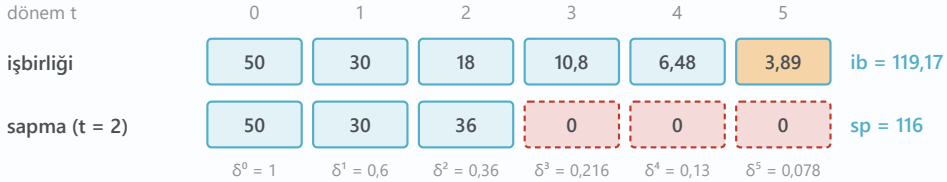
```
 1 def karsilastir(delta, s, T):
 2     ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
 ▶     for t in range(T):
 ▶         ib += delta**t * 50
 ▶         if t < s:
 6             sp += delta**t * 50
 ▶         elif t == s:
 8             sp += delta**t * 100   # A fiyat kırar
 ▶         # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`t = 4` `ib = 115.28` `sp = 116.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



t = 5: işbirliği 3,89 ekler ve **öne geçer**: 119,17 > 116. Kısa vadeli 18'lik kazanç, kaybedilen gelecek kârlarla geri ödenir.

Adım 7 / 9

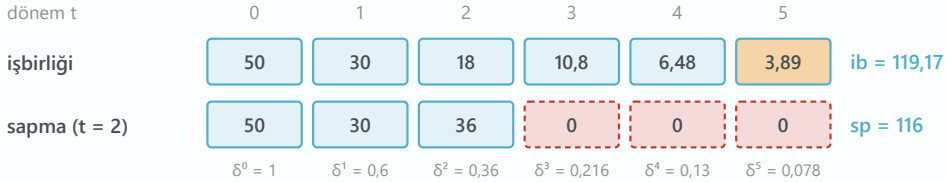
```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
▶      for t in range(T):
▶          ib += delta**t * 50
▶          if t < s:
6              sp += delta**t * 50
▶          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
▶          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`t = 5` `ib = 119.168` `sp = 116.0`

ÇIKTI

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



Sonuç yazdırılır. Sonsuz ufukta fark daha büyüktür: 50/(1 − 0,6) = 125'e karşı 116.

Adım 8 / 9

```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
3      for t in range(T):
4          ib += delta**t * 50
5          if t < s:
6              sp += delta**t * 50
7          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
9          # t > s: tetik devrede, kâr 0
▶      return round(ib, 2), round(sp, 2)
▶  print(karsilastir(0.6, 2, 6))
12 print(karsilastir(0.4, 2, 6))
```

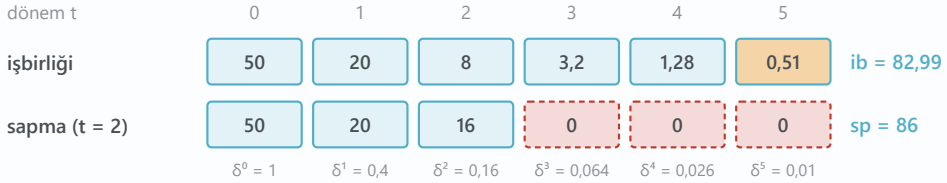
DEĞİŞKENLER

`ib = 119.17` `sp = 116.0`

ÇIKTI

```
(119.17, 116.0)
```

Kod İzleme: Tetik Stratejisinde Sapma Kârlı mı?



δ = 0,4: sapma 86, işbirliği 82,99 (sonsuzda 83,3). Sapma kârlı, danışıklılık çöker; eşik δ = 1/2.

Adım 9 / 9

```
1  def karsilastir(delta, s, T):
2      ib, sp = 0.0, 0.0            # işbirliği / sapma yolu
3      for t in range(T):
4          ib += delta**t * 50
5          if t < s:
6              sp += delta**t * 50
7          elif t == s:
8              sp += delta**t * 100   # A fiyat kırar
9          # t > s: tetik devrede, kâr 0
10     return round(ib, 2), round(sp, 2)
11 print(karsilastir(0.6, 2, 6))
▶  print(karsilastir(0.4, 2, 6))
```

DEĞİŞKENLER

`ib = 82.99` `sp = 86.0`

ÇIKTI

```
(119.17, 116.0)
(82.99, 86.0)
```

Danışıklılığı Kolaylaştıran ve Zorlaştıran Koşullar

firma n	pay 100/n	gereken δ
2	50	0,50
3	33,3	0,67
4	25	0,75
5	20	0,80
10	10	0,90

$(100/n)/(1 - \delta) \geq 100 \iff \delta \geq 1 - 1/n$. Firma çoğaldıkça işbirliği payı küçülür, sapmanın cazibesi büyür.

Sık fiyat güncelleme

Fiyat dakikada bir güncellenirse gelecek neredeyse bugün kadar değerlidir ($\delta \approx 1$); sapanın kazancı birkaç dakikayla sınırlı kalır.

Şeffaf fiyatlar

Rakip fiyatı anında görülürse sapma hemen fark edilir, ceza gecikmez.

Fiyat eşleme garantisi

"Rakip ucuzsa farkı öderiz" fiyat kırmanın kazancını sıfırlar; danışıklılığı korur.

Düzenleme önerileri

Algoritmayı benzetim ortamında sınamak, karar kayıtlarını denetime açmak.

Oyuna Dönüş: Yapay Zekâ Ajan Turnuvası

Mahkûmun İkilemi, Seviye 1

Kazançlar (3,3), (0,5), (5,0), (1,1); üç maçın beklenen toplamı:

Ajan	Beklenen toplam
Hep İhanet	**544**
En İyi Tepki	535,5
Rastgele	303,5
Hep İşbirliği	93

En İyi Tepki ilk turda (geçmiş yokken) işbirliği yapar; her maç 60 tur.

İhanet baskın olduğu için En İyi Tepki ilk turdan sonra hep ihanet eder; karşılıklı işbirliği neredeyse hiç görülmez.

Ortam kazananı belirler

Yazı-Tura Eşleştirmede sıralama değişir: sabit ajanları sömüren En İyi Tepki yaklaşık +118 ile birinci olur; yalnızca Rastgele ajan hiçbir rakibe karşı sömürülemez (her maçta beklenen 0).

Gerçekte: Axelrod'un (1980) tekrarlanan Mahkûmun İkilemi turnuvasını Tit-for-Tat kazandı: karşılıklılık ve ceza tehdidi işbirliğini korudu. Buradaki ajanlarda geçmişe göre cezalandırma yok; Calvano vd.'nin (2020) fiyat ajanları ise önceki dönemi hatırladığı için cezalandırmayı öğrenip yüksek fiyatı sürdürebildi.

Evrensel en iyi strateji yoktur: başarı oyuna, rakip kitlesine, gürültüye ve hafızaya bağlıdır.

Otomatik Mekanizma Tasarımı

Klasik yol

- İnsan bir kural önerir (Vickrey, VCG, Myerson).
- Teşvik uyumluluğu ve optimalliği elle ispatlanır.
- Tek nesne gibi yapılı problemlerde çok başarılı; çok nesneli, çok alıcılı durumlarda optimum çoğu zaman bilinmez.

Otomatik yol

- Mekanizma bir optimizasyon probleminin çözümüdür: amaç (gelir, refah) + kısıtlar: IC (teşvik uyumluluğu: doğruyu söylemek en iyisi), IR (bireysel rasyonellik: katılmak zarar ettirmez).
- Sonlu tip kümelerinde doğrusal programlama (Conitzer ve Sandholm, 2002).
- Sinir ağı ile açık artırma: IC kısıtı "pişmanlık" cezasına dönüşür (Dütting ve diğ., 2019). Burada pişmanlık, teklifçinin yalan söyleyerek kazanabileceği ek faydadır.

Yaklaşık IC

Öğrenilen kural tam doğru olmayabilir; küçük pişmanlık kalır.

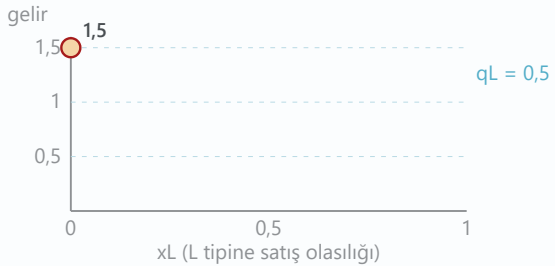
Genelleme

Eğitim verisindeki dağılım değişirse kural etkisizleşebilir.

Açıklanabilirlik

"Neden bu fiyat?" sorusu ağırlıklarla yanıtlanamaz.

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

L (değer 1): x = 0, p = 0

H (değer 3): x = 1, p = 3

H'nin faydası: dürüst 0, L dese 0

Tek alıcı, tek nesne. Tip L (değer 1) olasılığı qL, tip H (değer 3). xL = 0: L'ye hiç satılmaz, H'ye 3'ten satılır. Gelir 0,5·3 = **1,5**. Genel kural: L'ye fiyat, L'nin ödemeye razı olduğu en yüksek değerdir: pL = xL·1. H, L gibi davranırsa 3·xL − pL kazanır; H'yi doğruyu söylemeye razı eden en yüksek fiyat pH = pL + 3·(1 − xL)'dir.

Adım 1 / 8

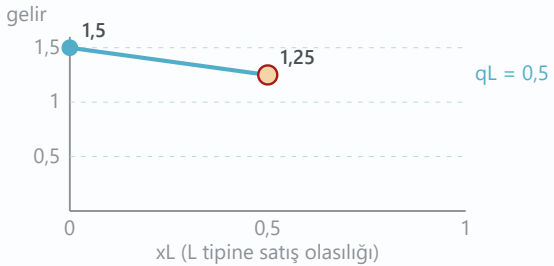
```
1 def gelir(xL, qL, vL=1, vH=3):
▶     pL = xL * vL                # L'nin IR kısıtı sıkı
▶     pH = pL + vH * (1 - xL)     # H'nin IC kısıtı sıkı
▶     return qL * pL + (1 - qL) * pH
▶ for qL in [0.5, 0.75]:
▶     G = [gelir(x, qL) for x in [0, 0.5, 1]]
7     print(qL, G)
```

DEĞİŞKENLER

qL = 0.5 | xL = 0 | pL = 0 | pH = 3 | gelir = 1.5

ÇIKTI

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

L (değer 1): x = 0,5, p = 0,5

H (değer 3): x = 1, p = 2

H'nin faydası: dürüst 1, L dese 1

xL = 0,5: L'ye yarı olasılıkla 0,5'e satmak H'ye bir yalan seçeneği açar (L desem 3·0,5 − 0,5 = 1 kazanırım). H'nin fiyatı 2'ye iner. Gelir 0,25 + 1 = **1,25**.

Adım 2 / 8

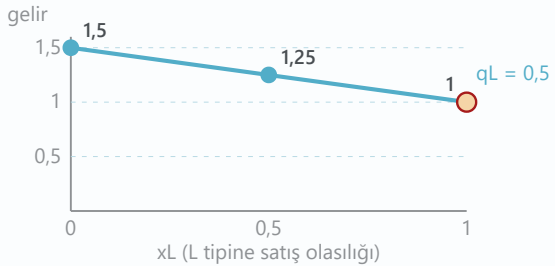
```
1 def gelir(xL, qL, vL=1, vH=3):
    pL = xL * vL                # L'nin IR kısıtı sıkı
    pH = pL + vH * (1 - xL)     # H'nin IC kısıtı sıkı
    return qL * pL + (1 - qL) * pH
for qL in [0.5, 0.75]:
    G = [gelir(x, qL) for x in [0, 0.5, 1]]
7   print(qL, G)
```

DEĞİŞKENLER

qL = 0.5 | xL = 0.5 | pL = 0.5 | pH = 2 | gelir = 1.25

ÇIKTI

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

L (değer 1): x = 1, p = 1

H (değer 3): x = 1, p = 1

H'nin faydası: dürüst 2, L dese 2

xL = 1: herkese 1'den satılır; H yalan söylese de aynı fiyatı öder. Gelir **1**.

Adım 3 / 8

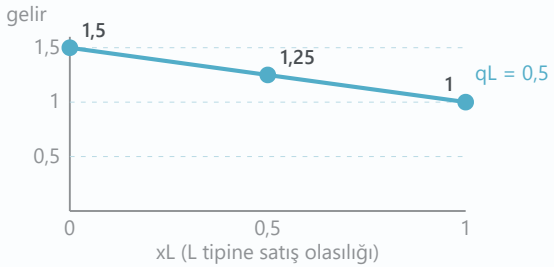
```
1 def gelir(xL, qL, vL=1, vH=3):
▶     pL = xL * vL                # L'nin IR kısıtı sıkı
▶     pH = pL + vH * (1 - xL)     # H'nin IC kısıtı sıkı
▶     return qL * pL + (1 - qL) * pH
▶ for qL in [0.5, 0.75]:
▶     G = [gelir(x, qL) for x in [0, 0.5, 1]]
7     print(qL, G)
```

DEĞİŞKENLER

qL = 0.5 | xL = 1 | pL = 1 | pH = 1 | gelir = 1.0

ÇIKTI

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

qL = 0,5: en iyisi xL = 0, yani **yalnız H'ye 3'ten sat**.

Adım 4 / 8

```
1 def gelir(xL, qL, vL=1, vH=3):
2     pL = xL * vL                # L'nin IR kısıtı sıkı
3     pH = pL + vH * (1 - xL)     # H'nin IC kısıtı sıkı
4     return qL * pL + (1 - qL) * pH
5 for qL in [0.5, 0.75]:
6     G = [gelir(x, qL) for x in [0, 0.5, 1]]
▶     print(qL, G)
```

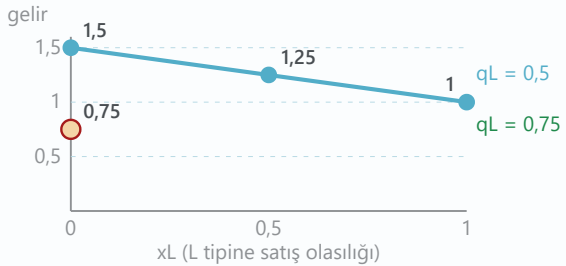
DEĞİŞKENLER

qL = **0.5** G = **[1.5, 1.25, 1.0]**

ÇIKTI

```
0.5 [1.5, 1.25, 1.0]
```

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

L (değer 1): x = 0, p = 0

H (değer 3): x = 1, p = 3

H'nin faydası: dürüst 0, L dese 0

qL = 0,75: L tipi çoğunlukta. Yalnız H'ye satmak artık az kazandırır: 0,25·3 = **0,75**.

Adım 5 / 8

```
1 def gelir(xL, qL, vL=1, vH=3):
      pL = xL * vL                  # L'nin IR kısıtı sıkı
      pH = pL + vH * (1 - xL)       # H'nin IC kısıtı sıkı
      return qL * pL + (1 - qL) * pH
  for qL in [0.5, 0.75]:
      G = [gelir(x, qL) for x in [0, 0.5, 1]]
7     print(qL, G)
```

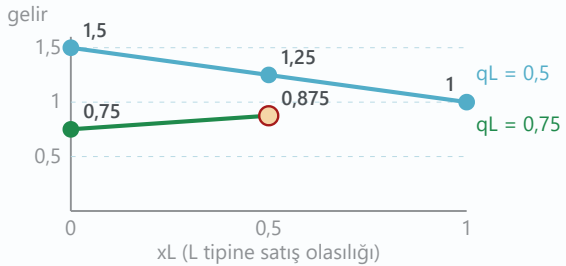
DEĞİŞKENLER

qL = 0.75 | xL = 0 | pL = 0 | pH = 3 | gelir = 0.75

ÇIKTI

```
0.5 [1.5, 1.25, 1.0]
```

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

L (değer 1): x = 0,5, p = 0,5

H (değer 3): x = 1, p = 2

H'nin faydası: dürüst 1, L dese 1

xL = 0,5: 0,75·0,5 + 0,25·2 = **0,875**.

Adım 6 / 8

```
1 def gelir(xL, qL, vL=1, vH=3):
      pL = xL * vL                  # L'nin IR kısıtı sıkı
      pH = pL + vH * (1 - xL)       # H'nin IC kısıtı sıkı
      return qL * pL + (1 - qL) * pH
  for qL in [0.5, 0.75]:
      G = [gelir(x, qL) for x in [0, 0.5, 1]]
7     print(qL, G)
```

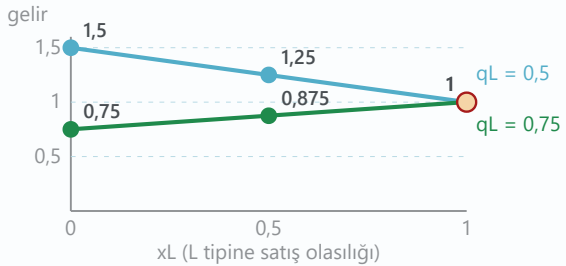
DEĞİŞKENLER

qL = **0.75** xL = **0.5** pL = **0.5** pH = **2** gelir = **0.875**

ÇIKTI

```
0.5 [1.5, 1.25, 1.0]
```

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

L (değer 1): x = 1, p = 1

H (değer 3): x = 1, p = 1

H'nin faydası: dürüst 2, L dese 2

xL = 1: herkese 1'den sat: **1**.

Adım 7 / 8

```
1 def gelir(xL, qL, vL=1, vH=3):
▶     pL = xL * vL                # L'nin IR kısıtı sıkı
▶     pH = pL + vH * (1 - xL)     # H'nin IC kısıtı sıkı
▶     return qL * pL + (1 - qL) * pH
▶ for qL in [0.5, 0.75]:
▶     G = [gelir(x, qL) for x in [0, 0.5, 1]]
7     print(qL, G)
```

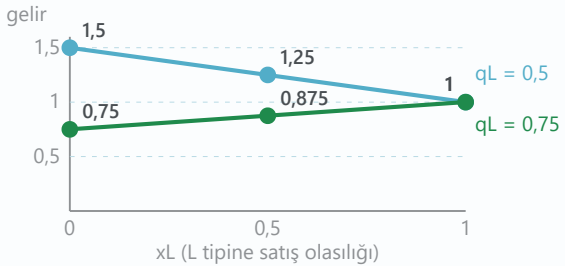
DEĞİŞKENLER

qL = **0.75** xL = **1** pL = **1** pH = **1** gelir = **1.0**

ÇIKTI

```
0.5 [1.5, 1.25, 1.0]
```

Kod İzleme: İki Tipli Alıcıya Otomatik Tasarım



mekanizma

qL = 0,75: en iyisi **herkese 1'den sat**. Gelir xL'de doğrusal olduğu için optimum hep bir uçtadır (doğrusal programın köşesi): iki çözüm de ilan edilmiş bir fiyattır.

Adım 8 / 8

```
1 def gelir(xL, qL, vL=1, vH=3):
2     pL = xL * vL                # L'nin IR kısıtı sıkı
3     pH = pL + vH * (1 - xL)     # H'nin IC kısıtı sıkı
4     return qL * pL + (1 - qL) * pH
5 for qL in [0.5, 0.75]:
6     G = [gelir(x, qL) for x in [0, 0.5, 1]]
▶     print(qL, G)
```

DEĞİŞKENLER

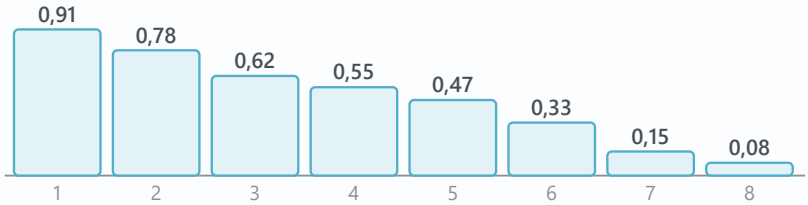
qL = **0.75**

G = **[0.75, 0.875, 1.0]**

ÇIKTI

```
0.5 [1.5, 1.25, 1.0]
0.75 [0.75, 0.875, 1.0]
```

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



Geçmişte gözlenen 8 alıcı değeri büyükten küçüğe sıralanır.

Adım 1 / 12

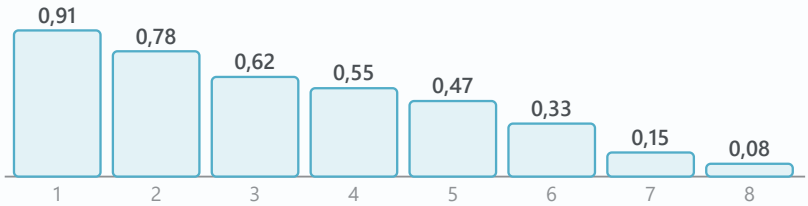
```
1  def en_iyi_fiyat(orn):
▶    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
6      p = v[k - 1]        # k. en yüksek
7      g = p * k / n       # k alıcı alır
8      if g > gelir:
9        fiyat, gelir = p, g
10   return fiyat, gelir
```

DEĞİŞKENLER

```
orn = [0.62, 0.15, 0.91, 0.47, 0.33,
       0.78, 0.55, 0.08]
```

```
v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33,
     0.15, 0.08]
```

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



n = 8 örnek. Fiyat p ise örneklerden p'ye eşit veya büyük olanlar satın alır.

Adım 2 / 12

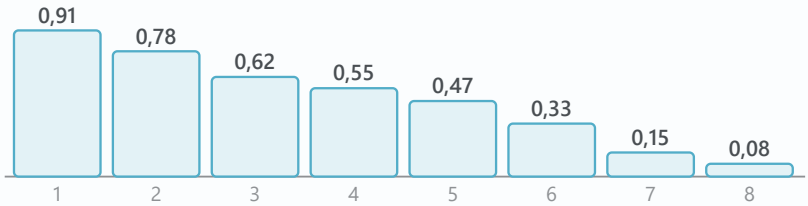
```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
▶    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
6      p = v[k - 1]        # k. en yüksek
7      g = p * k / n       # k alıcı alır
8      if g > gelir:
9        fiyat, gelir = p, g
10   return fiyat, gelir
```

DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



Şimdiye kadarki en iyi fiyat ve deneysel gelir.

Adım 3 / 12

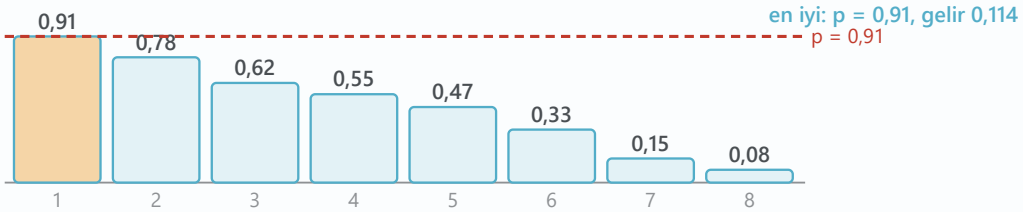
```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
▶    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
6      p = v[k - 1]        # k. en yüksek
7      g = p * k / n       # k alıcı alır
8      if g > gelir:
9        fiyat, gelir = p, g
10   return fiyat, gelir
```

DEĞİŞKENLER

`v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]`

`n = 8` `fiyat = 0` `gelir = 0`

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 1: p = 0,91 → 1 alıcı alır, g = 0,91·1/8 = **0,114** → yeni en iyi.

Adım 4 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]        # k. en yüksek
▶      g = p * k / n       # k alıcı alır
▶      if g > gelir:
▶        fiyat, gelir = p, g
10   return fiyat, gelir
```

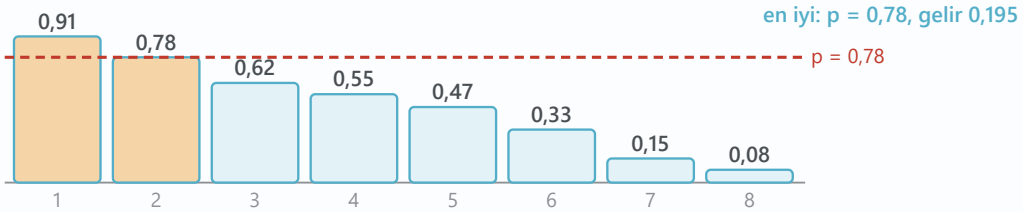
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 1 | p = 0.91 | g = 0.114

fiyat = 0.91 | gelir = 0.114

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 2: p = 0,78 → 2 alıcı alır, g = 0,78·2/8 = **0,195** → yeni en iyi.

Adım 5 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]        # k. en yüksek
▶      g = p * k / n       # k alıcı alır
▶      if g > gelir:
▶        fiyat, gelir = p, g
10   return fiyat, gelir
```

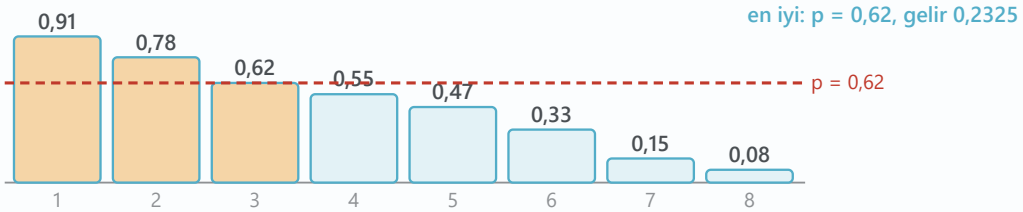
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 2 | p = 0.78 | g = 0.195

fiyat = 0.78 | gelir = 0.195

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 3: p = 0,62 → 3 alıcı alır, g = 0,62·3/8 = **0,2325** → yeni en iyi.

Adım 6 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]          # k. en yüksek
▶      g = p * k / n         # k alıcı alır
▶      if g > gelir:
▶        fiyat, gelir = p, g
10   return fiyat, gelir
```

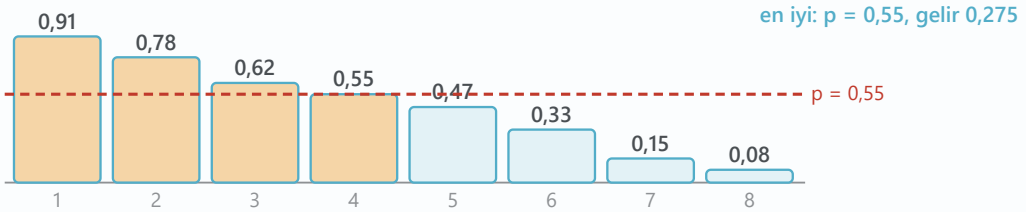
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 3 | p = 0.62 | g = 0.2325

fiyat = 0.62 | gelir = 0.2325

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 4: p = 0,55 → 4 alıcı alır, g = 0,55·4/8 = **0,275** → yeni en iyi.


```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]        # k. en yüksek
▶      g = p * k / n       # k alıcı alır
▶      if g > gelir:
▶        fiyat, gelir = p, g
10   return fiyat, gelir
```

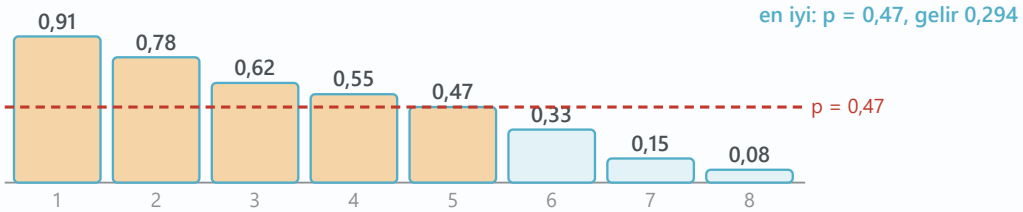
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 4 | p = 0.55 | g = 0.275

fiyat = 0.55 | gelir = 0.275

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 5: p = 0,47 → 5 alıcı alır, g = 0,47·5/8 = **0,294** → yeni en iyi.

Adım 8 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]        # k. en yüksek
▶      g = p * k / n       # k alıcı alır
▶      if g > gelir:
▶        fiyat, gelir = p, g
10   return fiyat, gelir
```

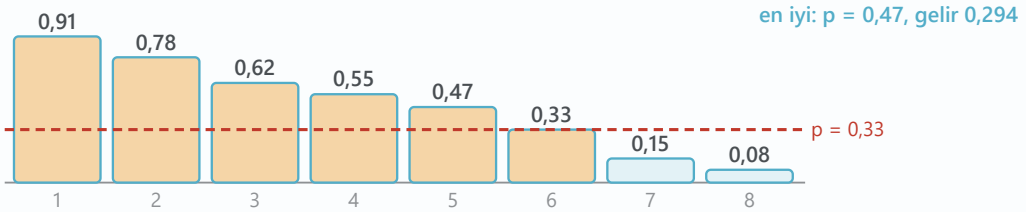
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 5 | p = 0.47 | g = 0.294

fiyat = 0.47 | gelir = 0.294

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 6: p = 0,33 → 6 alıcı alır, g = 0,33·6/8 = **0,2475** ≤ 0,294, değişmez.

Adım 9 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]          # k. en yüksek
▶      g = p * k / n         # k alıcı alır
▶      if g > gelir:
9        fiyat, gelir = p, g
10   return fiyat, gelir
```

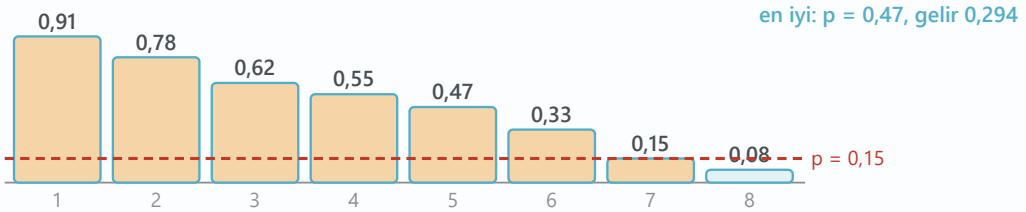
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 6 | p = 0.33 | g = 0.2475

fiyat = 0.47 | gelir = 0.294

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 7: p = 0,15 → 7 alıcı alır, g = 0,15·7/8 = **0,131** ≤ 0,294, değişmez.

Adım 10 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]          # k. en yüksek
▶      g = p * k / n         # k alıcı alır
▶      if g > gelir:
9        fiyat, gelir = p, g
10   return fiyat, gelir
```

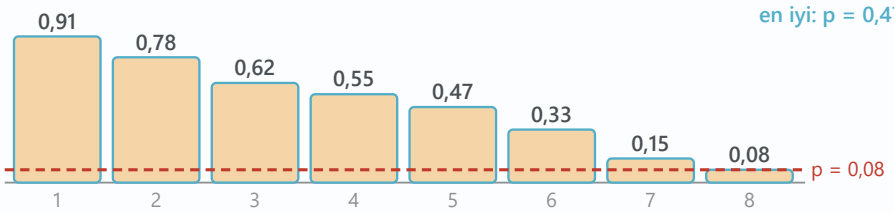
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 7 | p = 0.15 | g = 0.131

fiyat = 0.47 | gelir = 0.294

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



k = 8: p = 0,08 → 8 alıcı alır, g = 0,08·8/8 = **0,08** ≤ 0,294, değişmez.

Adım 11 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
▶      p = v[k - 1]          # k. en yüksek
▶      g = p * k / n         # k alıcı alır
▶      if g > gelir:
9        fiyat, gelir = p, g
10   return fiyat, gelir
```

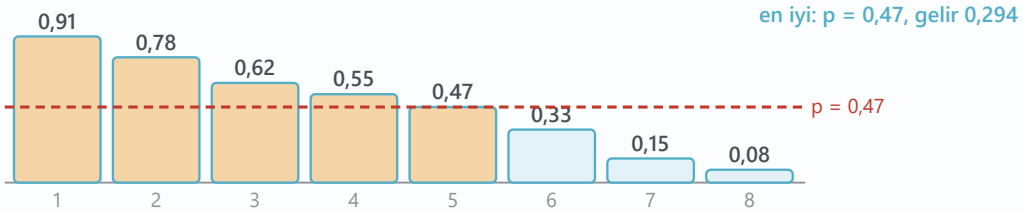
DEĞİŞKENLER

v = [0.91, 0.78, 0.62, 0.55, 0.47, 0.33, 0.15, 0.08]

n = 8 | k = 8 | p = 0.08 | g = 0.08

fiyat = 0.47 | gelir = 0.294

Kod İzleme: Veriden Rezerv Fiyatı Öğrenmek



Öğrenilen fiyat **0,47**, deneysel gelir 0,294. Değerler aslında U[0, 1]'den (0 ile 1 arasındaki her değer eşit olasılıklı) geliyorsa gerçek gelir 0,47·0,53 ≈ **0,249** (en iyisi 0,25).

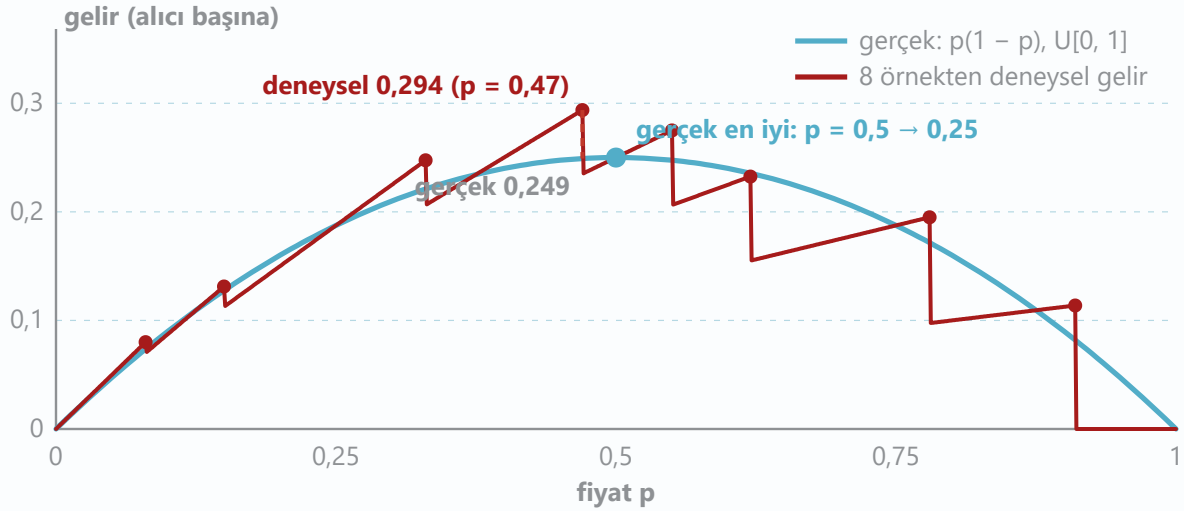
Adım 12 / 12

```
1  def en_iyi_fiyat(orn):
2    v = sorted(orn, reverse=True)
3    n = len(v)
4    fiyat, gelir = 0, 0
5    for k in range(1, n + 1):
6      p = v[k - 1]        # k. en yüksek
7      g = p * k / n       # k alıcı alır
8      if g > gelir:
9        fiyat, gelir = p, g
▶    return fiyat, gelir
```

DEĞİŞKENLER

`fiyat = 0.47` `gelir = 0.294`

Deneysel Gelir Eğrisi Gerçeğinden Ne Kadar Sapar?



- Deneysel eğri, örnek değerlerde sıçrayan basamaklı bir fonksiyondur; en iyisi her zaman bir örnek değerde çıkar (burada 0,47).
- Seçilen fiyatın örnekteki başarısı (0,294) gerçek başarısını (0,249) abartır: ölçülen ile seçilen aynı veri olduğu için **iyimser yanlılık** oluşur. Gerçek en iyi 0,25 ile fark küçüktür ama örnek azaldıkça büyür.

Kaç Örnek Yeter? Örneklem Karmaşıklığı

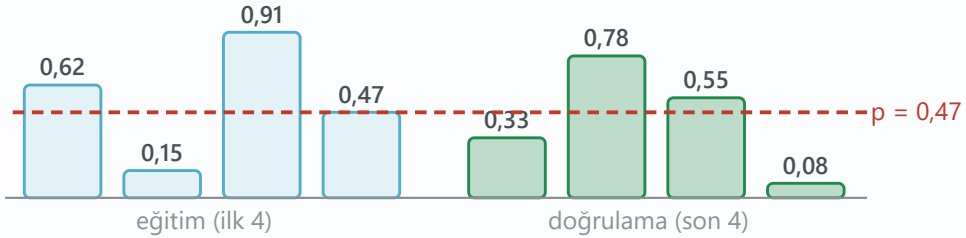
örnek n	deneysel gelir	gerçek gelir	0,25 − gerçek
10	0,311	0,224	0,0262
100	0,263	0,245	0,005
1.000	0,253	0,249	0,0011
10.000	0,25	0,25	0,0002

Değerler U[0, 1]; her n için yüzlerce tekrarın ortalaması (betikte simülasyon). En iyi gelir 0,25 (p = 0,5).

- **Aşırı uyum**: az örnekte deneysel gelir gerçeğinden yüksek görünür; öğrenilen fiyat örneklere "ezberlenmiştir".
- Örneklem karmaşıklığı: en iyi gelire istenen yakınlıkta bir fiyat için (yüksek olasılıkla) kaç örnek gerekir?
- Değer dağılımı çok uç değerler üretmiyorsa, iyi bir fiyat için gereken örnek sayısı teklifçi sayısı ve istenen doğrulukla ölçülü (polinom) biçimde artar (Cole ve Roughgarden, 2014). Uç değerli dağılımlarda böyle bir garanti verilemeyebilir.

Ders: veriden öğrenilen mekanizma, az veride kendi başarısını abartır. Eğitim ve değerlendirme verisini ayırmak, makine öğrenmesindeki kadar burada da gereklidir.

Seçmek ve Ölçmek İçin Ayrı Veri



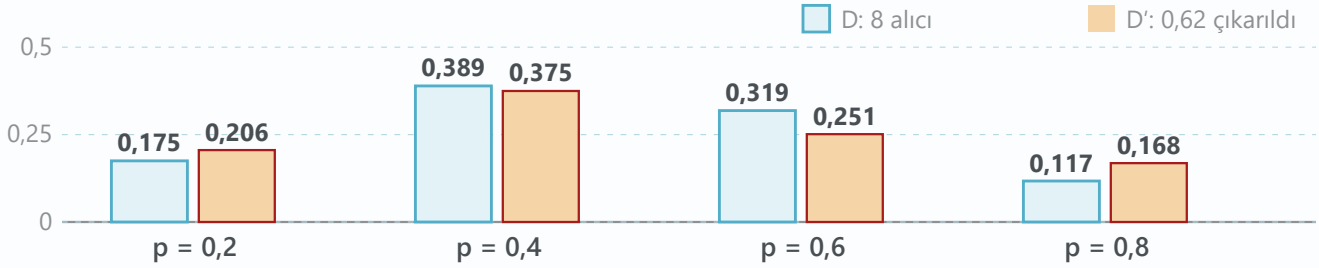
Ölçüm (p = 0,47)	gelir
eğitimde (seçildiği veri): 0,47·3/4	0,353
doğrulamada: 0,78 ve 0,55 alır → 0,47·2/4	0,235
gerçek, U[0, 1]: 0,47·0,53	0,249

- Eğitim ölçümü iyimserdir (0,353); doğrulama ölçümü yanlı değildir ve gerçeğe yakındır.
- Roller değişince: eğitim = son 4 → p = 0,55, ilk 4'te gelir 0,275. **Çapraz doğrulama** iki tahmini ortalar: (0,235 + 0,275)/2 = 0,255.
- Bedeli: fiyat yalnız 4 örnekle seçildiği için daha gürültülüdür.

Mekanizmaya özgü risk: alıcılar geçmiş tekliflerinin gelecekteki fiyatı belirlediğini bilirse bugün düşük teklif verebilir. Veri de stratejik olabilir.

Diferansiyel Mahremiyet: Tek Kişinin Verisi Sonucu Az Etkilesin

- **Diferansiyel mahremiyet**: tek bir kişinin verisi eklense ya da çıkarılsa da her sonucun olasılığı ancak biraz değişir. Buradaki ε, mahremiyetin ölçüsüdür: küçük ε, daha çok mahremiyet (en fazla e^{ε} kat değişim).
- **Üstel mekanizma** (McSherry ve Talwar, 2007): en iyi fiyatı kesin seçmek yerine bir kura çeker; iyi fiyatlar üstel olarak daha yüksek, kötü fiyatlar küçük ama sıfır olmayan olasılık alır.
- **Oyun kuramı bağı**: tek kişinin raporu sonucu az etkilediği için yalan söylemek de az kazandırır → *yaklaşık* teşvik uyumluluğu, üstelik mahremiyetle birlikte.



Rezerv fiyatı örneği, ε = 2: bir alıcı çıkarılınca olasılıklar en fazla 1,44 kat değişir (sınır $e^2 \approx 7{,}39$).

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



Veri: 8 alıcının değeri (veriden fiyat öğrenme örneğindeki değerler). Aday rezerv fiyatları 0,2 – 0,8. Δ: bir kişi çıkarılınca bir fiyatın puanının en çok ne kadar değişebileceği; fiyatlar 1'den küçük olduğu için Δ = 1.

Adım 1 / 9

```
1  import math
2  def ustel(v, fiyatlar, eps):
3      R = [p * sum(x >= p for x in v) for p in fiyatlar]
4      w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
5      W = sum(w)
6      return [x / W for x in w]
▶ v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
▶ P = [0.2, 0.4, 0.6, 0.8]
9  q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
11 q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

DEĞİŞKENLER

v = `8 alıcı`

P = `[0.2, 0.4, 0.6, 0.8]`

ÇIKTI

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



R = p·(alıcı sayısı): 0,2·6 = 1,20, 0,4·5 = 2,00, 0,6·3 = 1,80, 0,8·1 = 0,80.

Adım 2 / 9

```
1  import math
2  def ustel(v, fiyatlar, eps):
▶      R = [p * sum(x >= p for x in v) for p in fiyatlar]
4      w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
5      W = sum(w)
6      return [x / W for x in w]
7  v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
8  P = [0.2, 0.4, 0.6, 0.8]
▶  q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
11 q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

DEĞİŞKENLER

`n = 8`

`R = [1.20, 2.00, 1.80, 0.80]`

ÇIKTI

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



w = exp(ε·R/2) = exp(R): 3,32, 7,39, 6,05, 2,23. Farklar üstel büyür ama kötü fiyatın da şansı kalır.

Adım 3 / 9

```
1 import math
2 def ustel(v, fiyatlar, eps):
3     R = [p * sum(x >= p for x in v) for p in fiyatlar]
▶     w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
5     W = sum(w)
6     return [x / W for x in w]
7 v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
8 P = [0.2, 0.4, 0.6, 0.8]
9 q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
11 q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

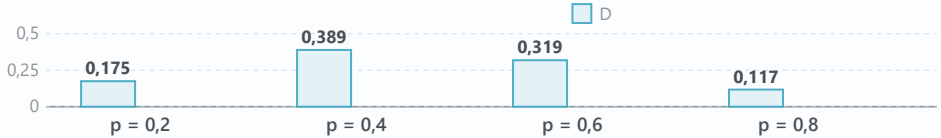
DEĞİŞKENLER

R = [1.20, 2.00, 1.80, 0.80]

w = [3.32, 7.39, 6.05, 2.23]

ÇIKTI

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



W = 18,98; olasılıklar q = w/W = 0,175, 0,389, 0,319, 0,117.

Adım 4 / 9

```
1  import math
2  def ustel(v, fiyatlar, eps):
3      R = [p * sum(x >= p for x in v) for p in fiyatlar]
4      w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
▶      W = sum(w)
▶      return [x / W for x in w]
7  v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
8  P = [0.2, 0.4, 0.6, 0.8]
9  q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
11 q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

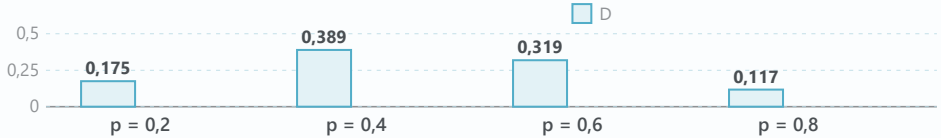
DEĞİŞKENLER

W = 18.984

q = [0.17, 0.39, 0.32, 0.12]

ÇIKTI

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



Çıktı yazdırılır. p = 0,4 en olası (%38,9) ama tek başına değil.

Adım 5 / 9

```python
import math
def ustel(v, fiyatlar, eps):
    R = [p * sum(x >= p for x in v) for p in fiyatlar]
    w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
    W = sum(w)
    return [x / W for x in w]
v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
P = [0.2, 0.4, 0.6, 0.8]
q = ustel(v, P, 2)
print([round(x, 3) for x in q])
q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
print([round(x, 3) for x in q])
```

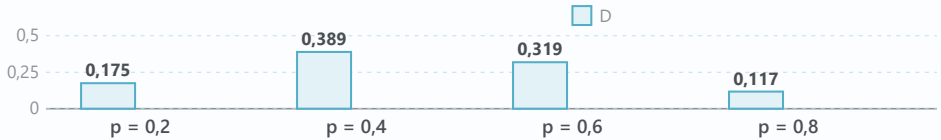
DEĞİŞKENLER

`q = [0.17, 0.39, 0.32, 0.12]`

ÇIKTI

```
[0.175, 0.389, 0.319, 0.117]
```

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



R = p·(alıcı sayısı): 0,2·5 = 1,00, 0,4·4 = 1,60, 0,6·2 = 1,20, 0,8·1 = 0,80. 0,62 gidince p = 0,6'nın geliri 0,6 düştü.

Adım 6 / 9

```
1  import math
2  def ustel(v, fiyatlar, eps):
▶      R = [p * sum(x >= p for x in v) for p in fiyatlar]
4      w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
5      W = sum(w)
6      return [x / W for x in w]
7  v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
8  P = [0.2, 0.4, 0.6, 0.8]
9  q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
▶  q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

DEĞİŞKENLER

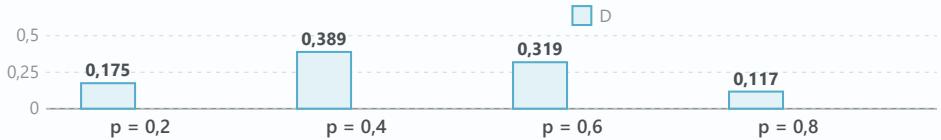
n = 7

R = [1.00, 1.60, 1.20, 0.80]

ÇIKTI

```
[0.175, 0.389, 0.319, 0.117]
```

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



w = exp(ε·R/2) = exp(R): 2,72, 4,95, 3,32, 2,23. Farklar üstel büyür ama kötü fiyatın da şansı kalır.

Adım 7 / 9

```
1 import math
2 def ustel(v, fiyatlar, eps):
3     R = [p * sum(x >= p for x in v) for p in fiyatlar]
▶     w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
5     W = sum(w)
6     return [x / W for x in w]
7 v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
8 P = [0.2, 0.4, 0.6, 0.8]
9 q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
11 q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

DEĞİŞKENLER

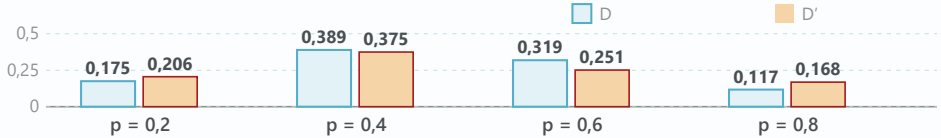
R = [1.00, 1.60, 1.20, 0.80]

w = [2.72, 4.95, 3.32, 2.23]

ÇIKTI

```
[0.175, 0.389, 0.319, 0.117]
```

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



W = 13,22; olasılıklar q = w/W = 0,206, 0,375, 0,251, 0,168.

Adım 8 / 9

```
1 import math
2 def ustel(v, fiyatlar, eps):
3     R = [p * sum(x >= p for x in v) for p in fiyatlar]
4     w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
▶     W = sum(w)
▶     return [x / W for x in w]
7 v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
8 P = [0.2, 0.4, 0.6, 0.8]
9 q = ustel(v, P, 2)
10 print([round(x, 3) for x in q])
11 q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
12 print([round(x, 3) for x in q])
```

DEĞİŞKENLER

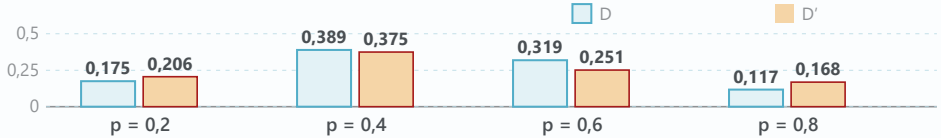
W = 13.217

q = [0.21, 0.37, 0.25, 0.17]

ÇIKTI

```
[0.175, 0.389, 0.319, 0.117]
```

İleri Okuma · Kod İzleme: Üstel Mekanizma ile Fiyat Seçmek



İki dağılım yan yana: en büyük oran 1,44 ≤ e^2 = 7,39. Tek kişinin verisi sonucu sınırlı etkiler: ε-diferansiyel mahremiyet.

Adım 9 / 9

```python
import math
def ustel(v, fiyatlar, eps):
    R = [p * sum(x >= p for x in v) for p in fiyatlar]
    w = [math.exp(eps * r / 2) for r in R]   # Δ = 1
    W = sum(w)
    return [x / W for x in w]
v = [0.62, 0.15, 0.91, 0.47, 0.33, 0.78, 0.55, 0.08]
P = [0.2, 0.4, 0.6, 0.8]
q = ustel(v, P, 2)
print([round(x, 3) for x in q])
q = ustel(v[1:], P, 2)          # 0.62 çıkarıldı
print([round(x, 3) for x in q])
```

DEĞİŞKENLER

`q = [0.21, 0.37, 0.25, 0.17]`

ÇIKTI

```
[0.175, 0.389, 0.319, 0.117]
[0.206, 0.375, 0.251, 0.168]
```

Blokzincir ve Akıllı Sözleşmeler

- **Akıllı sözleşme**: kodu herkese açık, zincirde otomatik yürütülen program → kendi kendini uygulayan bir mekanizma.
- Mekanizma tasarımının "tasarımcı kurallara bağlı kalır" (taahhüt) varsayımı kodla sağlanır; ama bekleyen işlemler de herkese görünür.
- Blok üreticisi hangi işlemin bloğa gireceğine ve **sırasına** karar verir: kıt kaynak → açık artırma.



- **İşlem ücreti açık artırması**: Kısıtlı blok alanı teklif verenlere dağıtılır.
- **Merkeziyetsiz borsa**: Otomatik piyasa yapıcı (AMM): fiyat bir formülle belirlenir.

- **Zincir üstü yönetişim**: Jeton (token) ağırlıklı oylama; oylama kurallarının sorunları burada da geçerlidir.

İşlem Ücreti Açık Artırmaları

Birinci fiyat (ilk tasarım)

- Her işlem bir ücret teklif eder; üretici en yüksek teklifleri alır, herkes kendi teklifini öder.
- Değerler 9, 7, 6, 4, 2 ve kapasite 3: dürüst teklifle 9 + 7 + 6 = 22 öder.
- Ama dürüst teklif en iyi değil: herkes teklifini kırpar, "ne kadar vermeliyim?" sorusu kullanıcıyı yorar.

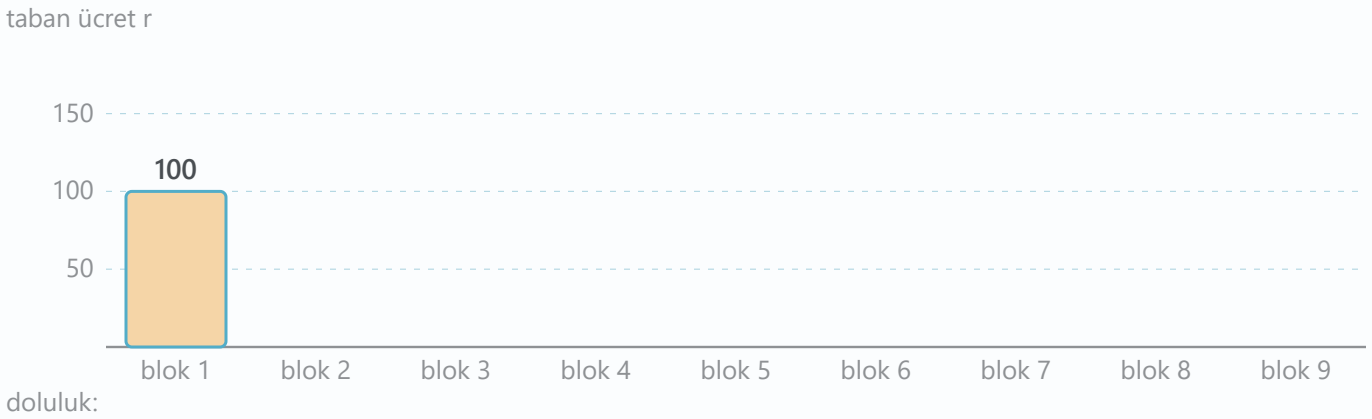
Taban ücret + bahşiş (EIP-1559 türü)

- Protokol bir taban ücret r belirler (burada 5); r **yakılır**, yani kimseye ödenmez, dolaşımdan çıkar. Üreticiye yalnız bahşiş (1) gider.
- Değeri ≥ r + bahşiş olanlar girer: 3 işlem, 15 yakılır, 3 bahşiş.
- Talep kapasiteyi aşmıyorsa en iyi strateji basittir: "r + küçük bahşiş" öde; fiyat tahmini gerekmez.



Tasarım hedefleri (Roughgarden, 2021): kullanıcı için basit teklif, üretici için dürüst davranış, danışıklılığa direnç. Hepsini tam sağlayan kural bulmak zordur.

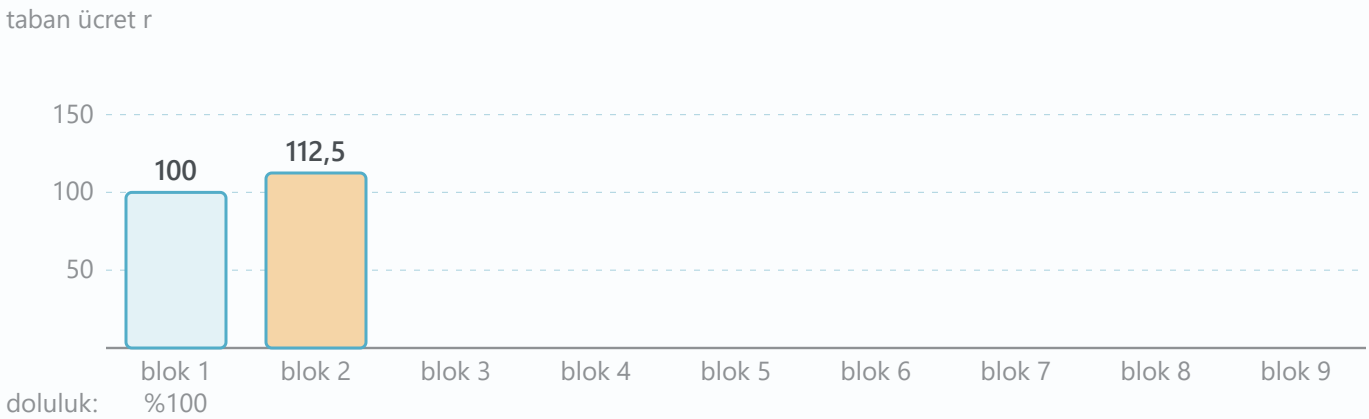
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Genel ilke: doluluk hedefin üstündeyse ücret artar, altındaysa düşer. EIP-1559'un ilk parametreleriyle: $r_{t+1} = r_t \cdot (1 + \frac{1}{8} \cdot (g_t - g_{hedef})/g_{hedef})$, g_{hedef} = blok kapasitesinin yarısı. Başlangıç r = 100.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

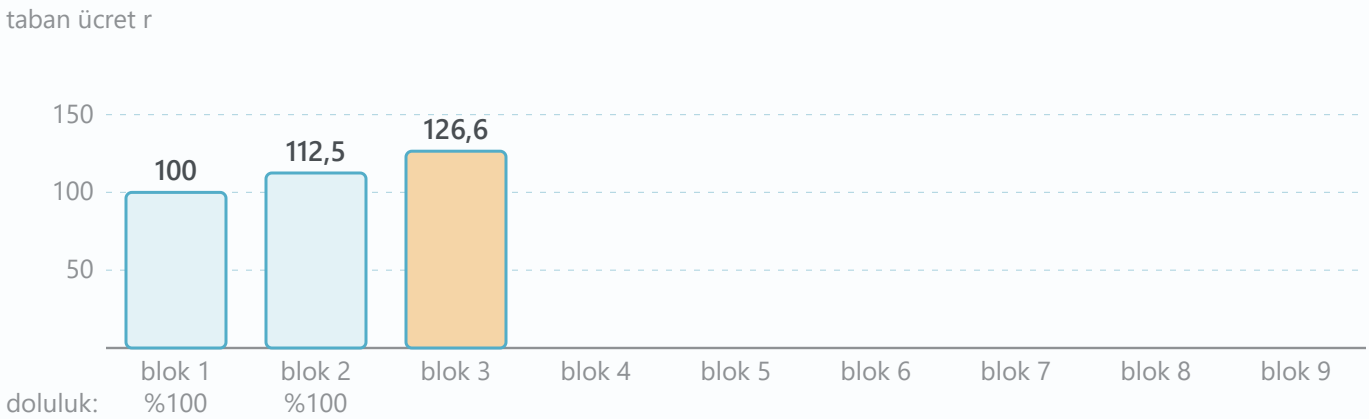
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 1 %100 dolu → değişim +%12,5 → r = **112,5**.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

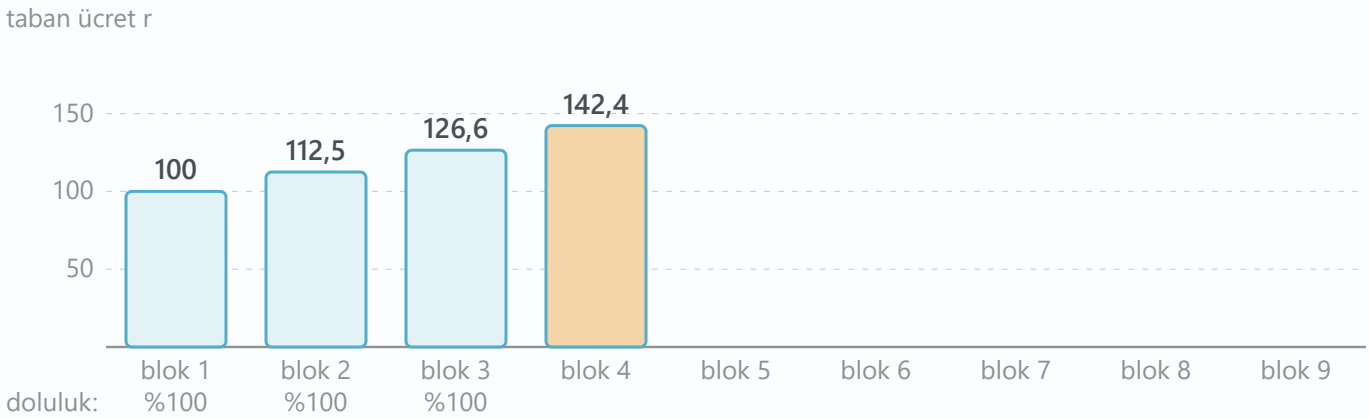
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 2 %100 dolu → değişim +%12,5 → r = **126,6**.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

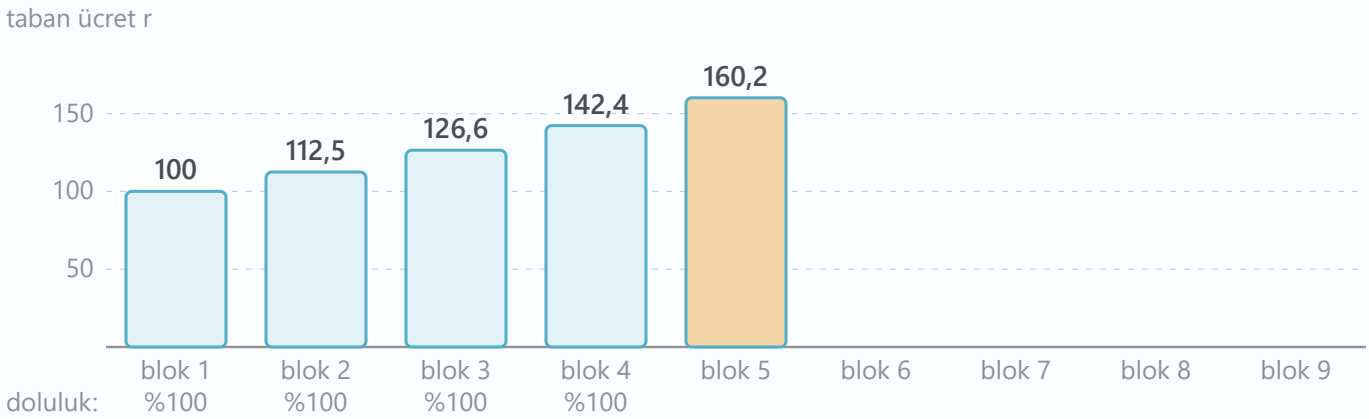
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 3 %100 dolu → değişim +%12,5 → r = **142,4**.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

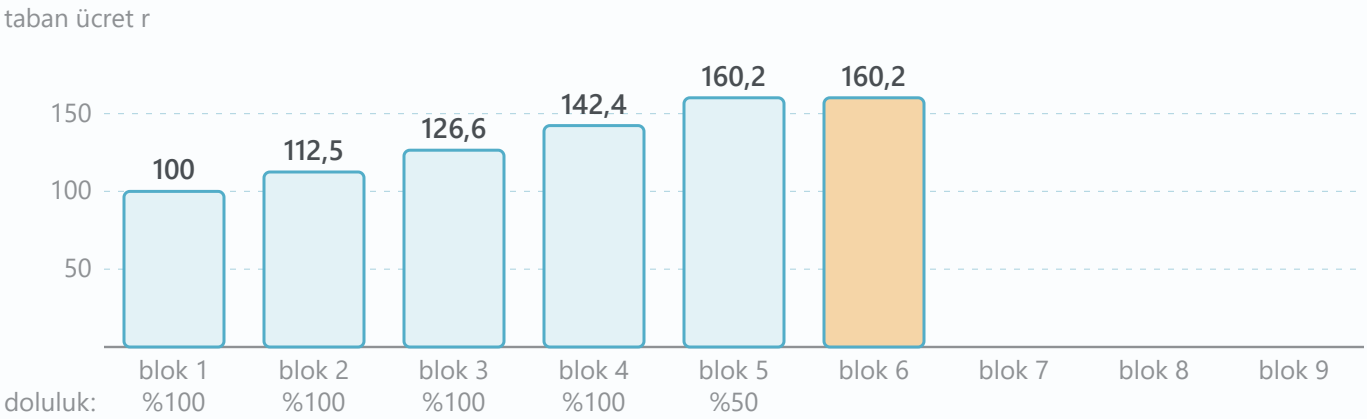
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 4 %100 dolu → değişim +%12,5 → r = **160,2**. Talep patlaması: ücret her blokta en çok %12,5 artar.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

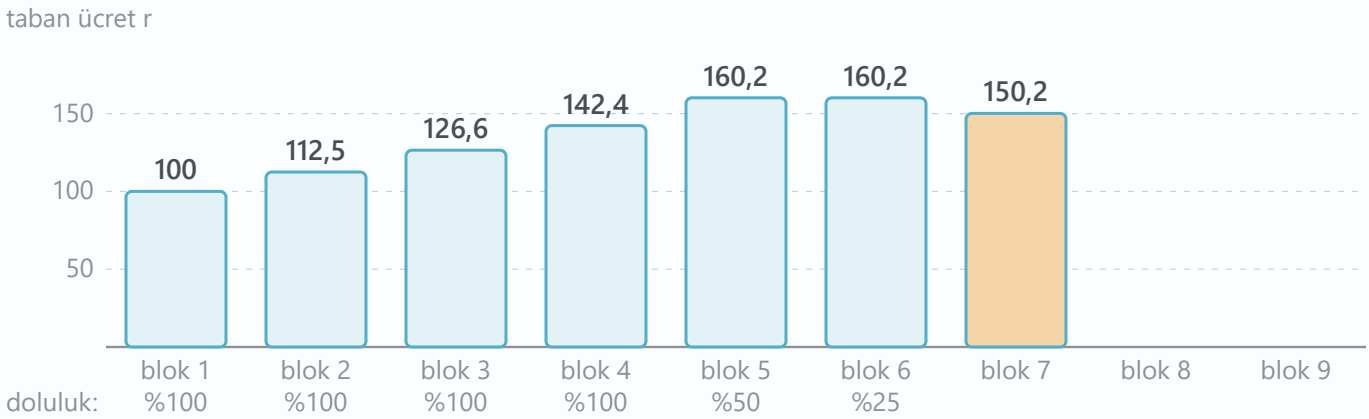
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 5 %50 dolu → değişim ±%0 → r = **160,2**. Tam hedefte ücret sabit kalır.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

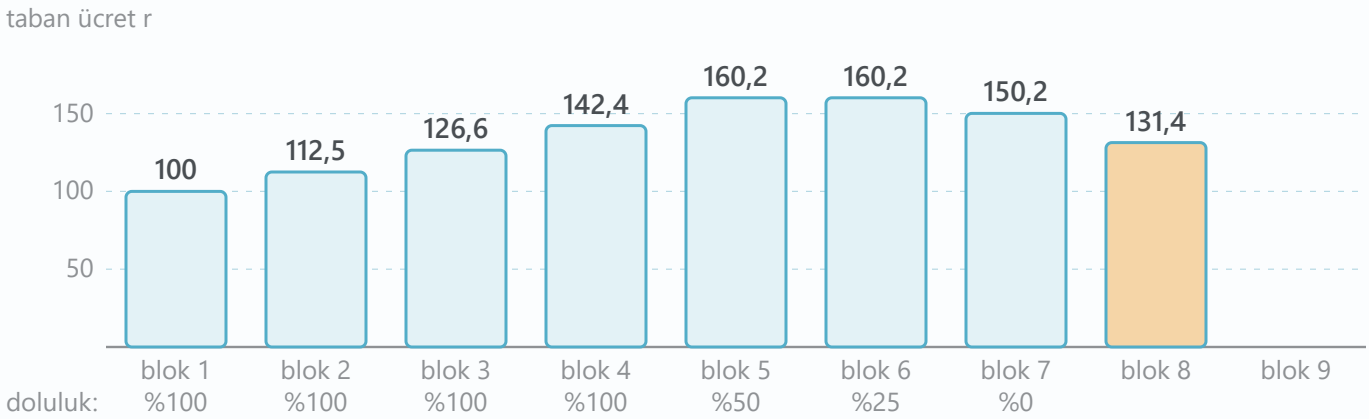
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 6 %25 dolu → değişim −%6,25 → r = **150,2**.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

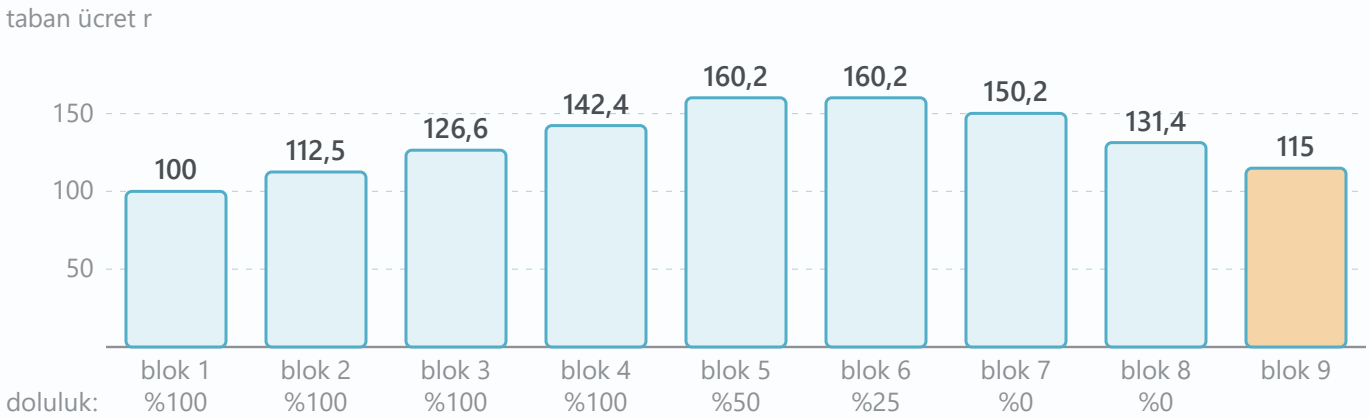
Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 7 %0 dolu → değişim −%12,5 → r = **131,4**.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

Adım Adım: Taban Ücret Talebe Göre Ayarlanır



Blok 8 %0 dolu → değişim −%12,5 → r = **115**. Boş bloklar ücreti düşürür; talep azalınca fiyat geri gelir.

Taban ücret talebi izleyen bir “ilan edilmiş fiyat” gibi çalışır; blok doluyken yine bahşişe dayalı açık artırmaya döner.

Neden Yakılır? İyi Bir Ücret Mekanizmasının Özellikleri

Kullanıcı teşvik uyumu
Talep kapasiteyi aşmıyorsa taban ücret + küçük bahşiş en iyi tekliftir; kullanıcı "ne kadar kırpmalıyım?" diye düşünmez.

Üretici teşvik uyumu
Üretici kurala uyarak en çok kazanır: sahte işlem eklemek ya da işlem saklamak kazandırmaz.

Danışıklılığa dayanıklılık
Üretici ve kullanıcılar zincir dışında anlaşıp ortak kazançlarını artıramaz.



- Taban ücret üreticiye gitseydi sahte işlemler **bedava** olurdu (üretici kendine öder) ve ücreti istediği gibi şişirebilirdi.
- Yakılınca her sahte işlem r kadar gerçek kayıptır: 5 sahte işlem 5r yakar.
- Bedava değil ama sınırsız da değil: tam blokta bahşiş yarışı (birinci fiyat) geri gelir.

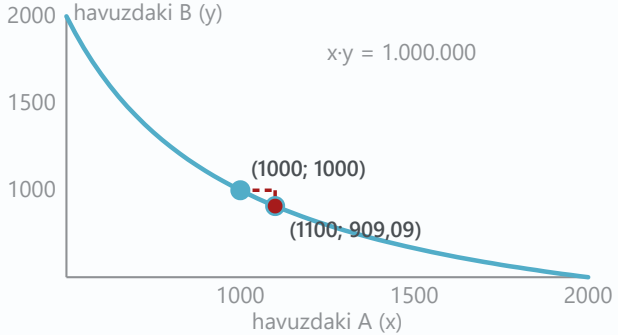
Bu üç özellik Roughgarden (2021) ile biçimselleşti; blok alanı kıtken üçü birlikte sağlanamaz (yan sözleşme tanımıyla Chung ve Shi, 2023; Roughgarden'ın tanımıyla Chung, Roughgarden ve Shi, 2024).

Otomatik Piyasa Yapıcı: Sabit Çarpım Kuralı

- Havuzda x A ve y B var; her işlem sonrası **x·y = k** korunur. Karşı taraf bir insan değil, formüldür.
- Δx A satan kullanıcı Δy B alır: (x + Δx)·(y − Δy) = x·y ⇒ Δy = y·Δx / (x + Δx).
- Kayma: başlangıç fiyatına göre eksik alınan B oranı (10 A için 10 B yerine 9,90 B → %1).

Δx (A)	alınan B	kayma
10	9,90	%1,0
100	90,91	%9,1
250	200,00	%20,0
500	333,33	%33,3

Havuz (1000; 1000), başlangıç fiyatı 1 A = 1 B. %0,3 işlem ücretiyle 100 A → 90,66 B; ücret likidite sağlayanlara kalır.



- Büyük işlem fiyatı kendi aleyhine kaydırır (**fiyat etkisi**); 100 A sonrası 1 B = 1,21 A.
- Dış piyasayla fark açılınca arbitrajcılar havuzu geri dengeler.
- Fiyat etkisi önceden hesaplanabildiği için öne geçme kârlıdır.

Adım Adım: MEV ve Sandviç Saldırısı

Sıra	İşlem	Havuz (A, B)	Sonuç
0	Başlangıç (x·y = 1.000.000)	(1000; 1000)	

Sabit çarpımlı havuz: A satınca B azalır, fiyat kayar. Kullanıcı 100 A ile tek başına 90,91 B alırdı.

Kullanıcının işlemi bloğa girmeden önce herkese görünür (bekleme havuzu, mempool).

Adım Adım: MEV ve Sandviç Saldırısı

Sıra	İşlem	Havuz (A, B)	Sonuç
0	Başlangıç (x·y = 1.000.000)	(1000; 1000)	
1	Saldırgan 50 A satar (öne geçme)	(1050; 952,38)	47,62 B alır

Sırayı belirleyebilen biri kendi işlemini kullanıcının **önüne** koyar: B'yi ucuzken alır, fiyatı yukarı iter.

Öne geçme (front-running).

Adım Adım: MEV ve Sandviç Saldırısı

Sıra	İşlem	Havuz (A, B)	Sonuç
0	Başlangıç (x·y = 1.000.000)	(1000; 1000)	
1	Saldırgan 50 A satar (öne geçme)	(1050; 952,38)	47,62 B alır
2	Kullanıcı 100 A satar	(1150; 869,57)	82,82 B alır (tek başına 90,91)

Kullanıcı artık kötü fiyattan işlem yapar: 82,82 B → **8,09 B kayıp**.

Kullanıcının işlemi fiyatı daha da yukarı iter.

Adım Adım: MEV ve Sandviç Saldırısı

Sıra	İşlem	Havuz (A, B)	Sonuç
0	Başlangıç (x·y = 1.000.000)	(1000; 1000)	
1	Saldırgan 50 A satar (öne geçme)	(1050; 952,38)	47,62 B alır
2	Kullanıcı 100 A satar	(1150; 869,57)	82,82 B alır (tek başına 90,91)
3	Saldırgan B'leri geri satar (arkadan gelme)	(1090,29; 917,18)	59,71 A alır

Saldırgan B'leri pahalıdan geri satar: 59,71 A → **kâr 9,71 A** (ücretler hariç).

Arkadan gelme (back-running): “sandviç” tamamlandı.

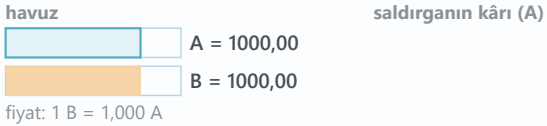
Adım Adım: MEV ve Sandviç Saldırısı

Sıra	İşlem	Havuz (A, B)	Sonuç
0	Başlangıç (x·y = 1.000.000)	(1000; 1000)	
1	Saldırgan 50 A satar (öne geçme)	(1050; 952,38)	47,62 B alır
2	Kullanıcı 100 A satar	(1150; 869,57)	82,82 B alır (tek başına 90,91)
3	Saldırgan B'leri geri satar (arkadan gelme)	(1090,29; 917,18)	59,71 A alır

MEV (maksimum elde edilebilir değer, maximal extractable value): işlemleri sıralama, ekleme ve çıkarma gücünden elde edilebilecek değer. Önlem fikirleri: toplu açık artırma (aynı fiyat), şifreli bekleme havuzu, sıralama hakkının açık artırılması.

Mekanizma tasarımı sorusu: sıralama gücü kimde, bedelini kim ödüyor?

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Kullanıcı 100 A satacak, en az 82 B istiyor (kayma toleransı). Saldırgan öne geçme miktarı a için üç değer dener.

Adım 1 / 12

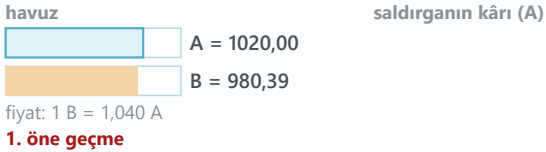
```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
▶  for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

DEĞİŞKENLER

`a = 20`

ÇIKTI

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



a = 20: saldırgan önce 20 A satar, 19,61 B alır. B azaldı → fiyat yükseldi.

Adım 2 / 12

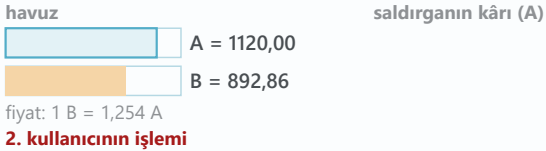
```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
▶      b1 = cikti(x, y, a)          # öne geç: a A sat
▶      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

DEĞİŞKENLER

`a = 20` `b1 = 19.61` `x = 1020.0` `y = 980.39`

ÇIKTI

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Kullanıcı 87,54 B alır (≥ 82, işlem geçer; tek başına 90,91 alırdı).

Adım 3 / 12

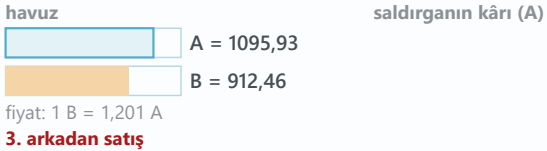
```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
▶      b2 = cikti(x, y, u)          # kullanıcı: u A sat
▶      if b2 < min_out:
7          return None              # işlem iptal
▶      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

DEĞİŞKENLER

`a = 20` `b2 = 87.54` `x = 1120.0` `y = 892.86`

ÇIKTI

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Saldırgan 19,61 B'yi geri satar, 24,07 A alır: kâr = 24,07 − 20 = **4,07 A**.

Adım 4 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
▶      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
▶      return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

DEĞİŞKENLER

`a = 20` `a2 = 24.07` `kar = 4.07`

ÇIKTI

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Sonuç yazdırılır.

Adım 5 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
▶ for a in [20, 50, 80]:
▶     print(a, sandvic(1000, 1000, a, 100, 82))
```

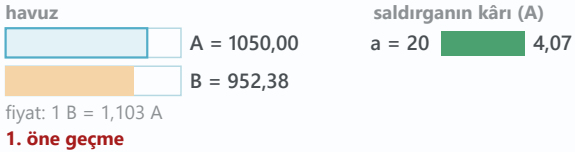
DEĞİŞKENLER

`a = 20` `sonuc = 4.07`

ÇIKTI

```
20 4.07
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



a = 50: saldırgan önce 50 A satar, 47,62 B alır. B azaldı → fiyat yükseldi.

Adım 6 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
▶      b1 = cikti(x, y, a)          # öne geç: a A sat
▶      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

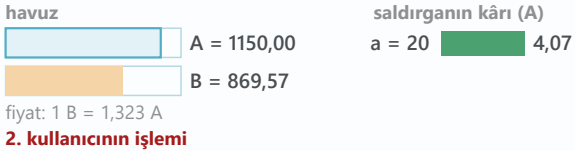
DEĞİŞKENLER

`a = 50` `b1 = 47.62` `x = 1050.0` `y = 952.38`

ÇIKTI

```
20 4.07
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Kullanıcı 82,82 B alır (≥ 82, işlem geçer; tek başına 90,91 alırdı).


```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
▶      b2 = cikti(x, y, u)          # kullanıcı: u A sat
▶      if b2 < min_out:
7          return None              # işlem iptal
▶      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

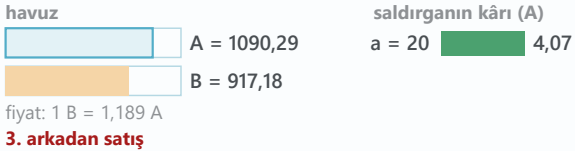
DEĞİŞKENLER

`a = 50` `b2 = 82.82` `x = 1150.0` `y = 869.57`

ÇIKTI

```
20 4.07
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Saldırgan 47,62 B'yi geri satar, 59,71 A alır: kâr = 59,71 − 50 = **9,71 A**.

Adım 8 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
▶      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
▶      return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

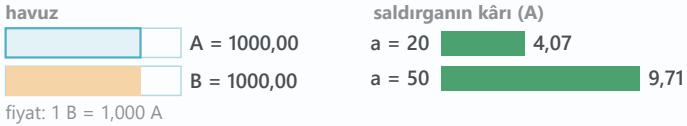
DEĞİŞKENLER

`a = 50` `a2 = 59.71` `kar = 9.71`

ÇIKTI

```
20 4.07
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Sonuç yazdırılır.

Adım 9 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
▶ for a in [20, 50, 80]:
▶     print(a, sandvic(1000, 1000, a, 100, 82))
```

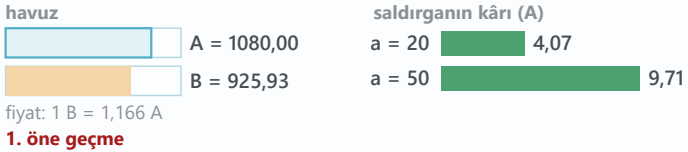
DEĞİŞKENLER

`a = 50` `sonuc = 9.71`

ÇIKTI

```
20 4.07
50 9.71
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



a = 80: saldırgan önce 80 A satar, 74,07 B alır. B azaldı → fiyat yükseldi.

Adım 10 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
▶      b1 = cikti(x, y, a)          # öne geç: a A sat
▶      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

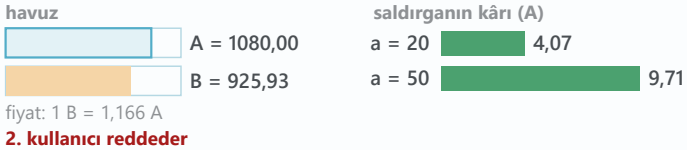
DEĞİŞKENLER

`a = 80` `b1 = 74.07` `x = 1080.0` `y = 925.93`

ÇIKTI

```
20 4.07
50 9.71
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Kullanıcı yalnızca 78,47 B alırdı < 82 → işlem **iptal**, saldırı başarısız (None).

Adım 11 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
▶      b2 = cikti(x, y, u)          # kullanıcı: u A sat
▶      if b2 < min_out:
▶          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
11 for a in [20, 50, 80]:
12     print(a, sandvic(1000, 1000, a, 100, 82))
```

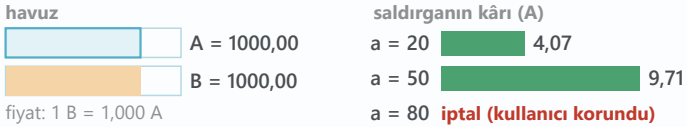
DEĞİŞKENLER

`a = 80` `b2 = 78.47` `min_out = 82`

ÇIKTI

```
20 4.07
50 9.71
```

Kod İzleme: Sandviç Saldırısı ve Kayma Sınırı



Daha büyük öne geçme daha çok kâr getirir, ama kullanıcının kayma sınırı onu keser: saldırganın en iyisi sınırı tam zorlayan a. Sıkı tolerans MEV'e karşı basit bir korumadır.

Adım 12 / 12

```
1  cikti = lambda x, y, dx: y - x * y / (x + dx)  # xy=k
2  def sandvic(x, y, a, u, min_out):
3      b1 = cikti(x, y, a)          # öne geç: a A sat
4      x, y = x + a, y - b1
5      b2 = cikti(x, y, u)          # kullanıcı: u A sat
6      if b2 < min_out:
7          return None              # işlem iptal
8      x, y = x + u, y - b2
9      a2 = cikti(y, x, b1)         # arkadan: b1 B sat
10     return round(a2 - a, 2)
▶ for a in [20, 50, 80]:
▶     print(a, sandvic(1000, 1000, a, 100, 82))
```

DEĞİŞKENLER

a = 80 sonuc = None

ÇIKTI

```
20 4.07
50 9.71
80 None
```

MEV'ye Karşı Önlemler

Yöntem	Fikir	Bedeli
Sıkı kayma toleransı	En az alınacak miktar; büyük öne geçme işlemi iptal ettirir	İşlem daha sık geri çevrilir
Taahhüt–açıklama	Önce işlemin özeti (hash) yayımlanır, içerik sıra kesinleşince açılır	İki adım ve gecikme; açılmayan işlemler
Şifreli bekleme havuzu	İşlem, sırası belirlendikten sonra çözülür	Anahtarı tutanlara güven gerekir
Toplu açık artırma	Bir zaman dilimindeki tüm emirler **aynı fiyattan** eşleşir; sıra önemsizleşir	Dilim kadar bekleme
Sıralama hakkını açık artırmak	Blok kurma hakkı açıkça satılır; MEV geliri gizli değil rekabetçi dağılır	MEV yok olmaz; sıralama az sayıda büyük blok kurucusunda toplanır

Bu önlemlerin uygulamadaki biçimleri sık değişir; kalıcı fikir, sıralama gücünün değer taşıması ve bu değerin ya azaltılması ya da açıkça fiyatlanmasıdır.

Toplu açık artırmada sandviç işe yaramaz: saldırganın "önden alış" ve "arkadan satış" emirleri kullanıcınınkiyle aynı partide, **aynı fiyattan** eşleşir; alış ile satış fiyatı aynı olduğu için kâr 0'dır.

Hesaplama ve Ölçek

- Büyük oyunlarda denge bulmak zor olabilir (PPAD) → **yaklaşık denge** ve öğrenme dinamikleri.
- Çok oyunculu pokerde insan üstü sonuçlar pişmanlığı en aza indirme ve arama ile elde edildi (Brown ve Sandholm, 2019).
- Kombinatoryal açık artırmada kazananı belirlemek NP-zordur; teklif dili ve yaklaşık algoritmalar gerekir.
- Kuramsal garanti kadar gecikme, veri erişimi ve enerji maliyeti de önemlidir.

nesne m	paket sayısı $2^m - 1$
5	31
10	1.023
20	1.048.575
30	1.073.741.823

Her teklifçi her pakete değer bildirirse girdi üstel büyür. İki eylemli n oyunculu oyunda strateji profili 2^n: n = 20 için 1.048.576.

Kombinatoryal Açık Artırmada Kazanan Belirleme

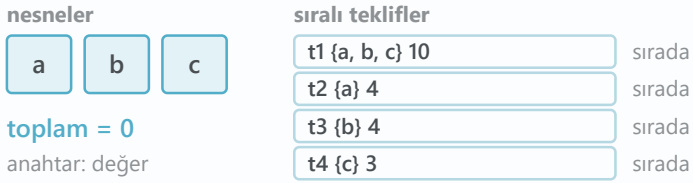
- Her teklif bir paket ve değer: (S_i, v_i). Birbiriyle **çakışmayan** teklifleri seçip toplam değeri maksimize etmek: **kazanan belirleme problemi**.
- Ağırlıklı küme paketlemedir: NP-zor ve genel durumda iyi yaklaştırılamaz.
- Kaba kuvvet n teklif için 2^n alt kümeyi dener; pratikte bu aramayı akıllıca budayan çözücüler kullanılır.
- VCG ödemeleri için problem n + 1 kez çözülür (her teklifçi bir kez çıkarılarak). Yaklaşık çözülürse VCG'nin doğruluk garantisi bozulabilir.

Teklif	paket	değer
t1	{a, b, c}	10
t2	{a}	4
t3	{b}	4
t4	{c}	3

Olurlu seçim	toplam
{t1}	10
{t2, t3}	8
{t2, t3, t4}	11

t1 diğer üçüyle çakışır; optimum 11.

Kod İzleme: Açgözlü Kazanan Belirleme



Değere göre sıralama: t1 (10), t2 (4), t3 (4), t4 (3). Toplam 0, kullanılan nesne yok.

Adım 1 / 11

```
1  def acgozlu(teklifler, anahtar):
▶      sirali = sorted(teklifler, key=anahtar, reverse=True)
▶      toplam, kullanilan = 0, set()
4      for S, v in sirali:
5          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
▶  print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

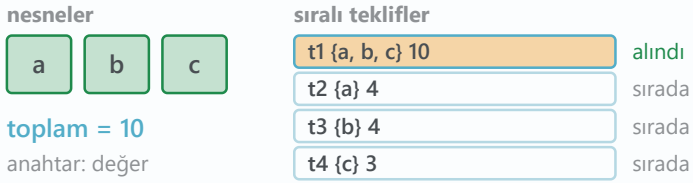
DEĞİŞKENLER

`sirali = [t1, t2, t3, t4]`

`toplam = 0` `kullanilan = {}`

ÇIKTI

Kod İzleme: Açgözlü Kazanan Belirleme



t1 = {a, b, c}: hiçbir nesne kullanılmamış → alınır. Toplam 10; üç nesne de doldu.

Adım 2 / 11

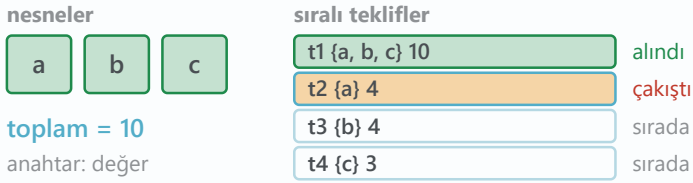
```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
▶              toplam += v
▶              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

DEĞİŞKENLER

`S = {a,b,c}` `v = 10` `toplam = 10` `kullanilan = {a,b,c}`

ÇIKTI

Kod İzleme: Açgözlü Kazanan Belirleme



t2 = {a}: a zaten kullanıldı → kesişim boş değil, atlanır.

Adım 3 / 11

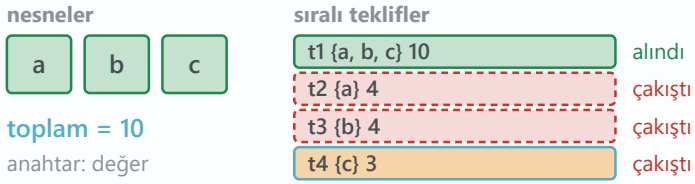
```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

DEĞİŞKENLER

`S = {a}` `v = 4` `toplam = 10` `kullanilan = {a,b,c}`

ÇIKTI

Kod İzleme: Açgözlü Kazanan Belirleme



t3 ve t4 de çakışır. Büyük paket erken alındığı için küçük tekliflerin hepsi dışarıda kaldı.

Adım 4 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

DEĞİŞKENLER

`S = {c}` `v = 3` `toplam = 10` `kullanilan = {a,b,c}`

ÇIKTI

Kod İzleme: Açgözlü Kazanan Belirleme



Sonuç **10**; optimum 11 idi. Değere göre açgözlü seçim büyük paketlere aldanır.

Adım 5 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
4      for S, v in sirali:
5          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
▶      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
▶  print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

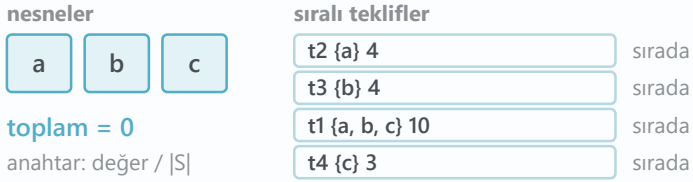
DEĞİŞKENLER

`toplam = 10`

ÇIKTI

```
10
```

Kod İzleme: Açgözlü Kazanan Belirleme



Nesne başına değer: t2 ve t3 (4), t1 (10/3 ≈ 3,33), t4 (3). Eşitlikte sorted ilk sırayı korur.

Adım 6 / 11

```
1  def acgozlu(teklifler, anahtar):
▶      sirali = sorted(teklifler, key=anahtar, reverse=True)
▶      toplam, kullanilan = 0, set()
4      for S, v in sirali:
5          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
▶  print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

DEĞİŞKENLER

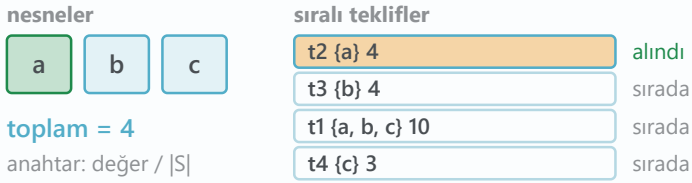
`sirali = [t2, t3, t1, t4]`

`toplam = 0` `kullanilan = {}`

ÇIKTI

```
10
```

Kod İzleme: Açgözlü Kazanan Belirleme



t2 = {a} alınır: toplam 4.

Adım 7 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
▶              toplam += v
▶              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

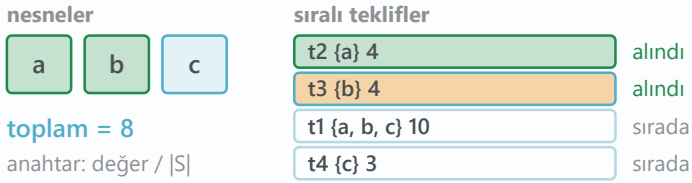
DEĞİŞKENLER

`S = {a}` `v = 4` `toplam = 4` `kullanilan = {a}`

ÇIKTI

```
10
```

Kod İzleme: Açgözlü Kazanan Belirleme



t3 = {b} alınır: toplam 8.

Adım 8 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
▶              toplam += v
▶              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

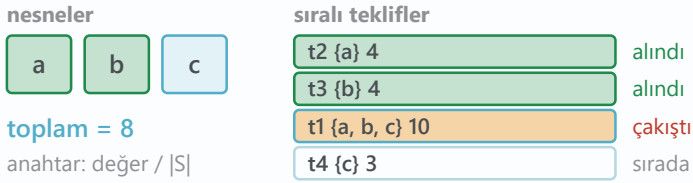
DEĞİŞKENLER

`S = {b}` `v = 4` `toplam = 8` `kullanilan = {a,b}`

ÇIKTI

```
10
```

Kod İzleme: Açgözlü Kazanan Belirleme



t1 = {a, b, c}: a ve b dolu → atlanır (değeri 10 olsa da).

Adım 9 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

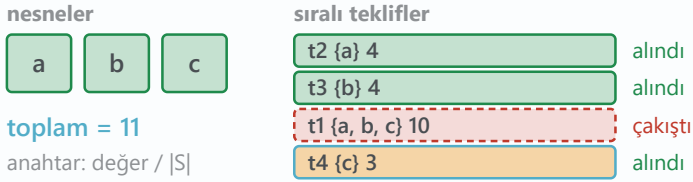
DEĞİŞKENLER

`S = {a,b,c}` `v = 10` `toplam = 8` `kullanilan = {a,b}`

ÇIKTI

```
10
```

Kod İzleme: Açgözlü Kazanan Belirleme



t4 = {c} boşta → alınır: toplam 11.

Adım 10 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
▶      for S, v in sirali:
▶          if not (S & kullanilan):  # çakışma yok
▶              toplam += v
▶              kullanilan |= S
8      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
11 print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

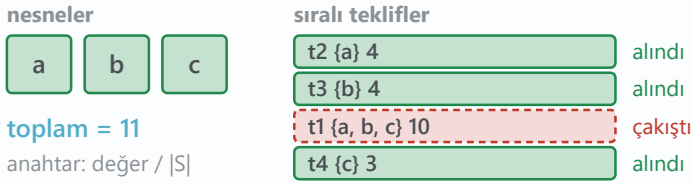
DEĞİŞKENLER

`S = {c}` `v = 3` `toplam = 11` `kullanilan = {a,b,c}`

ÇIKTI

```
10
```

Kod İzleme: Açgözlü Kazanan Belirleme



Sonuç **11** = optimum. Ama bu anahtar da yanılabilir: ({a, b}, 10) ve ({a}, 6) için önce 6'yı alır. Değer/√|S| ile sıralamak en kötü durumda √m kat garanti verir (Lehmann ve diğ., 2002).

Adım 11 / 11

```
1  def acgozlu(teklifler, anahtar):
2      sirali = sorted(teklifler, key=anahtar, reverse=True)
3      toplam, kullanilan = 0, set()
4      for S, v in sirali:
5          if not (S & kullanilan):  # çakışma yok
6              toplam += v
7              kullanilan |= S
▶      return toplam
9  T = [({'a','b','c'}, 10), ({'a'}, 4), ({'b'}, 4), ({'c'}, 3)]
10 print(acgozlu(T, lambda t: t[1]))
▶  print(acgozlu(T, lambda t: t[1] / len(t[0])))
```

DEĞİŞKENLER

`toplam = 11`

ÇIKTI

```
10
11
```

ÖLÇEK VE AÇIK SORULAR

Diğer Araştırma Yönleri

- **Sosyal ağlarda etki maksimizasyonu**: bir firma k kişiye ücretsiz ürün verecek; yayılmayı maksimize edecek k "tohum" kişi kimdir? (Kempe, Kleinberg ve Tardos, 2003)
- **Bağımsız çağlayan modeli**: yeni etkilenen her kişi, her komşusunu yalnızca bir kez, p olasılıkla etkiler.
- En iyi tohum kümesini bulmak NP-zordur; ama beklenen yayılım **azalan marjinal getirili** (alt modüler) olduğu için her adımda en çok katkı yapanı ekleyen açgözlü seçim, en iyinin yaklaşık **1 − 1/e ≈ %63**'ünü garanti eder.

Çok ajanlı pekiştirmeli öğrenme

Her ajanın ortamı öğrenen diğer ajanları da içerir; ortam durağan olmadığı için tek ajanlı yakınsama garantileri geçerli değildir.

Sürdürülebilir sistemler

Karbon emisyon izni piyasaları, elektrikte talep yanıtı: dışsallığı fiyatlayan mekanizmalar (Pigou vergisi).

Dayanıklılık

Saldırgan, hatalı ya da sınırlı rasyonel oyunculara rağmen iyi çalışan mekanizmalar.

Diğer yönler: bilgi tasarımı (Bayesçi ikna: bilgiyi açıklama kuralını seçerek alıcının kararını yönlendirmek), yapay zekâ hizalamasında insan tercihlerini toplama (bir sosyal seçim problemi), tahmin piyasaları ve doğrulanamayan raporları ödüllendiren eşler arası tahmin.

ÖLÇEK VE AÇIK SORULAR

Açık Araştırma Soruları

İnsan + yapay zekâ pazarları

İnsanlar ve ajanlar aynı pazarda etkileştiğinde kurallar nasıl değişmeli?

Ödünleşimleri ölçmek

Verimlilik, mahremiyet ve adalet arasındaki ödünleşimler nasıl ölçülmeli?

Uyarlanabilir dayanıklılık

Stratejiye dayanıklı mekanizmalar değişen davranışlara nasıl uyarlanır?

Kuram ve veri

Modellerin varsayımları gerçek platform verileriyle nasıl sınanır?

Öğrenen ajanların dengesi

Birbirine karşı öğrenen algoritmalar hangi sonuca yakınsar; danışıklılık nasıl önlenir?

Doğrulanabilir mekanizma

Öğrenilmiş bir kuralın teşvik uyumlu olduğu nasıl kanıtlanır veya denetlenir?

DEĞERLENDİRME

Özet

Konu	Açıklama	Bu bölümdeki sayı
Ajanlar arası pazarlık	Geriye doğru çıkarım; sabır ve son söz güçtür	T = 4: A 18,1; sonsuz ufuk 52,6
Algoritmik danışıklılık	Tetik stratejisi δ yeterince büyükse işbirliğini sürdürür	$\delta \geq 1/2$
Veriden fiyat	Deneysel gelir maksimizasyonu; az örnekte aşırı uyum	p = 0,47: deneysel 0,294, gerçek 0,249
Üstel mekanizma	Olasılık $\propto \exp(\varepsilon \cdot \text{gelir}/2\Delta)$; mahremiyet + yaklaşık IC	$\varepsilon = 2$: oran $\leq e^2 \approx 7{,}39$ (örnekte 1,44)
Taban ücret	Dolu blokta +%12,5, boş blokta −%12,5; ücret yakılır	100 → 160,2 → 115
MEV	Sıralama gücü değer taşır; sandviç saldırısı	kullanıcı −8,09 B, saldırgan +9,71 A
Harita	Denge → tasarım → uygulama → öğrenme → gelecek	hazırlıktan gelecek yönelimlere tek kavram haritası

DEĞERLENDİRME

Sık Yapılan Hatalar

Firma sayısını unutmak

İki firmada yeten sabır kalabalık piyasada yetmez: eşik $\delta \geq 1 - 1/n$, $n = 5$ iken 0,80.

Sonlu ve sonsuz ufku karıştırmak

Sonlu ufukta son öneren güçlüdür ($T = 4$'te A 18,1); sonsuz ufukta ilk öneren biraz öndedir (52,6).

Deneysel geliri gerçek sanmak

Aynı veride seçilip ölçülen fiyatın başarısı abartılıdır; ayrı doğrulama verisi gerekir.

Üstel mekanizmada doğrusal orantı

Olasılık gelirle değil $\exp(\varepsilon \cdot R/2\Delta)$ ile orantılıdır; gelir farkı olasılık oranına üstel yansır.

Taban ücreti toplamsal güncellemek

Değişim çarpımsaldır: iki dolu blok 100 → 112,5 → 126,6 (125 değil). Taban ücret üreticiye değil, yakılmaya gider.

Gevşek kayma toleransını güvenli sanmak

İşlemin geçmesini garantiler ama saldırgana daha büyük öne geçme alanı bırakır.

DEĞERLENDİRME

Örnek Problemler

1. 100 TL, iki tur (A sonra B önerir), $\delta = 0{,}9$. Alt oyun mükemmel dengede paylaşım nedir? Sonsuz ufukta A'nın payı?
2. Üç firma tekel kârını (100) eşit paylaşıyor; sapan bir dönem 100 alıyor, sonra 0. Tetik stratejisi hangi δ için sürer?
3. Örnek değerler 0,9; 0,6; 0,5; 0,2. Deneysel geliri maksimize eden fiyat ve gelir nedir?
4. $r = 100$ iken üç dolu blok ve bir boş blok gelirse taban ücret kaç olur?
5. Haritada Çevrim İçi Platformlar düğümündeki GSP–VCG karşılaştırmasında VCG hangi konudan, GSP'nin dayandığı ikinci fiyat fikri hangi konudan gelir?
6. Üstel mekanizmada iki aday fiyatın gelirleri 2 ve 1, $\varepsilon = 2$, $\Delta = 1$. Olasılıklar nedir?

▶ **Cevapları göster**

DEĞERLENDİRME

Etkileşimli Oyunlar

Yapay Zekâ Ajan Turnuvası

Sabit, uyarlanabilir ve öğrenen ajanların Mahkûmun İkilemi, Yazı-Tura Eşleştirme, Koordinasyon Oyunu ve TKM'de yarışı.

Age of Çotanak

Bölge kontrol oyunu; yapay zekâya karşı çekişmeli arama.

Kendin dene: pizza oylaması

Üç arkadaş tek pizzada anlaşmalı: her biri malzeme önerir, sonra oylarlar. Küçük bir konsol uygulaması bile oylama kuralını ve stratejik oyu gösterir.

Kaynaklar

- T. Roughgarden, *Twenty Lectures on Algorithmic Game Theory*, Cambridge University Press, 2016.
- A. Rubinstein, "Perfect Equilibrium in a Bargaining Model", *Econometrica*, 1982.
- E. Calvano, G. Calzolari, V. Denicolò, S. Pastorello, "Artificial Intelligence, Algorithmic Pricing, and Collusion", *American Economic Review*, 2020.
- V. Conitzer, T. Sandholm, "Complexity of Mechanism Design", *UAI*, 2002.
- P. Dütting, Z. Feng, H. Narasimhan, D. Parkes, S. Ravindranath, "Optimal Auctions through Deep Learning", *ICML*, 2019.
- F. McSherry, K. Talwar, "Mechanism Design via Differential Privacy", *FOCS*, 2007.
- R. Cole, T. Roughgarden, "The Sample Complexity of Revenue Maximization", *STOC*, 2014.
- T. Roughgarden, "Transaction Fee Mechanism Design", *ACM EC*, 2021.
- P. Daian ve diğ., "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability", *IEEE S&P*, 2020.
- N. Brown, T. Sandholm, "Superhuman AI for Multiplayer Poker", *Science*, 2019.
- D. Kempe, J. Kleinberg, É. Tardos, "Maximizing the Spread of Influence through a Social Network", *KDD*, 2003.
- H. Chung, E. Shi, *SODA*, 2023; H. Chung, T. Roughgarden, E. Shi, *ACM EC*, 2024 (işlem ücreti mekanizmalarında imkânsızlık).
- D. Lehmann, L. I. O'Callaghan, Y. Shoham, *J. ACM*, 2002; R. Axelrod, *J. Conflict Resolution*, 1980.

DEĞERLENDİRME

Dersten Sonra: Kitaplar, Ders Sayfaları, Araçlar

Kitaplar

- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (ed.), *Algorithmic Game Theory*, Cambridge UP, 2007.
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017.
- D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, Cambridge UP, 2010.
- R. A. Hearn, E. D. Demaine, *Games, Puzzles, and Computation*, A K Peters, 2009.

Ders sayfaları

- T. Roughgarden, CS364A, Stanford, 2013
- N. Shah, CSC304, Toronto, 2019
- M. Dinitz, 601.436/636, Johns Hopkins, 2022
- A. Procaccia, 15-896, CMU, 2016
- M. Hoefer, Algorithmic Game Theory, Frankfurt, 2022–2023

Araçlar

- Gambit: oyun ağacı ve matris çizme, denge hesaplama.
- Nashpy: Python'da iki oyunculu oyunların dengeleri.
- Game Theory 101: video dizisi.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim



Hazırlık: Σ ve argmax, olasılık ve beklenen değer, kazanç matrisini okuma, kayıtsızlık denklemi. Bütün bölümler bu araçları kullanır.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

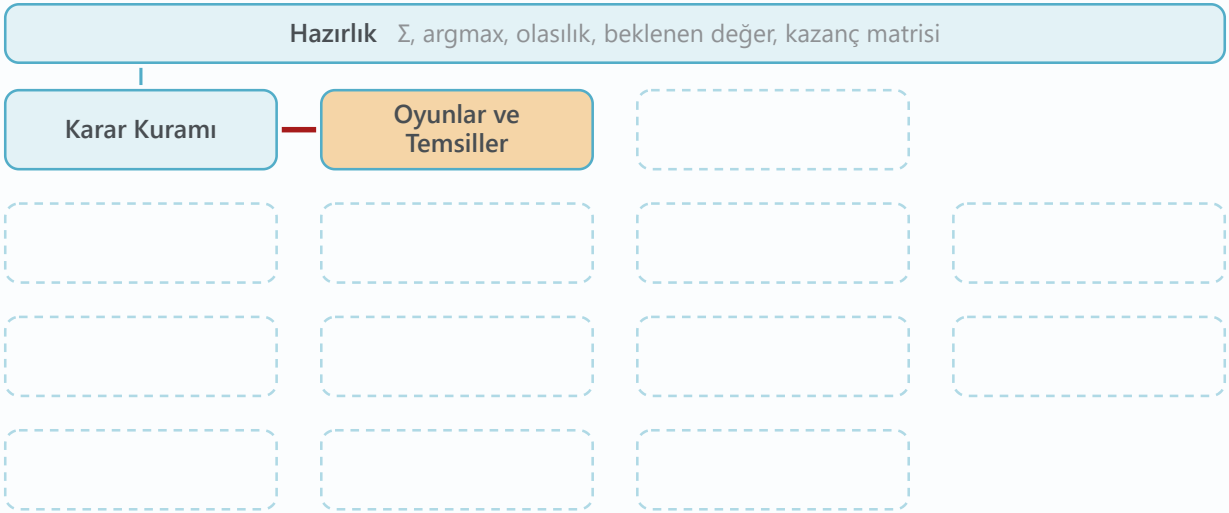


Karar Kuramı ← Hazırlık: tek kişilik karar: beklenen fayda, risk tutumu, Bayesçi karar ve bilginin değeri.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

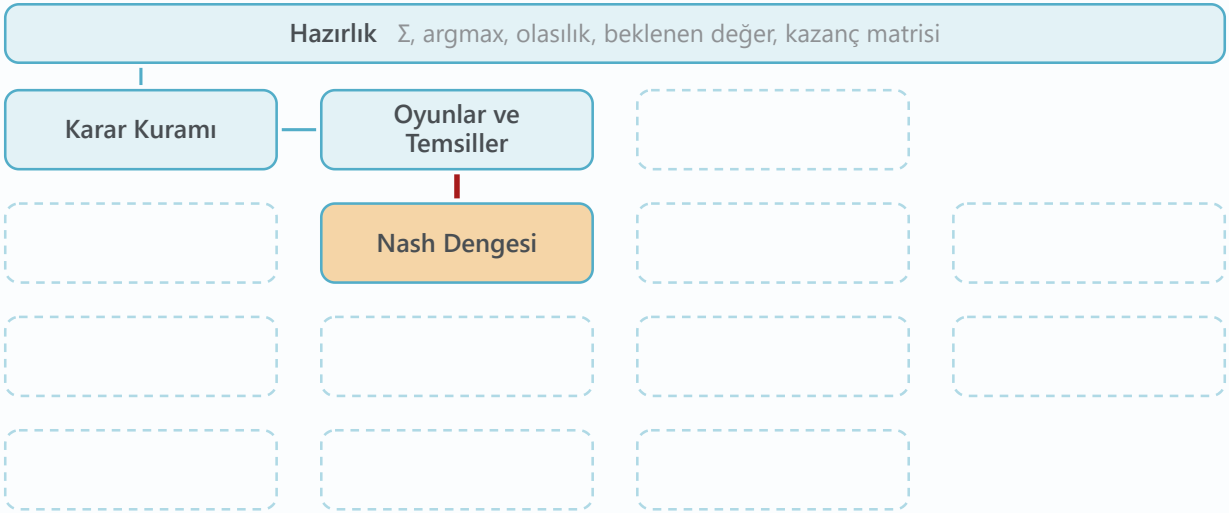


Oyunlar ve Temsiller ← Karar Kuramı: karar verenler çoğalınca oyun doğar: normal ve genişletilmiş form, geriye doğru çıkarım.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

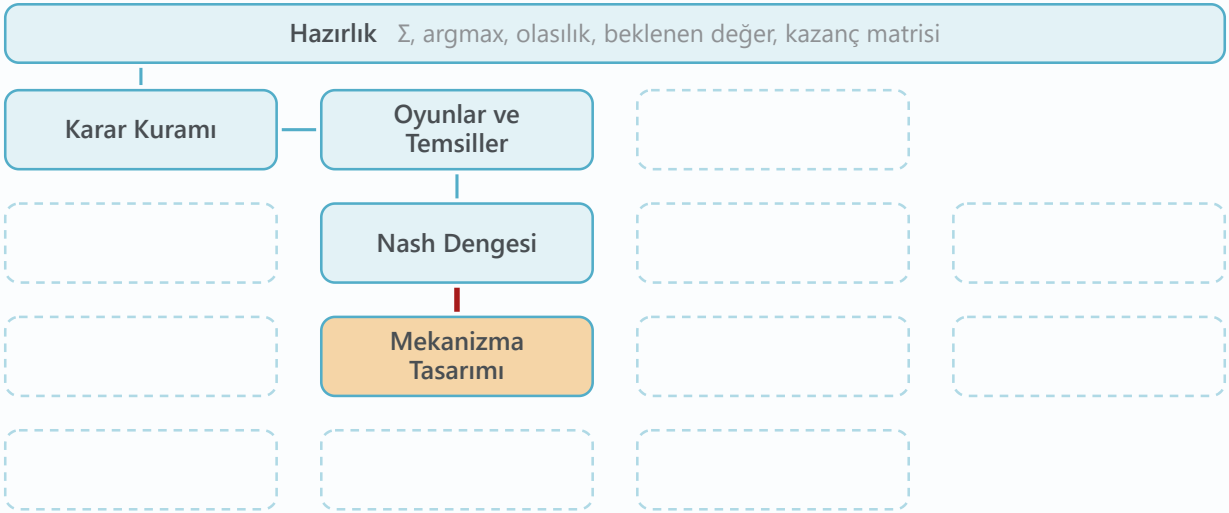


Nash Dengesi ← Oyunlar ve Temsiller: 2×2 oyunlarda işaretlenen dengeler genel Nash tanımına ve varlık teoremine götürür.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

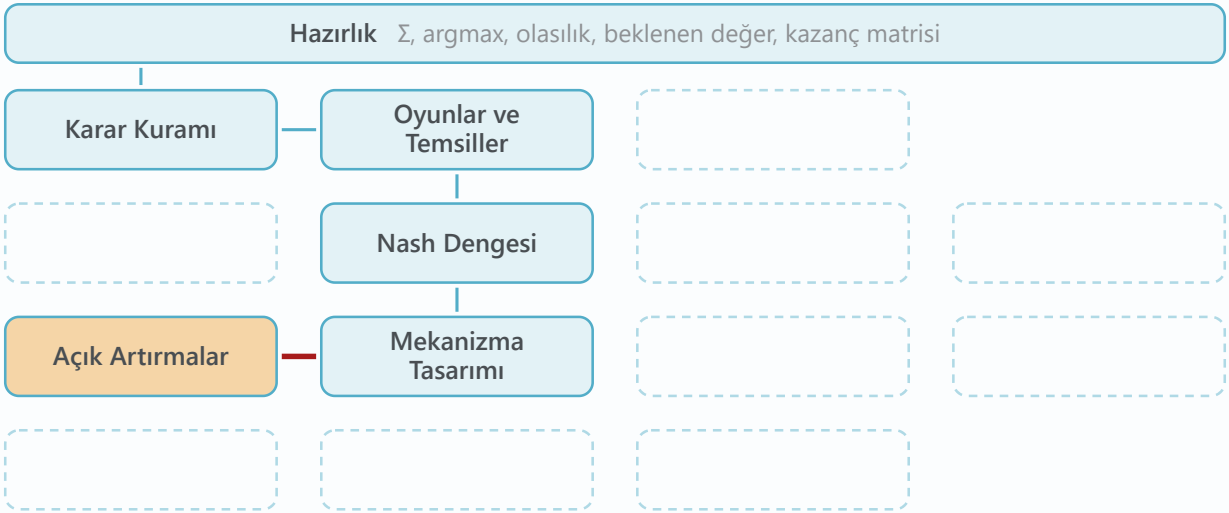


Mekanizma Tasarımı ← Nash Dengesi: denge + teşvik: oyunu çözmek yerine istenen dengeyi veren kuralı tasarlamak.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

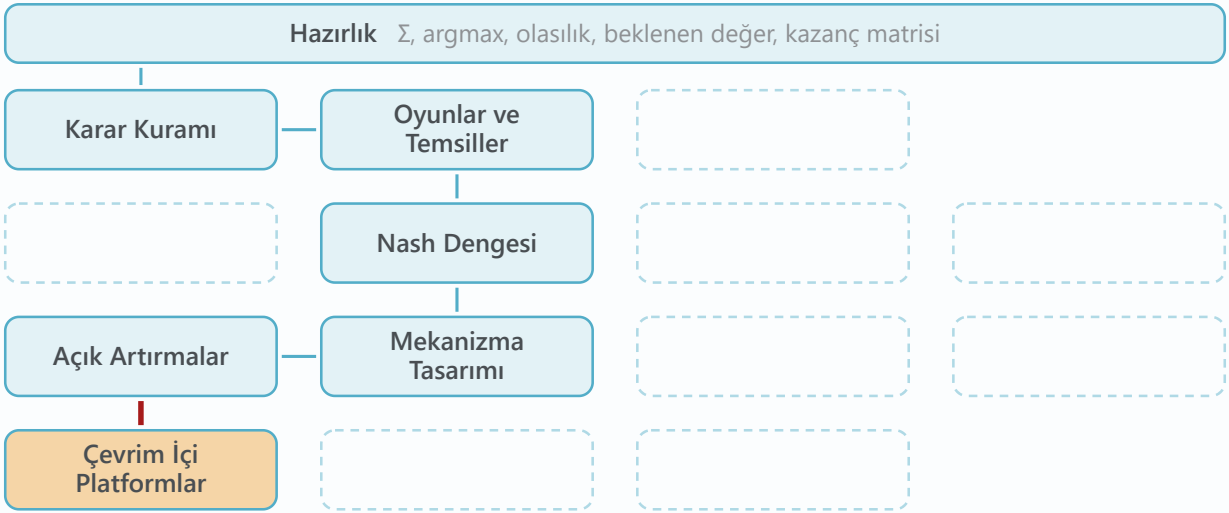


Açık Artırmalar ← Mekanizma Tasarımı: VCG'nin tek nesneli hâli ikinci fiyat (Vickrey) açık artırmasıdır.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

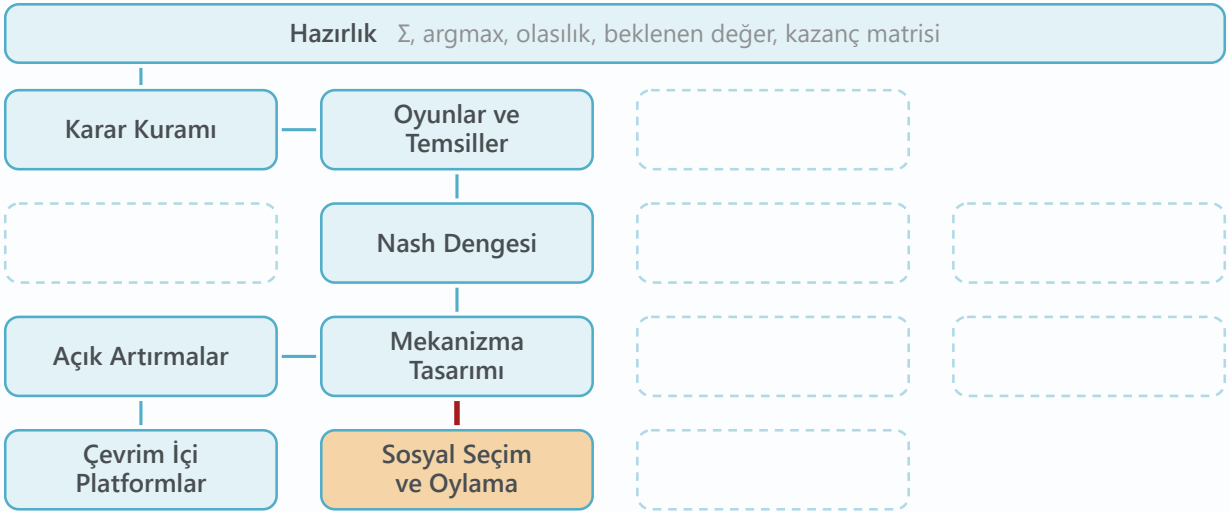


Çevrim İçi Platformlar ← Açık Artırmalar: reklam yuvası açık artırmaları: GSP, Vickrey fikrinin çok yuvalı genellemesidir.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

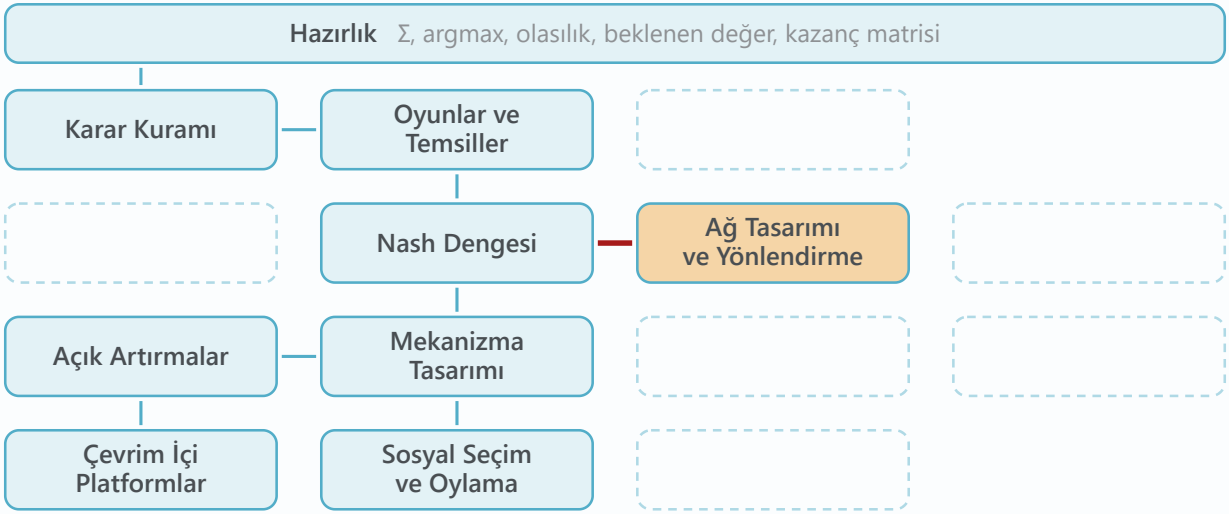


Sosyal Seçim ve Oylama ← Mekanizma Tasarımı: sosyal seçim fonksiyonu → oylama kuralları; Gibbard–Satterthwaite.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

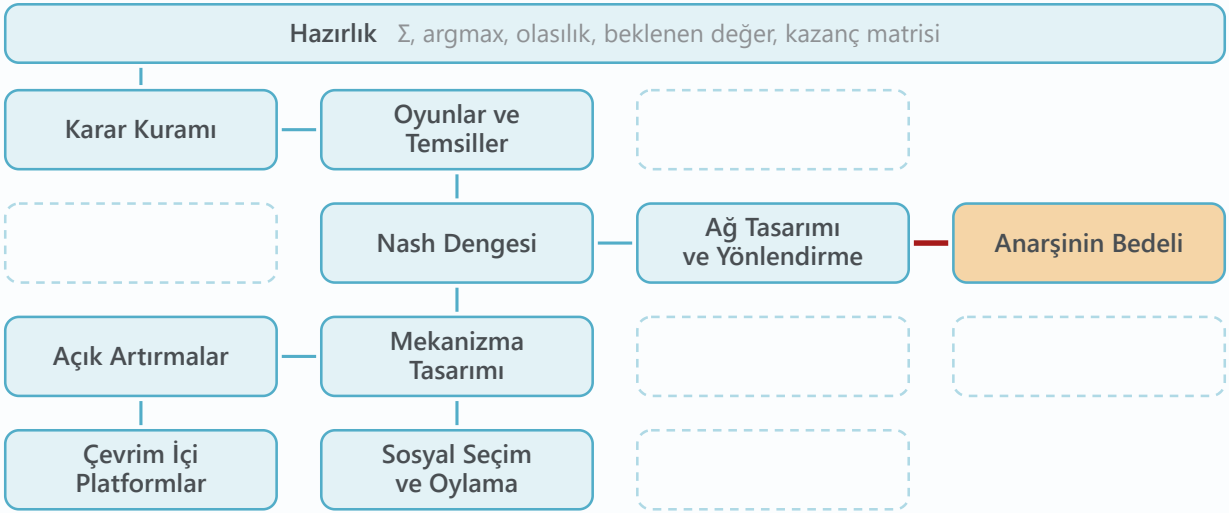


Ağ Tasarımı ve Yönlendirme ← Nash Dengesi: tıkanıklık oyunlarında Nash dengesi = Wardrop dengesi; Rosenthal potansiyeli.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim



Anarşinin Bedeli ← Ağ Tasarımı: Pigou ve Braess ağlarında denge kalitesi → anarşinin bedeli 4/3; tıkanıklık ücreti.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim



Çoklu Ajan Sistemleri ← Oyunlar ve Temsiller: işbirlikçi / işbirlikçi olmayan ayrımı → koalisyon oyunları, Shapley değeri.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim



Algoritmik Adalet ve Etik ← Sosyal Seçim ve Oylama: kolektif karardan adalete: ölçütler ve adil bölüşüm.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim

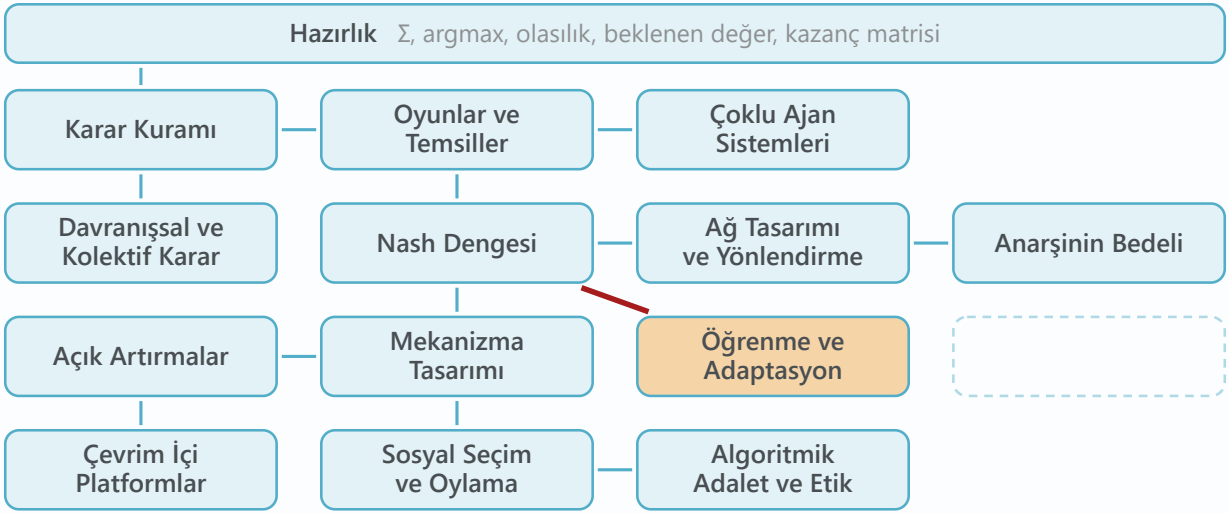


Davranışsal ve Kolektif Karar ← Karar Kuramı: insanlar beklenen faydadan sapar (kayıptan kaçınma); ortak mallarda bireysel ve toplu çıkar ayrışır.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim



Öğrenme ve Adaptasyon ← Nash Dengesi: denge nasıl bulunur? Oyuncular öğrenerek dengeye yakınsayabilir.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Dersin Haritası: Bütün Bölümler, Tek Resim



Gelecek Yönelimler: ölçek (anarşinin bedeli), öğrenen ajanlar (öğrenme) ve etik (adalet) bugünün açık sorularında birleşir.

Bölüm numaraları: Hazırlık 0 · Karar Kuramı 1 · Oyunlar ve Temsiller 2 · Nash Dengesi 3 · Mekanizma Tasarımı 4 · Açık Artırmalar 5 · Çevrim İçi Platformlar 6 · Sosyal Seçim ve Oylama 7 · Ağ Tasarımı ve Yönlendirme 8 · Anarşinin Bedeli 9 · Çoklu Ajan Sistemleri 10 · Algoritmik Adalet ve Etik 11 · Davranışsal ve Kolektif Karar 12 · Öğrenme ve Adaptasyon 13 · Gelecek Yönelimler 14.

EK

Ajan Turnuvası: Tekrarlanabilir Deney Protokolü

Dil modeli ajanlarının hamleleri rastgeledir: aynı eşleşme iki kez farklı skor verebilir. Bir sonuç, başkası aynı deneyi tekrarlayıp aynı sonucu alabiliyorsa güvenilirdir (tekrarlanabilirlik).

Kayda geçenler

- Sabit rastgele tohum (random seed)
- İstemin (prompt) tam metni
- Model sürümü ve sıcaklık (temperature) değeri
- Rakip sırası ve tur sayısı: sabit tutulur, raporda yazılır

Raporlananlar

- Her eşleşme en az N kez tekrarlanır (ör. N = 30)
- Ortalama ve yayılım: standart sapma ya da %95 güven aralığı
- Kod ve oyun kayıtları açık depoda paylaşılır

Örnek: 30 tekrarda ortalama skor 2,6, standart sapma 0,4. Yaklaşık %95 güven aralığı $2{,}6 \pm 1{,}96 \cdot 0{,}4 / \sqrt{30} \approx 2{,}6 \pm 0{,}14$, yani [2,46; 2,74].

EK

Baştan Sona Bir Örnek: Pizza Oylaması

Kişi	1.	2.	3.
Ali	mantar	sucuk	zeytin
Banu	sucuk	mantar	zeytin
Cem	zeytin	sucuk	mantar

- **Model:** oyuncular 3 kişi; strateji = bildirilen sıralama; kazanç = gerçek sıralamaya göre 2, 1, 0.
- **Çoğunluk:** her aday birer birincilik alır, 1–1–1 beraberlik.
- **Borda (2, 1, 0):** mantar 3, sucuk 4, zeytin 2 → sucuk. Sucuk ikili karşılaştırmaların ikisini de 2–1 kazanır: Condorcet kazananı.

Analiz: stratejik oy

Ali sucuğu sona atıp "mantar, zeytin, sucuk" bildirirse Borda puanları 3–3–3 olur. Beraberlik alfabetik çözülürse **mantar** kazanır: Ali yalan söyleyerek en sevdiğini aldırır.

Kendin dene

Konsol sürümü: sıralamaları al; çoğunluk, iki turlu, Borda, Condorcet ve onay oylaması kurallarını yan yana say. Web sürümü: kullanıcı bir oyuncunun bildirimini değiştirsin, kazananın nasıl kaydığını görsün. Beraberlik kuralını değiştirmek de kazananı değiştirir.

Borda'da gömme stratejisi (güçlü rakibi sona atmak), Gibbard–Satterthwaite teoreminin küçük bir yansımasıdır.

Son

Bölüm 14: Gelecek Yönelimler

Çalışma soruları · Terim sözlüğü