Bölüm 4: Mekanizma Tasarımı

Algoritmik Oyun Kuramı

Sosyal seçim fonksiyonu · Doğrudan mekanizma · Açıklama ilkesi · Teşvik uyumluluğu · Myerson lemması · VCG · Gale–Shapley · Medyan mekanizması · TTC · Bayesçi oyunlar

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_04_Mekanizma_Tasarimi.html

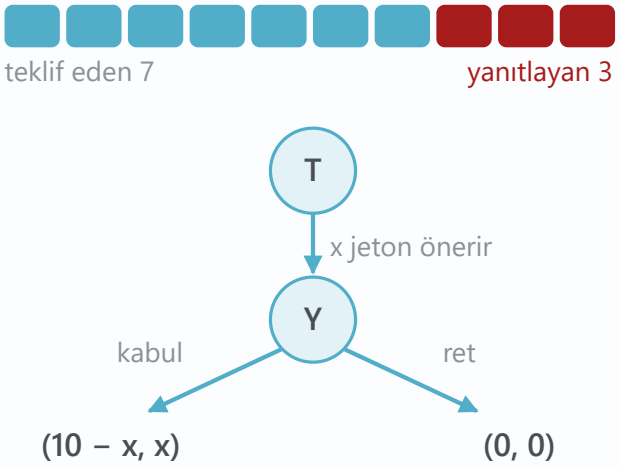
Oyunla Başlangıç: Ültimatom Oyunu

Oyunun kuralı

Teklif eden 10 jetonu bölüşür ve yanıtlayana bir pay önerir; yanıtlayan kabul ederse bölüşüm uygulanır, reddederse ikisi de 0 alır.

Merak sorusu

- Yanıtlayan siz olsaydınız 1 jetonluk teklifi kabul eder miydiniz?
- Teklif eden siz olsaydınız kaç jeton önerirdiniz?



Ültimatom Oyunu →

TEMEL KAVRAMLAR

Öğrenme Çıktıları

Kavramlar

Sosyal seçim fonksiyonu, doğrudan mekanizma, açıklama ilkesi ve Bayes–Nash dengesini açıklamak.

Teşvik analizi

Bir mekanizmanın teşvik uyumlu (IC) ve bireysel rasyonel (IR) olup olmadığını sınamak.

VCG hesabı

Verimli sonucu ve her oyuncunun VCG ödemesini “o olmasaydı” refahıyla hesaplamak.

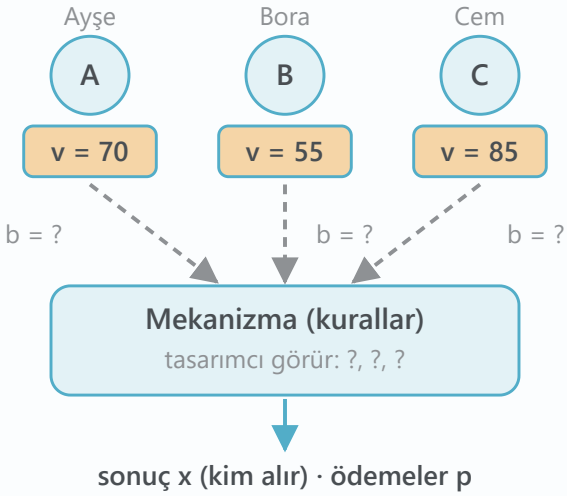
Parasız tasarım

Gale–Shapley ile kararlı eşleştirme, TTC ile takas bulmak; medyan mekanizmasının neden stratejiye dayanıklı olduğunu göstermek.

TEMEL KAVRAMLAR

Mekanizma Tasarımı Nedir?

- Oyun kuramının “ters” yaklaşımı: oyun verilmez, **oyunu biz kurarız**.
- İstenen sonuç için kurallar (kim ne alır, kim ne öder) tasarlanır.
- Zorluk: oyuncuların **özel bilgisi** (değer, tercih, maliyet) tasarımcıdan gizlidir.
- Amaç: bencil ve stratejik oyuncular kendi çıkarını gözetirken bile istenen sonuca ulaşmak.
- Örnekler: vergiler, ihaleler, seçim sistemleri, okul yerleştirme, reklam açık artırmaları.



2007 Nobel Ekonomi Ödülü: Hurwicz, Maskin ve Myerson — “mekanizma tasarımı kuramının temellerini attıkları için”. 2012'de Roth ve Shapley kararlı eşleştirme ve piyasa tasarımı çalışmalarıyla ödüllendirildi.

TEMEL KAVRAMLAR

Oyun Kuramı ve Mekanizma Tasarımı



Oyun kuramı: **Kurallar** → **Sonuç**. Mekanizma tasarımı: **Sonuç** → **Kurallar**. Tasarımcı oyuncuları zorlayamaz; yalnızca kuralları seçer.

TEMEL KAVRAMLAR

Teşvikler (Incentives)

İnsanlar teşviklere göre davranır

Kurallar belli bir davranışı kârlı yaparsa oyuncular o davranışa yönelir.

İyi bir mekanizma

Dürüst davranışı ödüllendirir, stratejik yalanı caydırır: *teşvik uyumluluğu* (incentive compatibility).

Beleşçi (free-rider) sorunu

Kamu malı finansmanında herkes katkısını düşük gösterip başkalarının ödemesinden yararlanmak ister.

Stratejik oy

Seçmen gerçek tercihi yerine kazanma şansı olan adaya oy verebilir.

Teşvik uyumsuzluğu → yanlış bilgi → verimsiz ya da adil olmayan sonuçlar.

TEMEL KAVRAMLAR

Gerçek Hayat Uygulamaları

Vergi sistemleri

Gelir beyanı özel bilgidir; denetim ve oranlar dürüst beyanı teşvik etmelidir.

Kamu malı finansmanı

Park, yol, kütüphane: kim ne kadar ödemeli, proje yapılmalı mı?

İhale ve açık artırmalar

Kamu alımları, frekans lisansları, sanat eserleri.

Çevrim içi reklam

Her arama sorgusunda milisaniyeler içinde çalışan reklam açık artırmaları.

Eşleştirme piyasaları

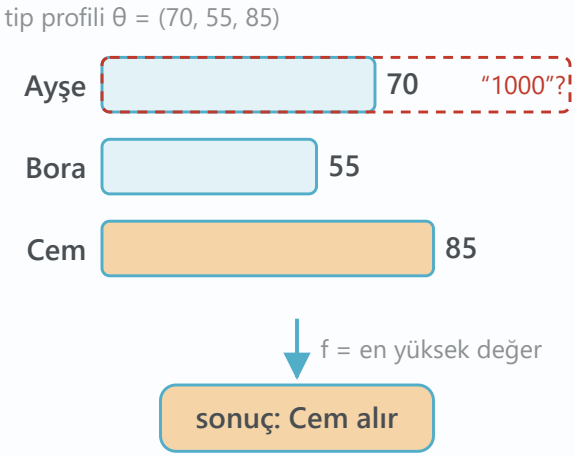
Okul yerleştirme, tıp ihtisası, yurt odası değişimi: para yok, kurallar var.

Grup proje notlama

Öğrenciler katkılarını kendileri bildirirse notlama kuralı abartmayı caydırmalıdır.

Sosyal Seçim Fonksiyonu (Social Choice Function)

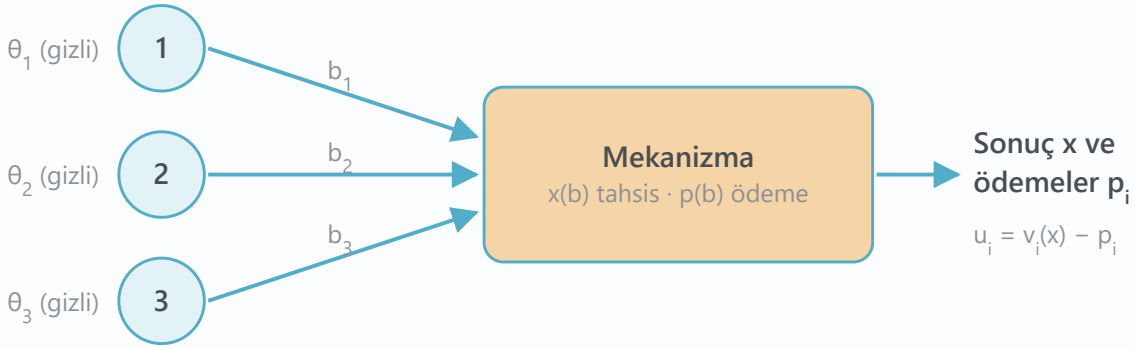
- Her oyuncunun bir **tipi** $\theta_i \in \Theta_i$ vardır: değeri, tercihi, maliyeti. Tip özel bilgidir.
- Olası sonuçlar kümesi O (ör. nesneyi kim alır, hangi proje yapılır).
- **Sosyal seçim fonksiyonu** $f : \Theta_1 \times \ldots \times \Theta_n \to O$, her tip profiline "istenen" sonucu atar.
- Sorun: tasarımcı θ'yı bilmez; f(θ)'yı uygulamak için oyunculardan bilgi alması gerekir.



Verimli seçim: *f(θ)* = en yüksek değerli oyuncu → Cem. Ama herkes "benim değerim 1000" derse?

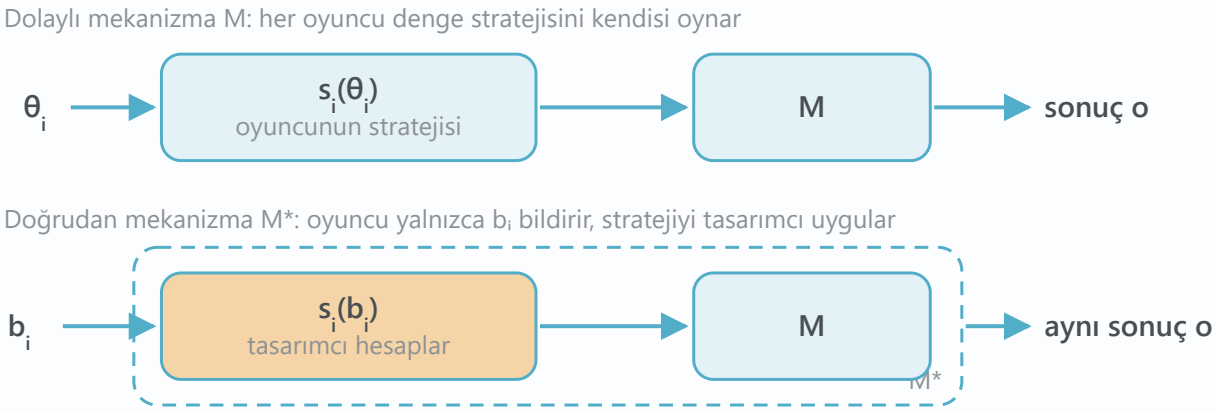
Mekanizma tasarımının sorusu: f'yi, oyuncular **kendi çıkarlarını** gözetirken de uygulayan kurallar var mı?

Doğrudan Mekanizma (Direct Mechanism)



- Doğrudan mekanizmada katılımcılar özel bilgilerini (tiplerini) **bildirir**: b_i (bildirim; yalan olabilir).
- Mekanizma bu bildirimlerden sonucu x(b) ve ödemeleri p(b) hesaplar. Yarı doğrusal fayda: $u_i = v_i(x) - p_i$ (v_i: tipi θ_i olan oyuncunun değerlemesi; tek nesnede $v_i = \theta_i$).
- Dolaylı mekanizmada ise strateji başka bir şeydir: artırmada el kaldırmak, sırayla teklif vermek…

Açıklama İlkesi (Revelation Principle)



- Bir sonuç dolaylı bir mekanizmada denge olarak uygulanabiliyorsa, uygun koşullarda dürüst bildirimin denge olduğu doğrudan bir mekanizma da aynı sonucu uygulayabilir.
- Neden? Yalan söylemek, M'de denge dışı bir strateji oynamakla aynıdır; dengede bu kârlı değildi.
- Örnek: İngiliz açık artırmasında herkes değerine kadar kalır. Tasarımcı “değerini yaz, senin yerine ben kalayım” derse doğrudan bir mekanizma (ikinci fiyat) çıkar ve dürüst yazmak en iyisidir.
- İlke dürüst bildirimin baskın strateji olduğunu her durumda söylemez: kullanılan denge kavramına (baskın strateji / Bayes–Nash) göre teşvik koşulu değişir.

Teşvik Uyumluluğu (IC) ve Bireysel Rasyonellik (IR)

Baskın stratejide teşvik uyumlu (DSIC)

Gerçek tipi θ_i olan oyuncu için doğruyu söylemenin faydası ≥ b_i yalanını söylemenin faydası (diğerleri ne bildirirse bildirsin):

$$u_i(\theta_i;\ \theta_i,\ b_{-i}) \geq u_i(\theta_i;\ b_i,\ b_{-i})$$

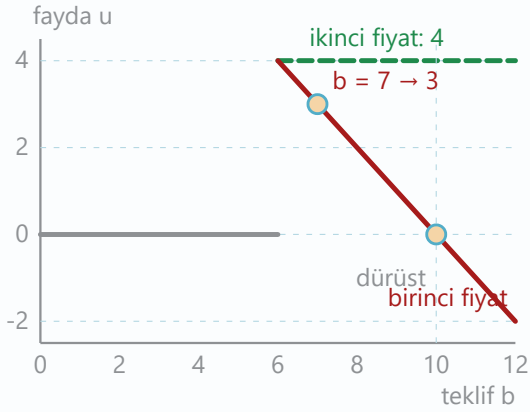
Bayesçi teşvik uyumlu (BIC)

Diğerleri doğruyu söylerken, tiplerin dağılımına göre *beklenen* faydada doğruyu söylemek en iyisidir. DSIC ⇒ BIC.

Bireysel rasyonellik (IR)

Katılım, oyuncuya sistemin dışında kalmaktan daha düşük fayda vermemelidir:

$$u_i(\theta_i;\ \theta_i,\ b_{-i}) \geq 0 \quad \text{(dürüst bildirimde)}$$



Değer 10, rakip teklif 6. Kazananın kendi teklifini ödediği kuralda (birinci fiyat) dürüst teklif 0, 7 teklifi 3 fayda verir: **IC değil**. Kazananın rakip teklifi ödediği kuralda (ikinci fiyat) 6'nın üstündeki her teklif 4 verir.

Oyuna Dönüş: Ültimatom Oyunu

Kuramın cevabı

Yalnız jetonu önemseyen yanıtlayana kabul x, ret 0 verir: x ≥ 0 her teklifi kabul etmek en iyi tepkidir (IR); x = 0'da ret de olabilir. Teklif eden bir "al ya da bırak" mekanizması sunar. Tam sayı tekliflerde iki alt oyun mükemmel denge vardır: **(10, 0)** (0 da kabul edilir) ve **(9, 1)** (kayıtsızken ret).

Gerçekte

İlk deneyden (Güth, Schmittberger ve Schwarze, 1982) beri sonuç benzer: en sık teklif pastanın %40–50'sidir ve %20'nin altındaki tekliflerin yaklaşık yarısı reddedilir (Camerer, 2003). İnsanlar adaletsiz bulduğu teklifi, kendi kazancı pahasına cezalandırır.

Oyundaki yanıtlayan	Kabul ettiği en küçük teklif	Teklif edenin kazancı
Kuram (yalnızca jeton)	1 (0'ı da kabul ederse 0)	9 (o zaman 10)
Seviye 1: hoşgörülü	1	9
Seviye 2: stratejik	3	7
Seviye 3: adalete duyarlı	5	5

Teşvik uyumlu bir mekanizma tasarlarken fayda fonksiyonu doğru seçilmelidir: oyuncu adaleti de önemsiyorsa, IR koşulu "x ≥ 0" değil, "x, adil bulduğu paya ulaşıyor" olur.

Adım Adım: Orantılı Maliyet Paylaşımı IC mi?

	Ayşe	Bora	Cem	Toplam
Gerçek değer v	30	25	20	75
Bildirim	—	—	—	—
Ödeme	—	—	—	—
Fayda u = v − ödeme	—	—	—	—
Park yapılır mı?	—			

Park 60 bin TL. Kural: bildirimlerin toplamı ≥ 60 ise park yapılır, maliyet **bildirimlerle orantılı** paylaşılır.

Çözüm fikri: oyuncunun bildirimi ödemesini **yalnızca seçilen sonucu değiştirerek** etkileyebilmeli → VCG.

Adım Adım: Orantılı Maliyet Paylaşımı IC mi?

	Ayşe	Bora	Cem	Toplam
Gerçek değer v	30	25	20	75
Bildirim	30	25	20	75
Ödeme	24	20	16	60
Fayda u = v − ödeme	**6**	**5**	**4**	15
Park yapılır mı?	Evet (toplam ≥ 60)			

Herkes dürüst: toplam 75 ≥ 60, park yapılır. Ödemeler 60·30/75 = 24, 20, 16; faydalar 6, 5, 4. Herkes ≥ 0 → IR sağlanır.

Çözüm fikri: oyuncunun bildirimi ödemesini **yalnızca seçilen sonucu değiştirerek** etkileyebilmeli → VCG.

Adım Adım: Orantılı Maliyet Paylaşımı IC mi?

	Ayşe	Bora	Cem	Toplam
Gerçek değer v	30	25	20	75
Bildirim	30	25	5	60
Ödeme	30	25	5	60
Fayda u = v − ödeme	**0**	**0**	**15**	15
Park yapılır mı?	Evet (toplam ≥ 60)			

Cem değerini 5 gösterir: toplam yine 60, park yine yapılır; Cem artık 5 öder ve faydası **4** → **15**. Yalan kazandırdı → IC değil.

Çözüm fikri: oyuncunun bildirimi ödemesini **yalnızca seçilen sonucu değiştirerek** etkileyebilmeli → VCG.

Adım Adım: Orantılı Maliyet Paylaşımı IC mi?

	Ayşe	Bora	Cem	Toplam
Gerçek değer v	30	25	20	75
Bildirim	15	12,5	10	37,5
Ödeme	0	0	0	0
Fayda u = v − ödeme	**0**	**0**	**0**	0
Park yapılır mı?	Hayır (toplam < 60)			

Herkes aynı mantıkla yarısını bildirirse toplam 37,5 < 60: park yapılmaz, toplam fayda 15 → 0. Beleşçilik verimsizlik doğurur.

Çözüm fikri: oyuncunun bildirimi ödemesini **yalnızca seçilen sonucu değiştirerek** etkileyebilmeli → VCG.

Mekanizma Tasarımının Ölçütleri

- **Teşvik uyumluluğu (IC)**: gerçek bilgiyi açıklamak en iyi seçenektir.
- **Bireysel rasyonellik (IR)**: katılmak, dışarıda kalmaktan kötü değildir.
- **Verimlilik (efficiency)**: **sosyal refahı** $SW(x) = \Sigma_i v_i(x)$ en yüksek sonuç (dürüst bildirimde).
- **Bütçe dengesi**: ödemelerin toplamı ≥ 0 (zayıf: dış finansman gerekmez) ya da $= 0$ (tam: fazla da kalmaz).

	IC	IR	Verimli	Tam bütçe
VCG kamu projesi	✓	✓	✓	✗
Orantılı pay kamu projesi	✗	✓	✓	✓
Sabit fiyat iki taraflı ticaret	✓	✓	✗	✓
Orta nokta iki taraflı ticaret	✗	✓	✓	✓

Örnek mekanizmalar: her biri bir ölçütten vazgeçiyor.

Bu ölçütler her zaman birlikte sağlanamaz: iki taraflı ticarette verimlilik + (Bayesçi) IC + IR + bütçe dengesi birlikte imkânsızdır (Myerson–Satterthwaite, 1983). Tasarım problemi aralarındaki **ödünleşimi** belirler.

Uygulama Kuramı: Dürüstlük Tek Denge mi?

- Açıklama ilkesi yalnızca dürüst bildirimin *bir* denge olduğunu söyler; aynı mekanizmada istenmeyen başka dengeler de bulunabilir.
- Kurallar iyi bir dengeyi mümkün kılar ama oyuncuların o dengede buluşacağını garanti etmez. Bütün dengeleri istenen sonuca götüren mekanizmaları uygulama kuramı (implementation theory) inceler (ek okuma: Maskin, 1999).

Vickrey'de kötü denge

Değerler Ayşe 10, Bora 5. Ayşe 0, Bora 100 teklif eder: Bora kazanır ve 0 öder (fayda 5). Ayşe kazanmak için 100'ü geçip 100 ödemeli (fayda −90), bu yüzden sapmaz. Bora da sapmakla daha fazla kazanamadığı için bu teklifler bir **Nash dengesi** oluşturur, ama nesne değeri düşük olan Bora'ya gider. Dürüst teklif baskın stratejidir; bu denge zayıf baskılanan tekliflerle kurulur.

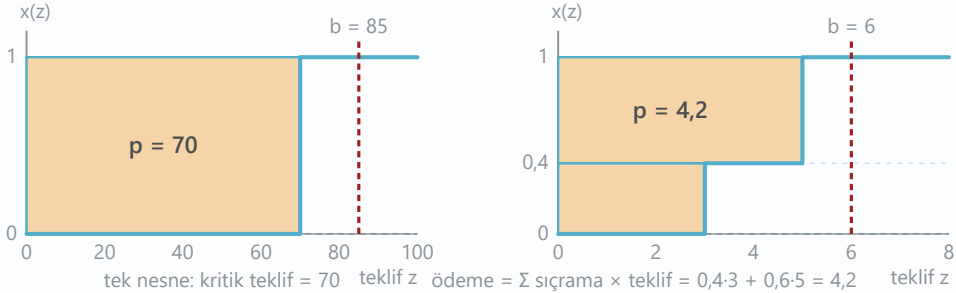
Ders: DSIC'te oyuncu rakibini tahmin etmek zorunda değildir; bu yüzden yalnızca Nash dengesine dayanan tasarımlardan daha sağlamdır. Yine de tek denge garantisi vermez.

Tek Parametreli Ortamlar: Myerson Lemması

- Her oyuncunun tek bir özel sayısı vardır: birim değer v_i. Mekanizma tahsis $x_i(b)$ ve ödeme $p_i(b)$ seçer; fayda $v_i \cdot x_i - p_i$.
- **Myerson lemması** (1981): x, DSIC bir mekanizmada kullanılabilir ⇔ x **monoton** (teklif artınca x_i azalmaz). Ödeme de tek biçimde belirlenir ($p_i(0) = 0$ ile):

$$p_i(b) = b_i \cdot x_i(b) - \int_0^{b_i} x_i(z, b_{-i})\, dz$$

= eğrinin **solunda**, 0 ile x(b) arasında kalan boyalı alan (b × x(b) dikdörtgeninden eğrinin altındaki alan çıkınca kalan kısım)



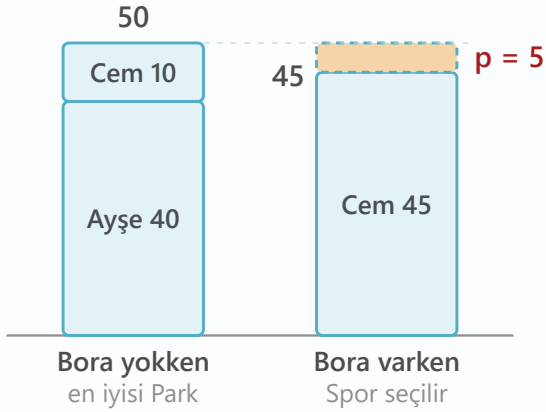
0/1 tahsiste ödeme = **kritik teklif** (kazanmak için gereken en düşük teklif): Vickrey'nin ikinci fiyatı tam olarak budur. Ödeme kritik teklif değilse IC bozulur: eşik 50, sabit ödeme 20 ise değeri 20 ile 50 arasındaki oyuncu 50'yi aşan bir teklifle abartıp kazanır.

Vickrey–Clarke–Groves (VCG) Mekanizması

- Bildirilen değerlere göre sosyal refahı en yüksek yapan sonucu seçer: $x^* = \text{argmax}_x \Sigma_i b_i(x)$ ($b_i(x)$: i'nin x için bildirdiği değer).
- Her oyuncunun ödemesi, katılımının diğer oyuncular üzerinde oluşturduğu **dışsallığa** (externality) dayanır (Clarke pivot kuralı):

$$p_i = \max_x \Sigma_{j \neq i} b_j(x) - \Sigma_{j \neq i} b_j(x^*)$$

= (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı)



Kamu projesi örneğinde Bora'nın dışsallığı.

- Bildirim ödemeyi yalnızca x^* üzerinden etkiler; bu yüzden uygun koşullarda dürüst bildirim **baskın stratejidir**.
- Oyuncunun faydası = sosyal refah − (kendisinden bağımsız bir terim): herkes sosyal refahı maksimize etmek ister.

VCG MEKANİZMASI

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	—	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	—	—	—		
(b) Σ i olmadan	—	—	—		

Üç sakin, üç proje. Her sakin her projeye değerini (bin TL) bildiriyor. Yalnızca **bir** proje yapılacak.

Adım 1 / 12

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

VCG MEKANİZMASI

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	—	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ i olmadan	—	—	—		

(a) Herkesle en iyi seçim: her projede üç değerin toplamı (sosyal refah) Park 50, Kütüphane 55, Spor 65. En yüksek toplam Spor salonunda: **x* = Spor salonu, refah 65**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	—	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Ayşe olmadan	10	50	65		

(b) Ayşe olmasaydı: yalnız Bora ve Cem değerlerinin toplamı Park 10, Kütüphane 50, Spor 65. En iyisi Spor; diğerlerinin refahı **65**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

VCG MEKANİZMASI

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	—	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Ayşe olmadan	10	50	65		

(c) Ayşe varken seçilen proje Spor salonu. Spor salonunda diğerlerinin refahı: Bora 20 + Cem 45 = **65**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Ayşe olmadan	10	50	65		

(d) Ödeme = (b) − (c) = 65 − 65 = 0. Ayşe olsa da olmasa da Spor seçiliyor; Ayşe başkalarına zarar vermiyor.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Bora olmadan	50	25	45		

(b) Bora olmasaydı: yalnız Ayşe ve Cem değerlerinin toplamı Park 50, Kütüphane 25, Spor 45. En iyisi Park; diğerlerinin refahı **50**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

VCG MEKANİZMASI

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	—	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Bora olmadan	50	25	45		

(c) Bora varken seçilen proje Spor salonu. Spor salonunda diğerlerinin refahı: Ayşe 0 + Cem 45 = **45**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	**5**	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Bora olmadan	50	25	45		

(d) Ödeme = (b) − (c) = 50 − 45 = 5. Bora olmasaydı Park seçilirdi ve diğerleri 50 alırdı; Bora yüzünden Spor seçiliyor ve diğerleri 45 alıyor. Bora'nın başkalarına verdiği zarar 5.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	**5**	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Cem olmadan	40	35	20		

(b) Cem olmasaydı: yalnız Ayşe ve Bora değerlerinin toplamı Park 40, Kütüphane 35, Spor 20. En iyisi Park; diğerlerinin refahı **40**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	**5**	—
Cem	10	20	45	—	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Cem olmadan	40	35	20		

(c) Cem varken seçilen proje Spor salonu. Spor salonunda diğerlerinin refahı: Ayşe 0 + Bora 20 = **20**.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	—
Bora	0	30	20	**5**	—
Cem	10	20	45	**20**	—
(a) Σ herkes	50	55	65 ✓		
(b) Σ Cem olmadan	40	35	20		

(d) Ödeme = (b) − (c) = 40 − 20 = 20. Cem olmasaydı Park seçilirdi ve diğerleri 40 alırdı; Cem yüzünden Spor seçiliyor ve diğerleri 20 alıyor. Cem'in başkalarına verdiği zarar 20.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Adım Adım: VCG ile Kamu Projesi Seçimi

	Park	Kütüphane	Spor salonu	Ödeme p_i	Fayda u_i
Ayşe	40	5	0	**0**	**0**
Bora	0	30	20	**5**	**15**
Cem	10	20	45	**20**	**25**
(a) Σ herkes	50	55	65 ✓	25	40
(b) Σ i olmadan	—	—	—		

Faydalar (değer − ödeme): Ayşe 0 − 0 = 0, Bora 20 − 5 = 15, Cem 45 − 20 = 25. Herkesin faydası ≥ 0 (IR). Toplanan 25 kimseye geri dağıtılmaz: bütçe fazlası.

Ödeme p_i = (b) − (c) = (i olmasaydı diğerlerinin en iyi refahı) − (i varken diğerlerinin refahı): **i'nin başkalarına verdiği zarar**.

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	—	—	—
w(a, D)	—	—	—

Oyuncu listesi çıkarılır. `v[i][a]` : i'nin a projesine bildirdiği değer.

Adım 1 / 16

```
1  def vcg(v, A):
▶    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje`

`N = [Ayşe, Bora, Cem]`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D)	—	—	—

Her proje için `w(a, N)` hesaplanır; en büyüğü Spor (65) → **x = S**.

Adım 2 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
▶    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje`

`N = [Ayşe, Bora, Cem]` `x = S`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D)	—	—	—

Ödeme sözlüğü boş başlar; şimdi her oyuncu için “o olmasaydı” sorusu sorulacak.

Adım 3 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
▶    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {}`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Ayşe	10	50	65

i = Ayşe: D, Ayşe dışındaki oyunculardır.

Adım 4 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
▶    for i in N:
▶      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {}`

`i = Ayşe` `D = [Bora, Cem]`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Ayşe	10	50	65

Ayşe yokken en iyi proje Spor salonu: **yok = 65**.

Adım 5 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
▶      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {}`

`i = Ayşe` `D = [Bora, Cem]` `yok = 65`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Ayşe	10	50	65

Ayşe varken seçilen Spor'da D'nin refahı: **var = 65**.

Adım 6 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
▶      var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {}`

`i = Ayşe` `D = [Bora, Cem]` `yok = 65`

`var = 65`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Ayşe	10	50	65

p[Ayşe] = 65 − 65 = 0.

Adım 7 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
▶      p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S`

`p = {Ayşe: 0}` `i = Ayşe`

`D = [Bora, Cem]` `yok = 65` `var = 65`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Bora	50	25	45

i = Bora: D, Bora dışındaki oyunculardır.

Adım 8 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
▶    for i in N:
▶      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {Ayşe: 0}` `i = Bora` `D = [Ayşe, Cem]`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Bora	50	25	45

Bora yokken en iyi proje Park: **yok = 50**.

Adım 9 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
▶      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {Ayşe: 0}` `i = Bora` `D = [Ayşe, Cem]` `yok = 50`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Bora	50	25	45

Bora varken seçilen Spor'da D'nin refahı: **var = 45**.

Adım 10 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
▶      var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {Ayşe: 0}` `i = Bora` `D = [Ayşe, Cem]` `yok = 50` `var = 45`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Bora	50	25	45

p[Bora] = 50 − 45 = 5.

Adım 11 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
▶      p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S`

`p = {Ayşe: 0, Bora: 5}` `i = Bora`

`D = [Ayşe, Cem]` `yok = 50` `var = 45`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Cem	40	35	20

i = Cem: D, Cem dışındaki oyunculardır.

Adım 12 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
▶    for i in N:
▶      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S`

`p = {Ayşe: 0, Bora: 5}` `i = Cem`

`D = [Ayşe, Bora]`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Cem	40	35	20

Cem yokken en iyi proje Park: **yok = 40**.

Adım 13 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
▶      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S`

`p = {Ayşe: 0, Bora: 5}` `i = Cem`

`D = [Ayşe, Bora]` `yok = 40`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Cem	40	35	20

Cem varken seçilen Spor'da D'nin refahı: **var = 20**.

Adım 14 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
▶      var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S` `p = {Ayşe: 0, Bora: 5}` `i = Cem` `D = [Ayşe, Bora]` `yok = 40` `var = 20`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D) · D = N − Cem	40	35	20

p[Cem] = 40 − 20 = 20.

Adım 15 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
▶      p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S`

`p = {Ayşe: 0, Bora: 5, Cem: 20}`

`i = Cem` `D = [Ayşe, Bora]` `yok = 40`

`var = 20`

Kod İzleme: `vcg(v, A)`

	Park	Kütüphane	Spor salonu
w(a, N)	50	55	65
w(a, D)	—	—	—

Sonuç: Spor salonu yapılır; ödemeler {Ayşe: 0, Bora: 5, Cem: 20}. Toplam 25.

Adım 16 / 16

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
▶    return x, p
```

DEĞİŞKENLER

`v = 3 oyuncu × 3 proje` `x = S`

`p = {Ayşe: 0, Bora: 5, Cem: 20}`

VCG MEKANİZMASI

VCG'de Yalan Söylemek Kazandırır mı?

Durum	Bildirim (P, K, S)	Seçilen	Ödeme	Gerçek fayda
Cem dürüst	10, 20, 45	Spor salonu	20	**45 − 20 = 25**

Referans: Cem dürüst bildirir → Spor seçilir, 20 öder, gerçek faydası 45 − 20 = **25**.

Adım 1 / 4

Tüm bildirimler (her proje için 0, 10, …, 100) denendiğinde hiçbir oyuncu dürüst bildirimden daha yüksek fayda elde edemiyor.

VCG MEKANİZMASI

VCG'de Yalan Söylemek Kazandırır mı?

Durum	Bildirim (P, K, S)	Seçilen	Ödeme	Gerçek fayda
Cem dürüst	10, 20, 45	Spor salonu	20	**45 − 20 = 25**
Cem abartır	10, 20, 100	Spor salonu	20	**45 − 20 = 25**

Cem Spor'u 100 gösterir: sonuç değişmez, ödeme de değişmez (ödeme diğerlerinin bildirimlerine bağlı) → yine 25.

Tüm bildirimler (her proje için 0, 10, ..., 100) denendiğinde hiçbir oyuncu dürüst bildirimden daha yüksek fayda elde edemiyor.

VCG'de Yalan Söylemek Kazandırır mı?

Durum	Bildirim (P, K, S)	Seçilen	Ödeme	Gerçek fayda
Cem dürüst	10, 20, 45	Spor salonu	20	**45 − 20 = 25**
Cem abartır	10, 20, 100	Spor salonu	20	**45 − 20 = 25**
Cem küçültür	10, 20, 25	Kütüphane	5	**20 − 5 = 15**

Cem Spor'u 25 gösterir: toplamlar 50, 55, 45 → Kütüphane seçilir. Cem 5 öder ama gerçek değeri 20 → fayda **15 < 25**.

Tüm bildirimler (her proje için 0, 10, …, 100) denendiğinde hiçbir oyuncu dürüst bildirimden daha yüksek fayda elde edemiyor.

VCG MEKANİZMASI

VCG'de Yalan Söylemek Kazandırır mı?

Durum	Bildirim (P, K, S)	Seçilen	Ödeme	Gerçek fayda
Cem dürüst	10, 20, 45	Spor salonu	20	**45 − 20 = 25**
Cem abartır	10, 20, 100	Spor salonu	20	**45 − 20 = 25**
Cem küçültür	10, 20, 25	Kütüphane	5	**20 − 5 = 15**
Bora abartır	0, 60, 20	Kütüphane	25	**30 − 25 = 5**

Bora Kütüphane'yi 60 gösterir: Kütüphane seçilir ama Bora 25 öder → 30 − 25 = **5 < 15**. Hiçbir yalan kazandırmıyor.

Tüm bildirimler (her proje için 0, 10, ..., 100) denendiğinde hiçbir oyuncu dürüst bildirimden daha yüksek fayda elde edemiyor.

VCG MEKANİZMASI

VCG'nin Özellikleri ve Sınırları

Güçlü yönler

+ DSIC: dürüst bildirim baskın strateji.
+ Verimli: sosyal refahı en yüksek sonuç seçilir.
+ Clarke pivotu ve negatif olmayan değerlerde IR; ödemeler ≥ 0.
+ Tek nesnede VCG = Vickrey (ikinci fiyat) açık artırması.

Sınırlar

− Tam bütçe dengesi yok: örnekte 25 birim fazla kalır (fazla verir, açık vermez; zayıf bütçe dengesi sağlanır).
− argmax hesabı kombinatoryal problemlerde NP-zor olabilir.
− Danışıklılığa (collusion) ve sahte kimliklere açık; gelir düşük kalabilir.
− Oyuncuların tüm değerlerini açıklaması mahremiyet sorunu doğurur.

Yüksek hesaplama maliyeti ve bütçe dengesi sorunları gerçek uygulamalarda sınırlayıcı olabilir. Sponsorlu arama reklamlarında VCG yerine çoğunlukla GSP kullanılır.

VCG MEKANİZMASI

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	–	–	–
w(a, D)	–	–	–

Tek nesne: sonuç `a` = nesneyi kim alır. `v[i][a]` = i'nin değeri (a = i ise), yoksa 0.

Adım 1 / 13

```
1  def vcg(v, A):
▶    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne`

`N = [Ayşe, Bora, Cem]`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	–	–	–

w(a, N) yalnızca alıcının değeridir: 70, 55, 85 → **x = Cem** (en yüksek değer).

Adım 2 / 13

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
▶    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne`

`N = [Ayşe, Bora, Cem]` `x = Cem`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	–	–	–

Ödemeler için her teklifçiye “sen olmasaydın?” diye sorulur.

Adım 3 / 13

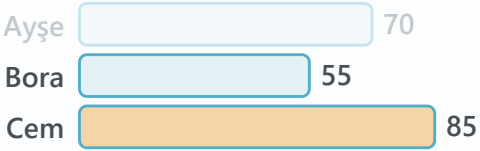
```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
▶    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {}`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	0	55	85

i = **Ayşe**: Ayşe'nin satırı silikleşir; D = diğerleri.

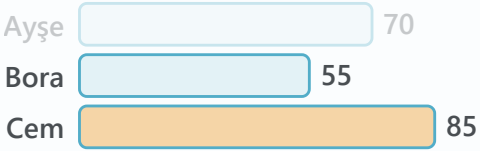

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
▶    for i in N:
▶      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {}` `i = Ayşe` `D = [Bora, Cem]`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	0	55	85

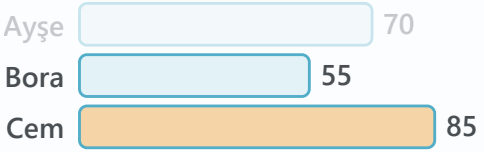
Ayşe yokken nesne Cem'e giderdi: **yok = 85**.


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
▶      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem` `p = {}` `i = Ayşe` `D = [Bora, Cem]` `yok = 85`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	0	55	85

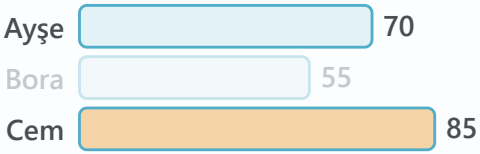
Nesne Cem'de; D'nin oradaki refahı var = 85. p[Ayşe] = 85 − 85 = **0** → kaybeden ödemez.


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
▶      var = w(x, D)                  # i varken
▶      p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem` `p = {Ayşe: 0}` `i = Ayşe` `D = [Bora, Cem]` `yok = 85` `var = 85`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	70	0	85

i = **Bora**: Bora'nın satırı silikleşir; D = diğerleri.

Adım 7 / 13

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
▶    for i in N:
▶      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

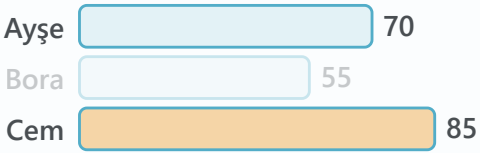
DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {Ayşe: 0}` `i = Bora`

`D = [Ayşe, Cem]`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	70	0	85

Bora yokken nesne Cem'e giderdi: **yok = 85**.


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
▶      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

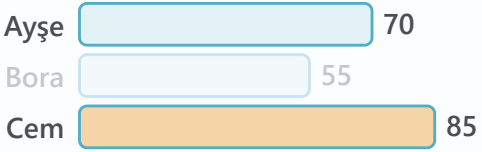
DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {Ayşe: 0}` `i = Bora`

`D = [Ayşe, Cem]` `yok = 85`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	70	0	85

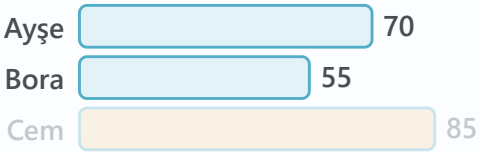
Nesne Cem'de; D'nin oradaki refahı var = 85. p[Bora] = 85 − 85 = **0** → kaybeden ödemez.


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
▶      var = w(x, D)                  # i varken
▶      p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`
`p = {Ayşe: 0, Bora: 0}` `i = Bora`
`D = [Ayşe, Cem]` `yok = 85` `var = 85`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	70	55	0

i = **Cem**: Cem'in satırı silikleşir; D = diğerleri.


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
▶    for i in N:
▶      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

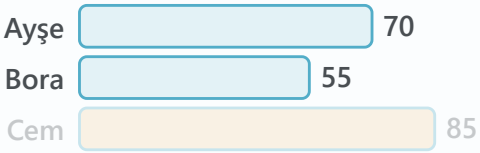
DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {Ayşe: 0, Bora: 0}` `i = Cem`

`D = [Ayşe, Bora]`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	70	55	0

Cem yokken nesne Ayşe'ye giderdi: **yok = 70**.


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
▶      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
12   return x, p
```

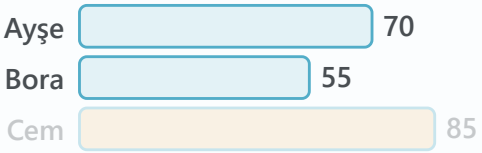
DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {Ayşe: 0, Bora: 0}` `i = Cem`

`D = [Ayşe, Bora]` `yok = 70`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	70	55	0

Nesne Cem'de; D'nin oradaki refahı var = 0. p[Cem] = 70 − 0 = **70** = ikinci en yüksek değer!


```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
▶      var = w(x, D)                  # i varken
▶      p[i] = yok - var
12   return x, p
```

DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {Ayşe: 0, Bora: 0, Cem: 70}`

`i = Cem` `D = [Ayşe, Bora]` `yok = 70`

`var = 0`

Kod İzleme: Tek Nesnede `vcg()` = Vickrey



	a = Ayşe	a = Bora	a = Cem
w(a, N)	70	55	85
w(a, D)	–	–	–

Sonuç: Cem kazanır ve **70** öder; kaybedenler 0. Tek nesnede VCG = Vickrey (ikinci fiyat).

Adım 13 / 13

```
1  def vcg(v, A):
2    N = list(v)              # oyuncular
3    def w(a, S):             # S'nin a'daki refahı
4      return sum(v[i][a] for i in S)
5    x = max(A, key=lambda a: w(a, N))
6    p = {}
7    for i in N:
8      D = [j for j in N if j != i]
9      yok = max(w(a, D) for a in A)  # i yokken
10     var = w(x, D)                  # i varken
11     p[i] = yok - var
▶    return x, p
```

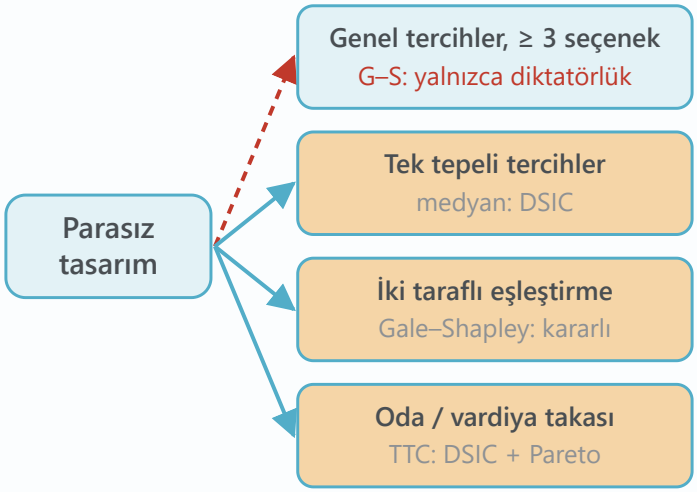
DEĞİŞKENLER

`v = 3 teklifçi, 1 nesne` `x = Cem`

`p = {Ayşe: 0, Bora: 0, Cem: 70}`

Para Olmadan Mekanizma Tasarımı

- Okul yerleştirme, yurt odası değişimi, oy verme: ödeme yapmak ya yasak ya da uygunsuzdur.
- Para yoksa ödeme ile teşvik düzeltilemez; yalnızca **sonuç kuralı** tasarlanabilir.
- Genel tercihlerde kötü haber: en az üç seçenekli, her sonucu mümkün kılan, stratejiye dayanıklı (yalan söylemenin hiç kazandırmadığı) her kural diktatörlüktür: sonucu hep aynı tek kişinin tercihi belirler (Gibbard–Satterthwaite).
- **Çıkış yolları**: tercihlerin yapısını kullan (tek tepeli → medyan) ya da hedefi değiştir ("en iyi" yerine **kararlı** → Gale–Shapley; takas → TTC).

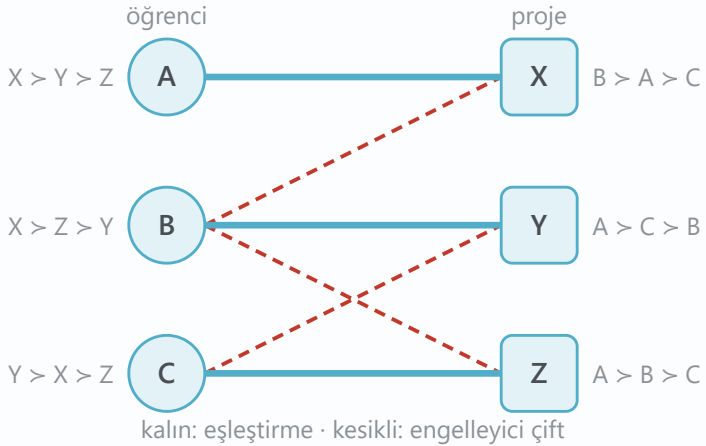


Kararlı Eşleştirme Problemi

- İki taraf: öğrenciler ve bitirme projeleri (her proje tek öğrenci alır). Her iki taraf da karşı tarafı sıralar.
- **Engelleyici çift** (blocking pair): birbirlerini şu anki eşlerine tercih eden öğrenci–proje çifti.
- Engelleyici çifti olmayan eşleştirme **kararlıdır** (stable): kimse anlaşmayı bozup kaçmak istemez.

Öğrenci	1.	2.	3.
A	X	Y	Z
B	X	Z	Y
C	Y	X	Z

Proje	1.	2.	3.
X	B	A	C
Y	A	C	B
Z	A	B	C



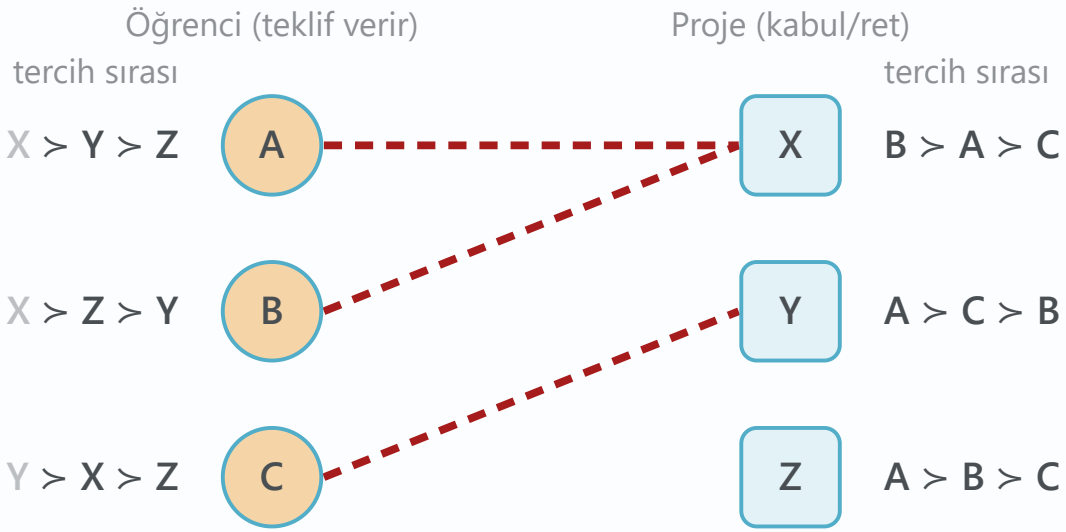
Kararsız örnek {A–X, B–Y, C–Z}: B, X'i Y'ye; X de B'yi A'ya tercih eder → **(B, X) engelleyici**. Aynı mantıkla (B, Z) ve (C, Y) de engelleyici (9 çiftin hepsi denetlendiğinde başka engelleyici çift çıkmaz).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



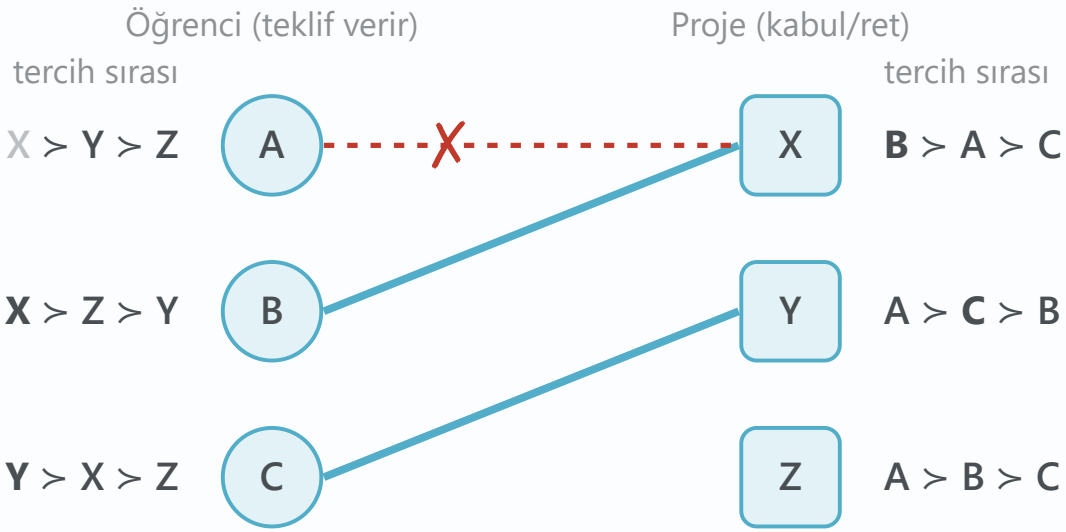
Başlangıç: herkes serbest. Öğrenciler tercih sırasına göre teklif verir; proje en iyi adayı **geçici** tutar, diğerlerini reddeder.

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



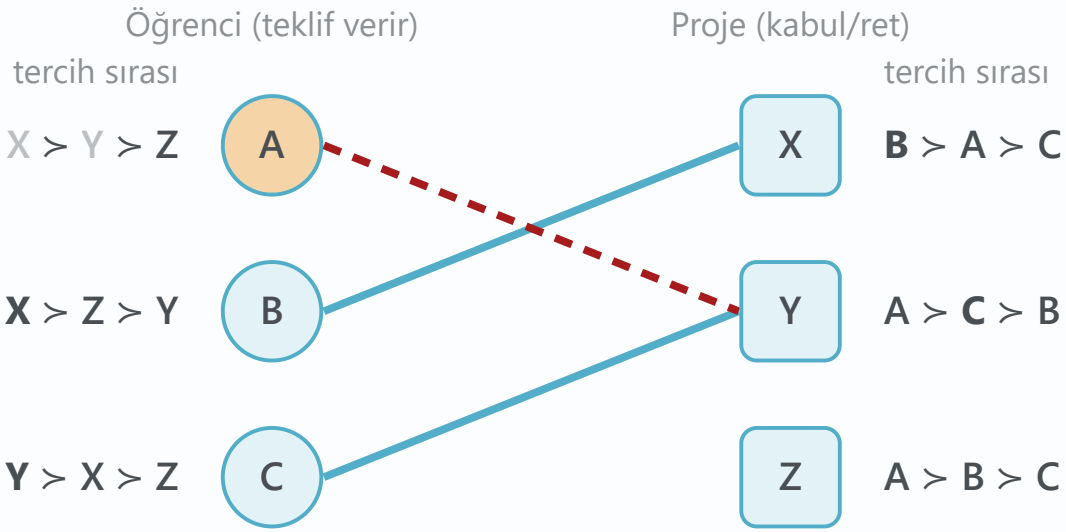
Tur 1 teklifleri: A→X, B→X, C→Y (serbest her öğrenci, henüz reddetmemiş en iyi projesine).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



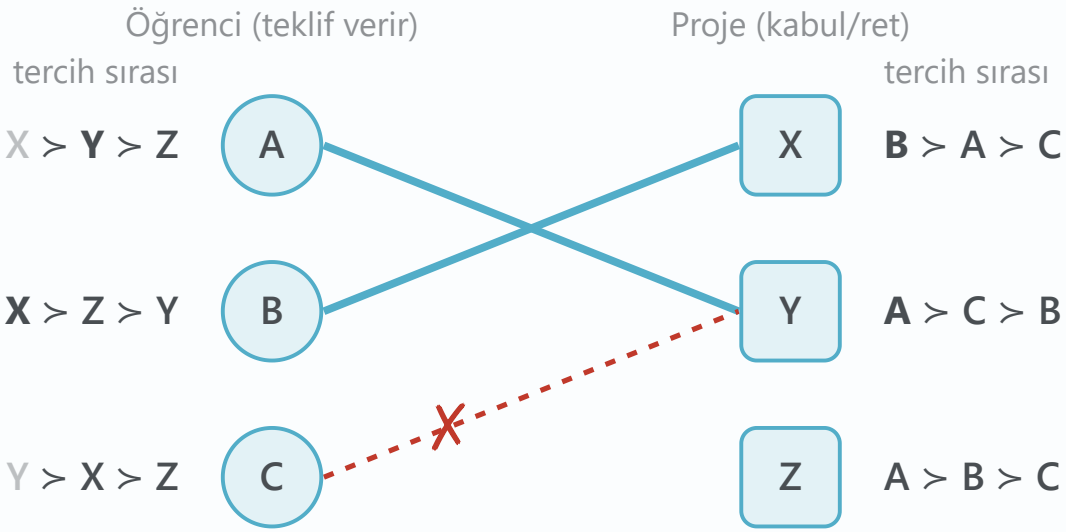
Tur 1 kararı — X: B tutulur, A reddedilir (1. sıradaki B daha iyi); Y: C kabul (geçici).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



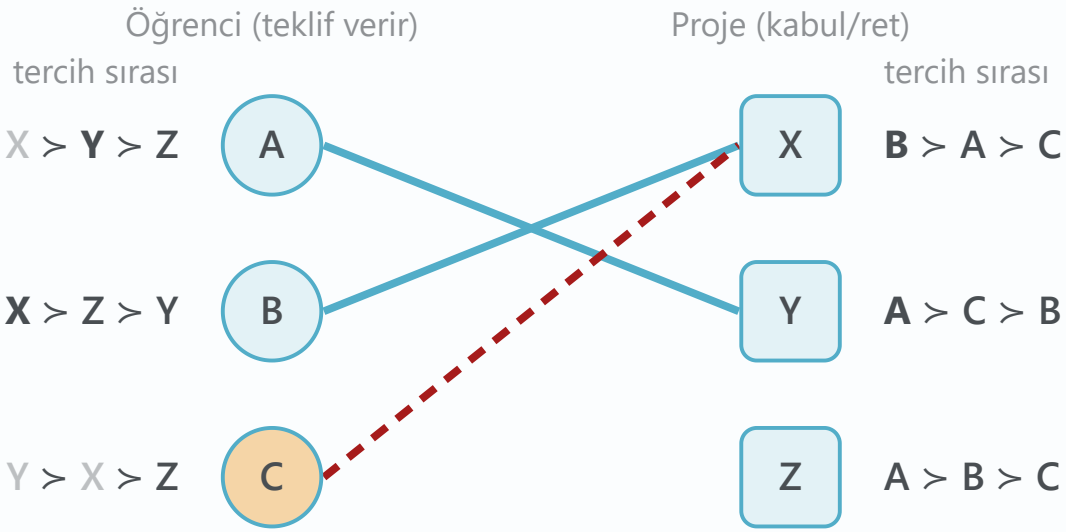
Tur 2 teklifleri: A→Y (serbest her öğrenci, henüz reddetmemiş en iyi projesine).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



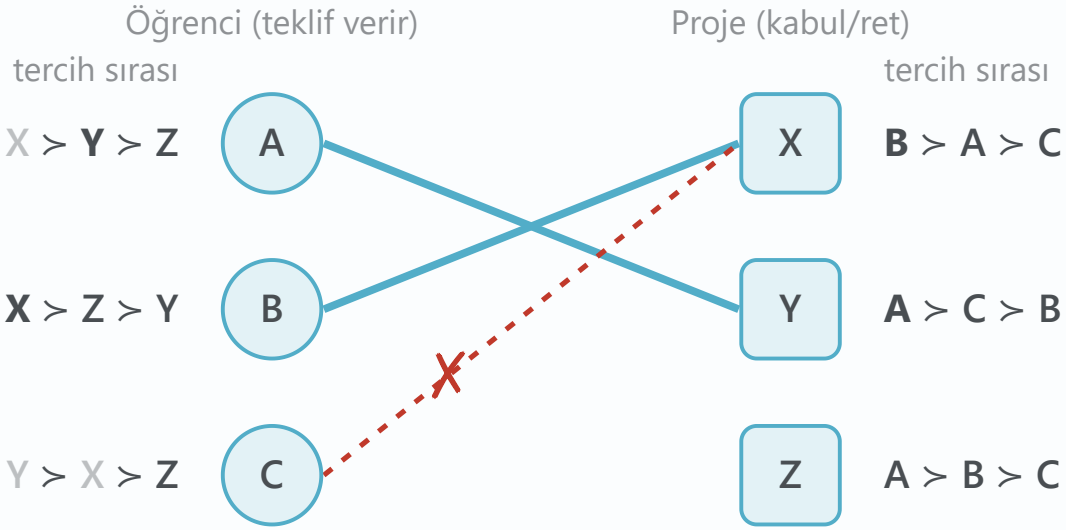
Tur 2 kararı — Y: A tutulur, C reddedilir (1. sıradaki A daha iyi).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



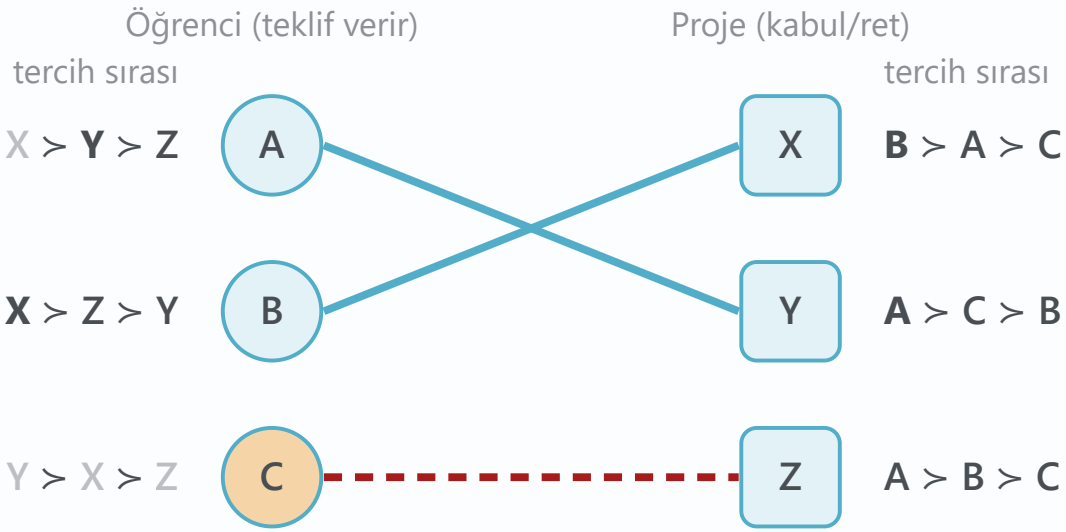
Tur 3 teklifleri: C→X (serbest her öğrenci, henüz reddetmemiş en iyi projesine).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



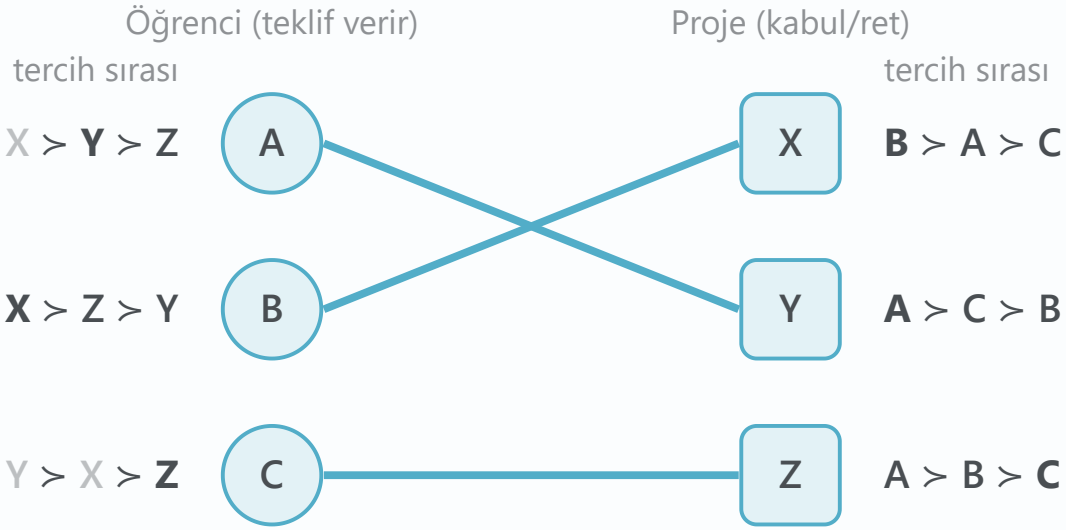
Tur 3 kararı — X: B tutulur, C reddedilir (1. sıradaki B daha iyi).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



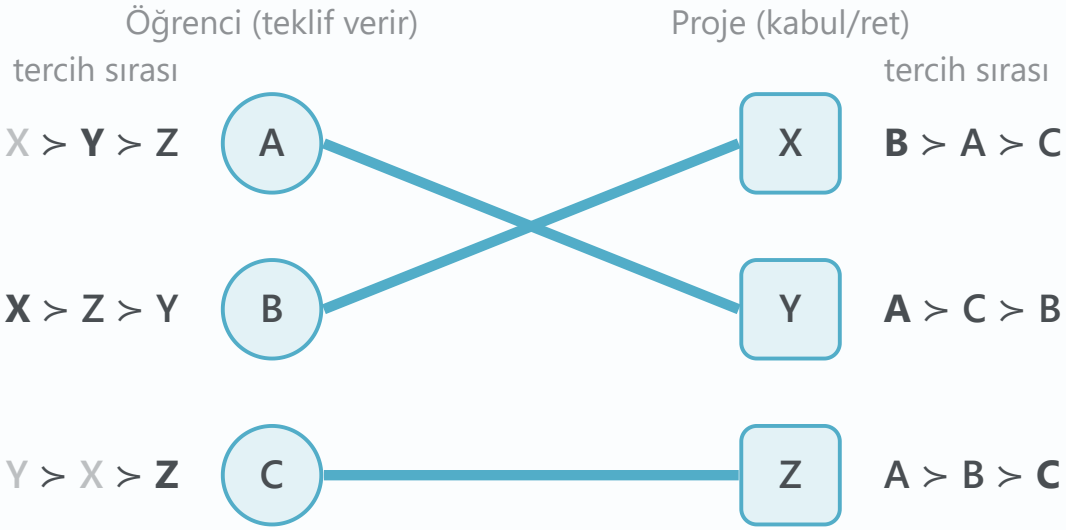
Tur 4 teklifleri: C→Z (serbest her öğrenci, henüz reddetmemiş en iyi projesine).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



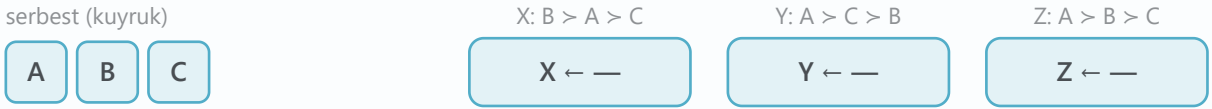
Tur 4 kararı — Z: C kabul (geçici).

Gale–Shapley: Ertelenmiş Kabul (Teklif Turları)



Herkes eşleşti → dur. **A–Y, B–X, C–Z**. Kontrol: A, X'i ister ama X, B'yi A'ya tercih eder; C'yi ise X de Y de istemez → engelleyici çift yok, **kararlı**.

Kod İzleme: `gale_shapley(ogr, proj)`



Herkes serbest, projeler boş. `proj[p][s]` : p'nin s'ye verdiği sıra (0 = en iyi).

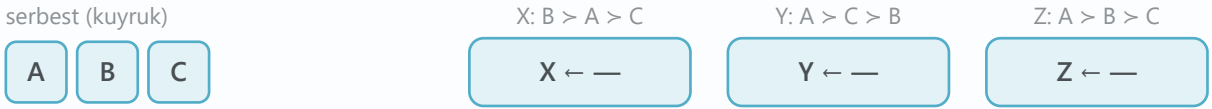
Adım 1 / 15

```
1  def gale_shapley(ogr, proj):
▶    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]        # sıradaki
7      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = [A, B, C]` `es = {}`

Kod İzleme: `gale_shapley(ogr, proj)`



Her öğrencinin sıradaki tercihi 0. indis (en sevdiği proje).

Adım 2 / 15

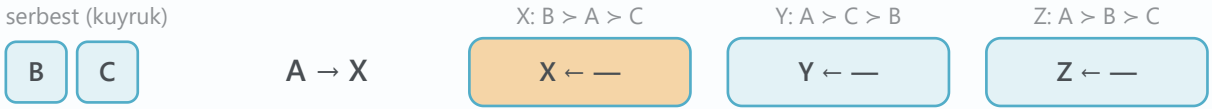
```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
▶    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]        # sıradaki
7      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = [A, B, C]` `es = {}`

`sira = {A: 0, B: 0, C: 0}`

Kod İzleme: `gale_shapley(ogr, proj)`



Kuyruktan **A** çıkar ve sıradaki tercihi **X**'e teklif verir.


```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
▶      s = serbest.pop(0)
▶      p = ogr[s][sira[s]]        # sıradaki
▶      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = [B, C]` `es = {}`

`sira = {A: 1, B: 0, C: 0}` `s = A`

`p = X`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)

B C

A → X

X: B ≻ A ≻ C

X ← A

Y: A ≻ C ≻ B

Y ← —

Z: A ≻ B ≻ C

Z ← —

X boş (r = None) → A **geçici olarak kabul** edilir.

Adım 4 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]          # sıradaki
7      sira[s] += 1
▶      r = es.get(p)                # p'deki aday
▶      if r is None or proj[p][s] < proj[p][r]:
▶        es[p] = s                  # kabul
▶        if r: serbest.append(r)
12     else: serbest.append(s)      # red
13   return es
```

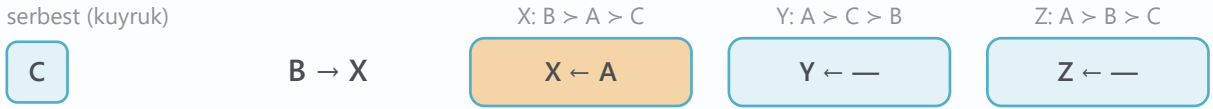
DEĞİŞKENLER

`serbest = [B, C]` `es = {X: A}`

`sira = {A: 1, B: 0, C: 0}` `s = A`

`p = X` `r = None`

Kod İzleme: `gale_shapley(ogr, proj)`



Kuyruktan **B** çıkar ve sıradaki tercihi **X**'e teklif verir.


```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
▶      s = serbest.pop(0)
▶      p = ogr[s][sira[s]]        # sıradaki
▶      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = [C]` `es = {X: A}` `sira = {A: 1, B: 1, C: 0}` `s = B` `p = X`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)	A reddedildi	X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
C, A	B → X	X ← B	Y ← —	Z ← —

X'in elinde A vardı; X için B daha iyi (sıra 0 < 1) → B kabul, **A kuyruğa döner**.

Adım 6 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]          # sıradaki
7      sira[s] += 1
▶      r = es.get(p)                # p'deki aday
▶      if r is None or proj[p][s] < proj[p][r]:
▶        es[p] = s                  # kabul
▶        if r: serbest.append(r)
12     else: serbest.append(s)      # red
13   return es
```

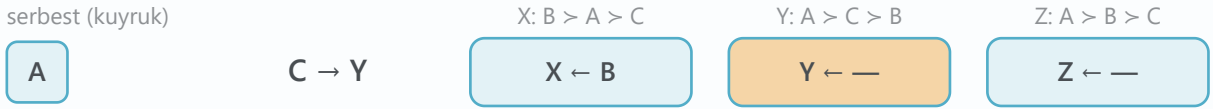
DEĞİŞKENLER

`serbest = [C, A]` `es = {X: B}`

`sira = {A: 1, B: 1, C: 0}` `s = B`

`p = X` `r = A`

Kod İzleme: `gale_shapley(ogr, proj)`



Kuyruktan **C** çıkar ve sıradaki tercihi **Y**'ye teklif verir.

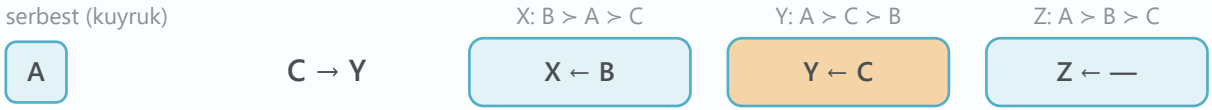
Adım 7 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
▶      s = serbest.pop(0)
▶      p = ogr[s][sira[s]]        # sıradaki
▶      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = [A]` `es = {X: B}` `sira = {A: 1, B: 1, C: 1}` `s = C` `p = Y`

Kod İzleme: `gale_shapley(ogr, proj)`



Y boş (r = None) → C **geçici olarak kabul** edilir.


```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]          # sıradaki
7      sira[s] += 1
▶      r = es.get(p)                # p'deki aday
▶      if r is None or proj[p][s] < proj[p][r]:
▶        es[p] = s                  # kabul
▶        if r: serbest.append(r)
12     else: serbest.append(s)      # red
13   return es
```

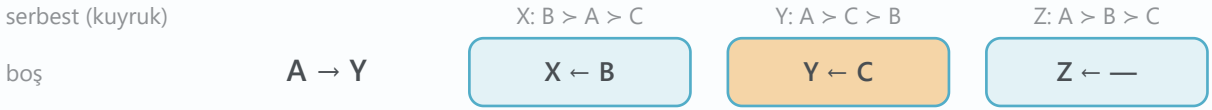
DEĞİŞKENLER

`serbest = [A]` `es = {X: B, Y: C}`

`sira = {A: 1, B: 1, C: 1}` `s = C`

`p = Y` `r = None`

Kod İzleme: `gale_shapley(ogr, proj)`



Kuyruktan **A** çıkar ve sıradaki tercihi **Y**'ye teklif verir.

Adım 9 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
▶      s = serbest.pop(0)
▶      p = ogr[s][sira[s]]        # sıradaki
▶      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = []` `es = {X: B, Y: C}` `sira = {A: 2, B: 1, C: 1}` `s = A` `p = Y`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)	C reddedildi	X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
C	A → Y	X ← B	Y ← A	Z ← —

Y'nin elinde C vardı; Y için A daha iyi (sıra 0 < 1) → A kabul, **C kuyruğa döner**.


```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]          # sıradaki
7      sira[s] += 1
▶      r = es.get(p)                # p'deki aday
▶      if r is None or proj[p][s] < proj[p][r]:
▶        es[p] = s                  # kabul
▶        if r: serbest.append(r)
12     else: serbest.append(s)      # red
13   return es
```

DEĞİŞKENLER

`serbest = [C]` `es = {X: B, Y: A}` `sira = {A: 2, B: 1, C: 1}` `s = A` `p = Y` `r = C`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)		X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
boş	C → X	X ← B	Y ← A	Z ← —

Kuyruktan **C** çıkar ve sıradaki tercihi **X**'e teklif verir.

Adım 11 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
▶      s = serbest.pop(0)
▶      p = ogr[s][sira[s]]        # sıradaki
▶      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = []` `es = {X: B, Y: A}` `sira = {A: 2, B: 1, C: 2}` `s = C` `p = X`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)	C reddedildi	X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
C	C → X	X ← B	Y ← A	Z ← —

X, elindeki B'yi C'ye tercih eder → **C reddedilir** ve kuyruğa döner.

Adım 12 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]          # sıradaki
7      sira[s] += 1
▶      r = es.get(p)                # p'deki aday
▶      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                  # kabul
11       if r: serbest.append(r)
▶      else: serbest.append(s)      # red
13   return es
```

DEĞİŞKENLER

`serbest = [C]` `es = {X: B, Y: A}` `sira = {A: 2, B: 1, C: 2}` `s = C` `p = X` `r = B`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)		X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
boş	C → Z	X ← B	Y ← A	Z ← —

Kuyruktan **C** çıkar ve sıradaki tercihi **Z**'ye teklif verir.

Adım 13 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
▶      s = serbest.pop(0)
▶      p = ogr[s][sira[s]]        # sıradaki
▶      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
13   return es
```

DEĞİŞKENLER

`serbest = []` `es = {X: B, Y: A}` `sira = {A: 2, B: 1, C: 3}` `s = C` `p = Z`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)		X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
boş	C → Z	X ← B	Y ← A	Z ← C

Z boş (r = None) → C **geçici olarak kabul** edilir.

Adım 14 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]          # sıradaki
7      sira[s] += 1
▶      r = es.get(p)                # p'deki aday
▶      if r is None or proj[p][s] < proj[p][r]:
▶        es[p] = s                  # kabul
▶        if r: serbest.append(r)
12     else: serbest.append(s)      # red
13   return es
```

DEĞİŞKENLER

`serbest = []` `es = {X: B, Y: A, Z: C}` `sira = {A: 2, B: 1, C: 3}` `s = C` `p = Z` `r = None`

Kod İzleme: `gale_shapley(ogr, proj)`

serbest (kuyruk)

boş

X: B ≻ A ≻ C	Y: A ≻ C ≻ B	Z: A ≻ B ≻ C
X ← B	Y ← A	Z ← C

Kuyruk boş: 6 teklifle bitti. Sonuç eşzamanlı turlardakiyle aynı: A–Y, B–X, C–Z.

Adım 15 / 15

```
1  def gale_shapley(ogr, proj):
2    serbest, es = list(ogr), {}
3    sira = {s: 0 for s in ogr}
4    while serbest:
5      s = serbest.pop(0)
6      p = ogr[s][sira[s]]        # sıradaki
7      sira[s] += 1
8      r = es.get(p)              # p'deki aday
9      if r is None or proj[p][s] < proj[p][r]:
10       es[p] = s                # kabul
11       if r: serbest.append(r)
12     else: serbest.append(s)    # red
▶    return es
```

DEĞİŞKENLER

`serbest = []` `es = {X: B, Y: A, Z: C}`

PARASIZ MEKANİZMALAR

Gale–Shapley'in Özellikleri

Her zaman kararlı

Algoritma sonlanır ve sonuç kararlıdır (Gale ve Shapley, 1962). En fazla n^2 teklif → $O(n^2)$.

Teklif veren tarafa en iyi

Teklif verenler, tüm kararlı eşleştirmeler içinde en iyi eşlerini alır; teklif alan taraf ise en kötüsünü.

Teklif verenler için DSIC

Teklif veren taraf için dürüst tercih sırası bildirmek baskın stratejidir. Teklif alan taraf ise bazen yalanla kazanabilir.

Uygulamalar

ABD tıp ihtisası eşleştirmesi (NRMP), Boston ve New York okul seçimi. 2012 Nobel: Roth ve Shapley.

Örneğimizde tek bir kararlı eşleştirme var (3! = 6 eşleştirmenin hepsi denendiğinde); projeler teklif verse de sonuç A–Y, B–X, C–Z olur.

Teklif Alan Taraf Yalanla Kazanabilir

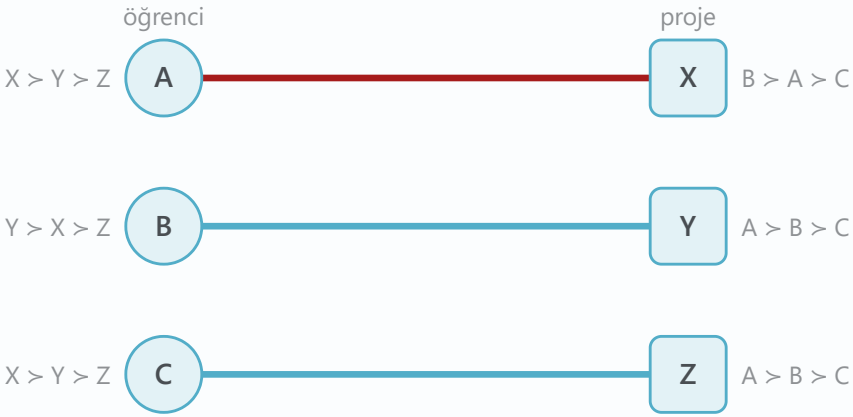


Yeni örnek. Öğrenciler teklif verir. Projelerin gerçek tercihleri: X: B ≻ A ≻ C; Y ve Z: A ≻ B ≻ C.

Adım 1 / 4

Bu örnekte iki kararlı eşleştirme var (3! = 6 eşleştirmenin hepsi denendiğinde): öğrencilere en iyi gelen A–X, B–Y, C–Z ve projelere en iyi gelen A–Y, B–X, C–Z. X'in yalanı sonucu projelere en iyi gelene taşır. Hiçbir kararlı mekanizma iki taraf için birden DSIC olamaz (Roth, 1982).

Teklif Alan Taraf Yalanla Kazanabilir



Dürüst bildirimle teklifler A→X, B→Y, C→X, C→Y, C→Z → **A–X, B–Y, C–Z**. X, 2. sıradaki A'yı alır.

Bu örnekte iki kararlı eşleştirme var (3! = 6 eşleştirmenin hepsi denendiğinde): öğrencilere en iyi gelen A–X, B–Y, C–Z ve projelere en iyi gelen A–Y, B–X, C–Z. X'in yalanı sonucu projelere en iyi gelene taşır. Hiçbir kararlı mekanizma iki taraf için birden DSIC olamaz (Roth, 1982).

Teklif Alan Taraf Yalanla Kazanabilir

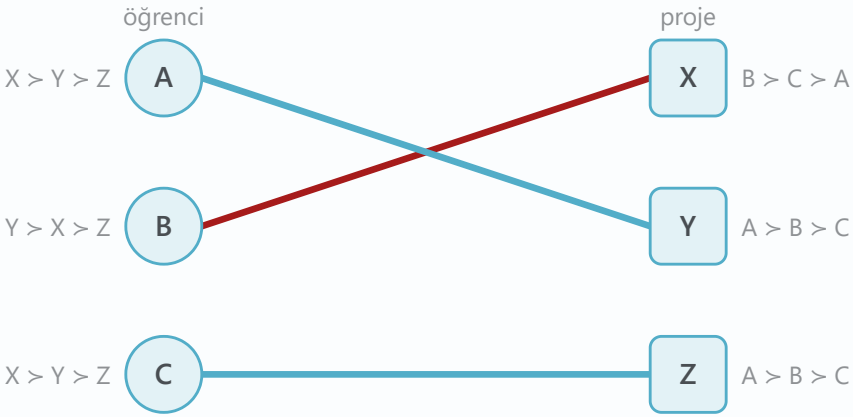


X yalan söyler: listesini **B ≻ C ≻ A** diye bildirir (A'yı en sona atar).

Adım 3 / 4

Bu örnekte iki kararlı eşleştirme var (3! = 6 eşleştirmenin hepsi denendiğinde): öğrencilere en iyi gelen A–X, B–Y, C–Z ve projelere en iyi gelen A–Y, B–X, C–Z. X'in yalanı sonucu projelere en iyi gelene taşır. Hiçbir kararlı mekanizma iki taraf için birden DSIC olamaz (Roth, 1982).

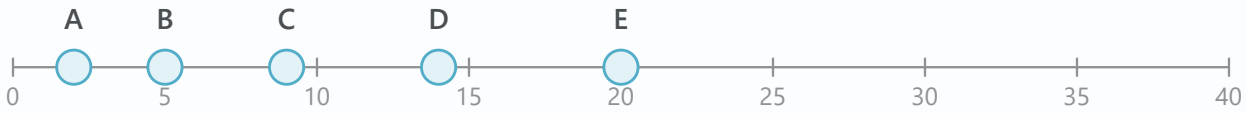
Teklif Alan Taraf Yalanla Kazanabilir



Teklifler A→X, B→Y, C→X, A→Y, B→X, C→Y, C→Z: A'nın reddi zincirleme B'yi X'e getirir → **B–X**. X, gerçek 1. tercihini aldı: yalan kazandırdı.

Bu örnekte iki kararlı eşleştirme var (3! = 6 eşleştirmenin hepsi denendiğinde): öğrencilere en iyi gelen A–X, B–Y, C–Z ve projelere en iyi gelen A–Y, B–X, C–Z. X'in yalanı sonucu projelere en iyi gelene taşır. Hiçbir kararlı mekanizma iki taraf için birden DSIC olamaz (Roth, 1982).

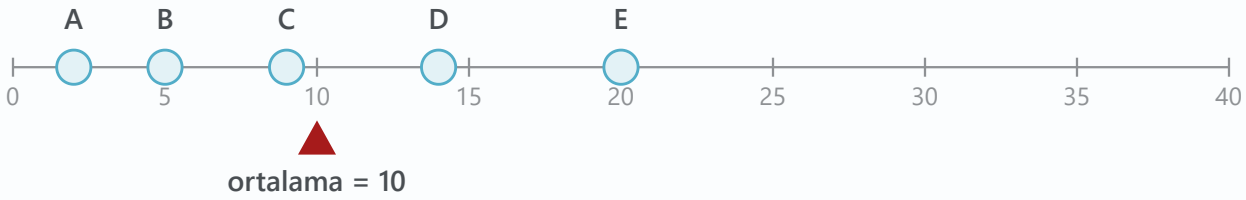
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı

Beş köy bir yol üzerinde: 2, 5, 9, 14, 20 km. Tek bir tesis (hastane) kurulacak; her köy tesisin **kendisine yakın** olmasını ister (maliyet = uzaklık).

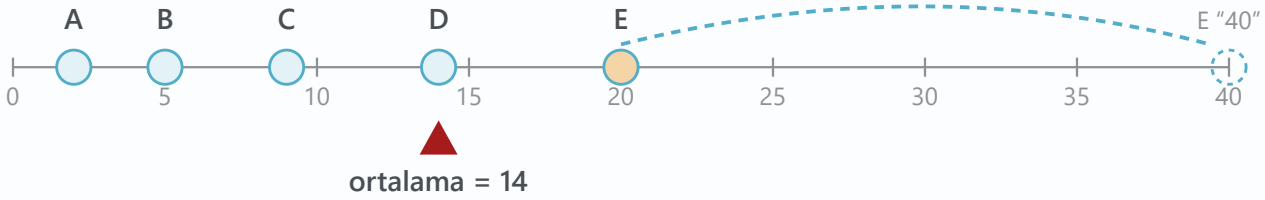
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı
Ortalama, herkes dürüst	10	**10**

Ortalama mekanizması: (2+5+9+14+20)/5 = 10. E'nin uzaklığı 10 km.

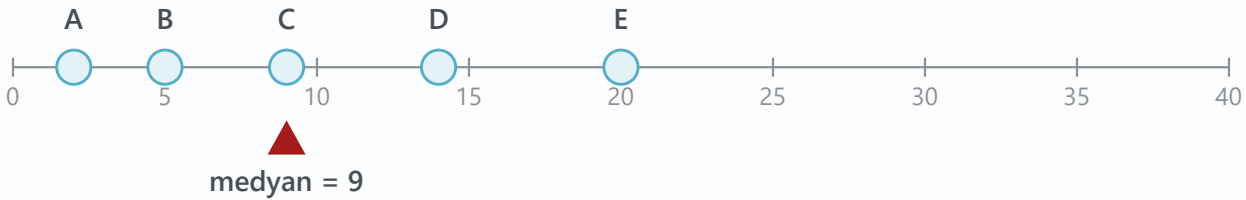
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı
Ortalama, herkes dürüst	10	**10**
Ortalama, E "40" der	14	**6**

E yalan söyleyip 40 bildirir: ortalama (2+5+9+14+40)/5 = 14. E'nin uzaklığı **10** → **6**. Yalan kazandırdı: ortalama manipüle edilebilir.

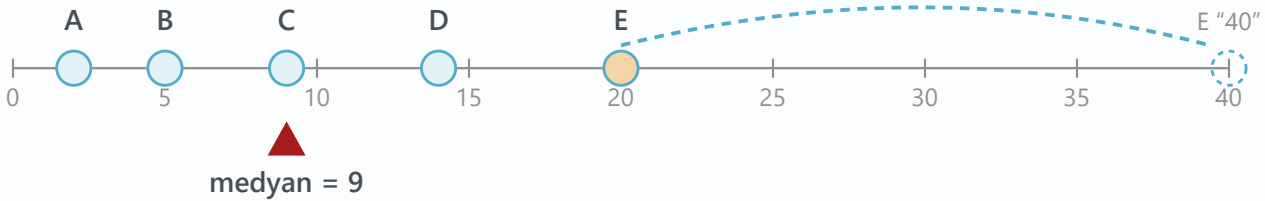
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı
Ortalama, herkes dürüst	10	**10**
Ortalama, E "40" der	14	**6**
Medyan, herkes dürüst	9	**11**

Medyan mekanizması: bildirimler sıralanır, ortadaki seçilir → 9. E'nin uzaklığı 11 km.

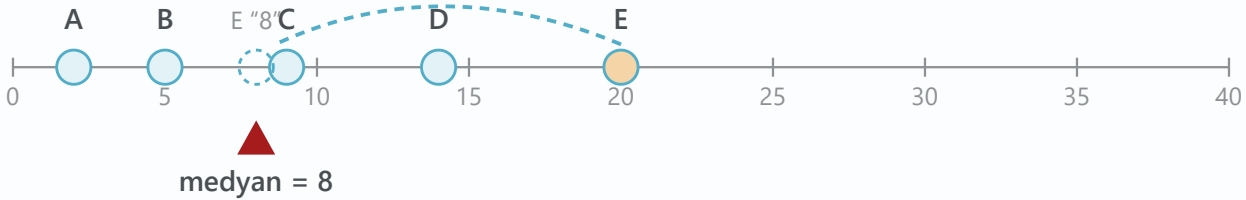
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı
Ortalama, herkes dürüst	10	**10**
Ortalama, E "40" der	14	**6**
Medyan, herkes dürüst	9	**11**
Medyan, E "40" der	9	**11**

E 40 bildirir: sıralama 2, 5, 9, 14, 40 → medyan **yine 9**. Uç değer medyanı çekemez.

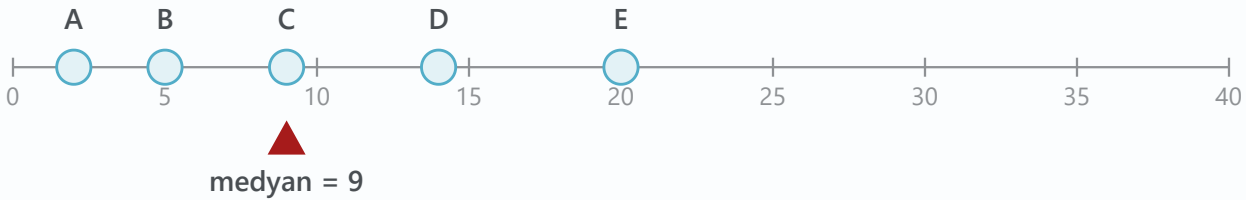
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı
Ortalama, herkes dürüst	10	**10**
Ortalama, E "40" der	14	**6**
Medyan, herkes dürüst	9	**11**
Medyan, E "40" der	9	**11**
Medyan, E "8" der	8	**12**

E medyanın öbür yanına geçip 8 bildirir: medyan 8 olur, E'nin uzaklığı **11** → **12**. Daha kötü!

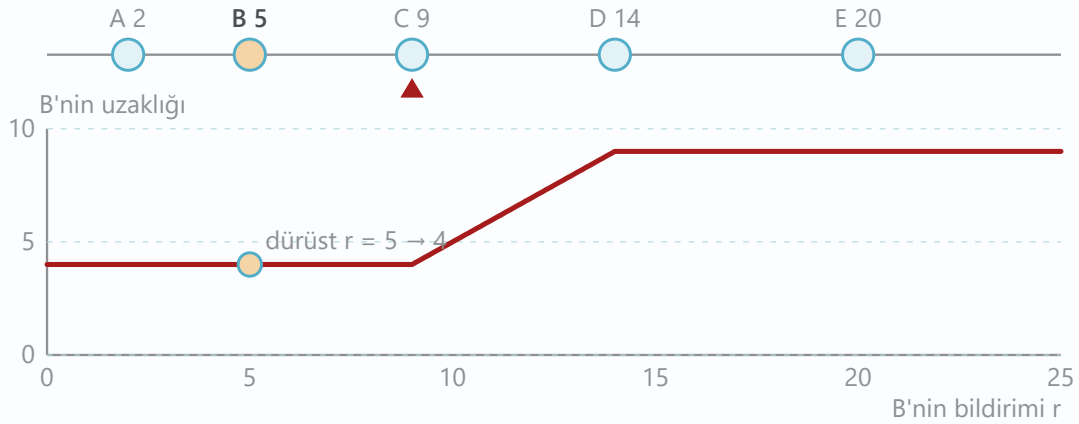
Tesis Yerleşimi: Ortalama mı, Medyan mı?



Durum	Tesis	E'nin uzaklığı
Ortalama, herkes dürüst	10	**10**
Ortalama, E “40” der	14	**6**
Medyan, herkes dürüst	9	**11**
Medyan, E “40” der	9	**11**
Medyan, E “8” der	8	**12**

Her köy için 0–60 arası tüm bildirimler denendiğinde medyanda **hiçbir yalan kazandırmıyor**. Üstelik toplam uzaklık medyanda 27, ortalamada 28: medyan toplam uzaklığı da en aza indirir.

Neden Medyan Stratejiye Dayanıklı?

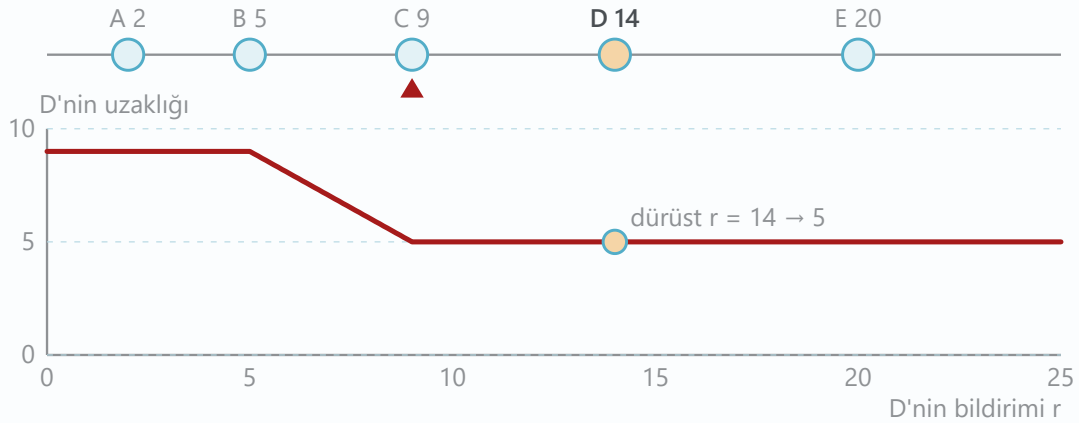


1) $x_i < m$ (B = 5, medyan 9): r ≤ 9 bildirimleri medyanı 9'da bırakır (uzaklık 4); r > 9 medyanı **sağa**, yani B'den uzağa iter. Hiçbir r 4'ün altına inemez.

Adım 1 / 3

Sonuç: her durumda dürüst bildirim baskın strateji → medyan mekanizması **stratejiye dayanıklıdır** (strategy-proof; Moulin, 1980). Aynı fikir tek tepeli tercihlerle oylamada da geçerlidir (Black).

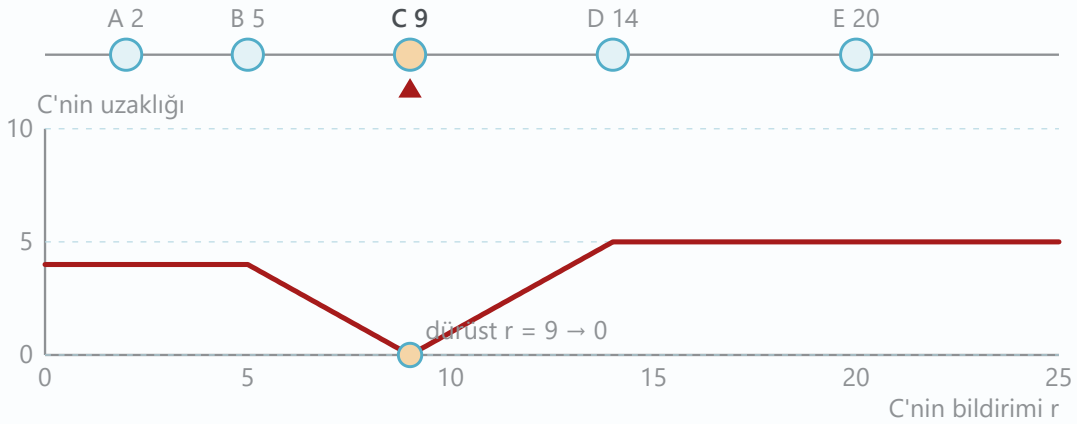
Neden Medyan Stratejiye Dayanıklı?



2) $x_i > m$ (D = 14): simetrik durum. r ≥ 9 medyanı değiştirmez (uzaklık 5); r < 9 medyanı sola, D'den uzağa çeker.

Sonuç: her durumda dürüst bildirim baskın strateji → medyan mekanizması **stratejiye dayanıklıdır** (strategy-proof; Moulin, 1980). Aynı fikir tek tepeli tercihlerle oylamada da geçerlidir (Black).

Neden Medyan Stratejiye Dayanıklı?



3) x_i = m (C = 9): tesis tam C'nin konumunda, uzaklık 0; her yalan medyanı ancak C'den uzaklaştırır.

Sonuç: her durumda dürüst bildirim baskın strateji → medyan mekanizması **stratejiye dayanıklıdır** (strategy-proof; Moulin, 1980). Aynı fikir tek tepeli tercihlerle oylamada da geçerlidir (Black).

Seri Diktatörlük ve Boston Mekanizması

Seri diktatörlük (serial dictatorship)

Sırası gelen, kalanlar içinden en sevdiğini alır. Kimse öncekilerin seçimini etkileyemez → DSIC ve Pareto optimal. Sıra kurayla belirlenirse **rastgele seri diktatörlük** (RSD).

Boston mekanizması (doğrudan kabul)

Her turda öğrenciler sıradaki tercihlerine başvurur. Her okul, o turda kendisini seçen öğrencilere boş koltuklarını öncelik sırasına göre verir; verilen koltuk sonraki turlarda geri alınmaz. İlk tercihini kaybeden, ikincisini dolmuş bulabilir.

Tercihler A: $X \succ Y \succ Z$, B: $X \succ Y \succ Z$, C: $Y \succ X \succ Z$. Öncelik X ve Z: A, B, C; Y: B, A, C. Her okul tek koltuk.

Tur	B dürüst	B "$Y \succ X \succ Z$" der
1	X: A, B → A; Y: C → C	X: A → A; Y: B, C → B
2	B → Y dolu	C → X dolu
3	B → Z	C → Z
Sonuç	A–X, **B–Z**, C–Y	A–X, **B–Y**, C–Z

B ikinci tercihini ilk sıraya yazarak Z yerine Y'yi alır: yalan kazandırdı.

Gale–Shapley aynı profilde dürüst bildirimle A–X, B–Y, C–Z verir. Abdulkadiroğlu ve Sönmez (2003), Boston mekanizmasında tercihleri olduğu gibi yazmanın öğrenciye zarar verebildiğini gösterdi; Boston şehri 2005'te okul yerleştirmesinde öğrenci önerili ertelenmiş kabule (Gale–Shapley) geçti.

Yurt Odası Takası: En İyi Takas Döngüleri (TTC)

- Her öğrencinin bir yurt odası var, para yok; her öğrenci başkalarının odalarını sıralar (Shapley ve Scarf, 1974; algoritma Gale'e atfedilir).
- **TTC**: herkes en sevdiği *kalan* odanın sahibini gösterir. Sonlu çizgede en az bir **döngü** vardır; döngüdekiler takas eder ve çıkar. Tekrarla.
- Dürüst tercih bildirmek baskın strateji (Roth, 1982); sonuç Pareto optimal ve çekirdektedir: kimse grup kurup daha iyisini yapamaz (Roth ve Postlewaite, 1977).
- Aynı kural nöbet ya da vardiya takasına da uyar; gerçek dünyada böbrek nakli eşleştirmelerinde de kullanılır (Roth, Sönmez ve Ünver, 2004).

Kişi	Tercih (oda sahibi)
A	$C \succ A \succ B$
B	$A \succ D \succ B$
C	$B \succ A \succ C$
D	$C \succ E \succ B \succ D$
E	$A \succ D \succ E$

Örnek: beş öğrenci yurt odası takası. Listede olmayan odalar kişinin kendi odasından kötüdür.

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A: —
B: —
C: —
D: —
E: —

Herkes kalanlar listesinde; henüz kimse takas etmedi.

Adım 1 / 13

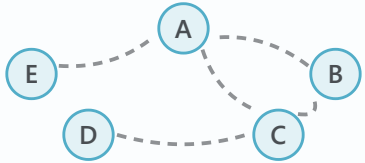
```
1  def ttc(pref):
▶    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5               if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [A, B, C, D, E]` `atama = {}`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A: —
B: —
C: —
D: —
E: —

Tur 1: her kalan kişi en sevdiği *kalan* odanın sahibini gösterir (oklar).


```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
▶    while kalan:
▶      g = {i: next(j for j in pref[i]
▶               if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

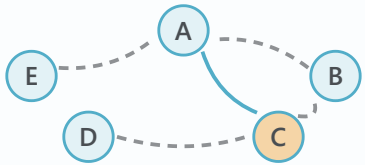
DEĞİŞKENLER

`kalan = [A, B, C, D, E]` `atama = {}`

`g = {A→C, B→A, C→B, D→C, E→A}`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A: —
B: —
C: —
D: —
E: —

A yola eklenir, ok C'yi gösterir.

Adım 3 / 13

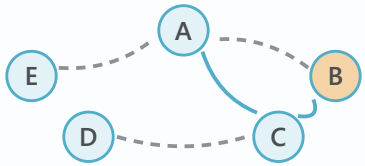
```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
▶        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [A, B, C, D, E]` `atama = {}`
`yol = [A]` `i = C`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A: —
B: —
C: —
D: —
E: —

C yola eklenir, ok B'yi gösterir.

Adım 4 / 13

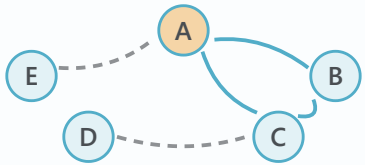
```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
▶        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [A, B, C, D, E]` `atama = {}`
`yol = [A, C]` `i = B`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A: —
B: —
C: —
D: —
E: —

B yola eklenir, ok A'yı gösterir; A zaten yolda → **döngü bulundu**.

Adım 5 / 13

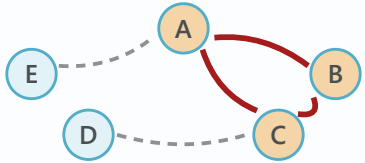
```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
▶        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [A, B, C, D, E]` `atama = {}`
`yol = [A, C, B]` `i = A`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A: —
B: —
C: —
D: —
E: —

Döngü: **A → C → B → A**. Döngüdeki herkes gösterdiği odayı alır.

Adım 6 / 13

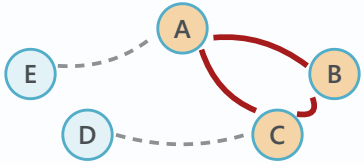
```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
▶      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [A, B, C, D, E]` `atama = {}`
`yol = [A, C, B]` `dongu = [A, C, B]`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D: —
E: —

Takas: A ← C'nin odası, C ← B'nin odası, B ← A'nın odası. Bu kişiler çıkar.

Adım 7 / 13

```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
▶      for k in dongu:              # takas
▶        atama[k] = g[k]; kalan.remove(k)
12   return atama
```

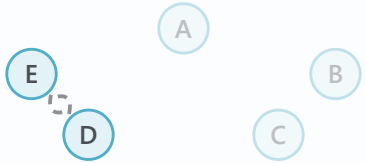
DEĞİŞKENLER

`kalan = [D, E]`

`atama = {A→C, C→B, B→A}`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D: —
E: —

Tur 2: her kalan kişi en sevdiği *kalan* odanın sahibini gösterir (oklar).

Adım 8 / 13

```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
▶    while kalan:
▶      g = {i: next(j for j in pref[i]
▶               if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

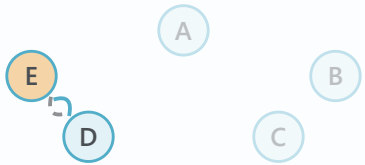
`kalan = [D, E]`

`atama = {A→C, C→B, B→A}`

`g = {D→E, E→D}`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D: —
E: —

D yola eklenir, ok E'yi gösterir.


```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
▶        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

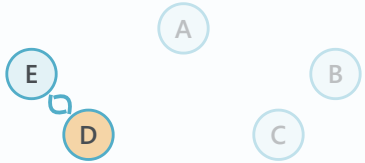
`kalan = [D, E]`

`atama = {A→C, C→B, B→A}` `yol = [D]`

`i = E`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D: —
E: —

E yola eklenir, ok D'yi gösterir; D zaten yolda → **döngü bulundu**.

Adım 10 / 13

```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
▶        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [D, E]`

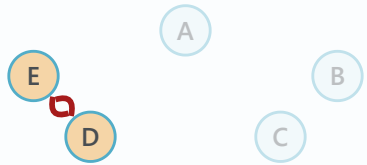
`atama = {A→C, C→B, B→A}`

`yol = [D, E]`

`i = D`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D: —
E: —

Döngü: **D → E → D**. Döngüdeki herkes gösterdiği odayı alır.

Adım 11 / 13

```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
▶      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
12   return atama
```

DEĞİŞKENLER

`kalan = [D, E]`

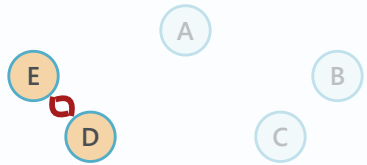
`atama = {A→C, C→B, B→A}`

`yol = [D, E]`

`dongu = [D, E]`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D ← E'nin odası
E ← D'nin odası

Takas: D ← E'nin odası, E ← D'nin odası. Bu kişiler çıkar.

Adım 12 / 13

```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
▶      for k in dongu:              # takas
▶        atama[k] = g[k]; kalan.remove(k)
12   return atama
```

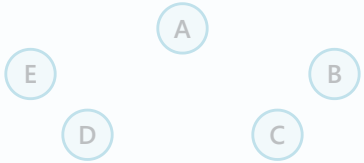
DEĞİŞKENLER

`kalan = []`

`atama = {A→C, C→B, B→A, D→E, E→D}`

Kod İzleme: `ttc(pref)`

A: C ≻ A ≻ B
B: A ≻ D ≻ B
C: B ≻ A ≻ C
D: C ≻ E ≻ B ≻ D
E: A ≻ D ≻ E



A ← C'nin odası
B ← A'nın odası
C ← B'nin odası
D ← E'nin odası
E ← D'nin odası

Bitti: A, B, C üçlü döngüyle; D ve E ikili döngüyle takas etti. Her kişinin tüm yalan listeleri denendiğinde hiçbiri daha iyi oda getirmiyor (DSIC); sonuç Pareto optimal.

Adım 13 / 13

```
1  def ttc(pref):
2    kalan, atama = list(pref), {}
3    while kalan:
4      g = {i: next(j for j in pref[i]
5                   if j in kalan) for i in kalan}
6      i, yol = kalan[0], []
7      while i not in yol:          # okları izle
8        yol.append(i); i = g[i]
9      dongu = yol[yol.index(i):]
10     for k in dongu:              # takas
11       atama[k] = g[k]; kalan.remove(k)
▶    return atama
```

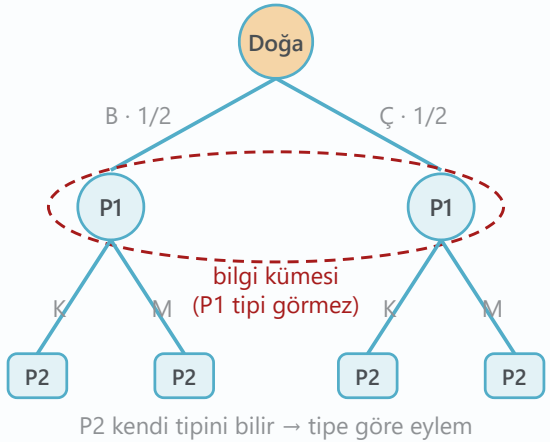
DEĞİŞKENLER

`kalan = []`

`atama = {A→C, C→B, B→A, D→E, E→D}`

Eksik Bilgi: Bayesçi Oyunlar (Bayesian Games)

- Şimdiye kadar tasarımcı tipleri bilmiyordu; oyuncular da çoğu zaman **birbirinin** tipini bilmez.
- Bayesçi oyun: oyuncular, tip kümeleri Θ_i, tipler üzerinde **ortak ön olasılık** (common prior) π, kazançlar $u_i(a, \theta)$.
- Önce şans (Doğa) her oyuncunun tipini seçer; her oyuncu yalnızca kendi tipini görür (Harsanyi dönüşümü).
- Strateji artık bir **kuraldır**: her tip için hangi eylemin seçileceği.
- **Bayes–Nash dengesi**: her oyuncu, her tipinde, diğerlerinin stratejilerine karşı *beklenen* kazancını maksimize eder.



Örnek oyun: P2'nin tipi B (buluşmak ister) ya da Ç (kaçınmak ister).

Mekanizma bağlantısı: açıklama ilkesinin Bayesçi sürümü → Bayes–Nash dengesinde uygulanabilen her sonuç, BIC bir doğrudan mekanizmayla da uygulanabilir.

Adım Adım: Bayes–Nash Dengesini Bulmak

Tip B · olasılık 1/2	P2: K	P2: M
P1: K	(2, 1)	(0, 0)
P1: M	(0, 0)	(1, 2)

Tip Ç · olasılık 1/2	P2: K	P2: M
P1: K	(2, 0)	(0, 2)
P1: M	(0, 1)	(1, 0)

Aday profil	P1: K beklenen	P1: M beklenen	Denge?

P1 konser (K) ya da maç (M) seçer ve K'yi tercih eder. P2 ya **B** (buluşmak ister) ya da **Ç** (kaçınmak ister) tipindedir; P1 tipi bilmez, olasılıklar 1/2–1/2.

Hücrelerde (P1 kazancı, P2 kazancı). 2 × 2 × 2 = 8 saf strateji profilinin hepsi denendi.

Adım Adım: Bayes–Nash Dengesini Bulmak

Tip B · olasılık 1/2	P2: K	P2: M
P1: K	(2, 1)	(0, 0)
P1: M	(0, 0)	(1, 2)

Tip Ç · olasılık 1/2	P2: K	P2: M
P1: K	(2, 0)	(0, 2)
P1: M	(0, 1)	(1, 0)

Aday profil	P1: K beklenen	P1: M beklenen	Denge?

Aday: P1 = K. P2'nin en iyi tepkisi tipe göre değişir: B tipi K (1 > 0), Ç tipi M (2 > 0) seçer → s_2 = (B→K, Ç→M).

Hücrelerde (P1 kazancı, P2 kazancı). 2 × 2 × 2 = 8 saf strateji profilinin hepsi denendi.

Adım Adım: Bayes–Nash Dengesini Bulmak

Tip B · olasılık 1/2	P2: K	P2: M
P1: K	(2, 1)	(0, 0)
P1: M	(0, 0)	(1, 2)

Tip Ç · olasılık 1/2	P2: K	P2: M
P1: K	(2, 0)	(0, 2)
P1: M	(0, 1)	(1, 0)

Aday profil	P1: K beklenen	P1: M beklenen	Denge?
P1 = K; B→K, Ç→M	½·2 + ½·0 = 1	½·0 + ½·1 = 0,5	**✓ evet**

P1'in beklenen kazancı: K → ½·2 + ½·0 = **1**; M → ½·0 + ½·1 = **0,5**. K en iyisi → **(K; B→K, Ç→M) Bayes–Nash dengesi**.

Hücrelerde (P1 kazancı, P2 kazancı). 2 × 2 × 2 = 8 saf strateji profilinin hepsi denendi.

Adım Adım: Bayes–Nash Dengesini Bulmak

Tip B · olasılık 1/2	P2: K	P2: M
P1: K	(2, 1)	(0, 0)
P1: M	(0, 0)	(1, 2)

Tip Ç · olasılık 1/2	P2: K	P2: M
P1: K	(2, 0)	(0, 2)
P1: M	(0, 1)	(1, 0)

Aday profil	P1: K beklenen	P1: M beklenen	Denge?
P1 = K; B→K, Ç→M	½·2 + ½·0 = 1	½·0 + ½·1 = 0,5	✓ evet

Öbür aday: P1 = M. O zaman B tipi M (2 > 0), Ç tipi K (1 > 0) seçer → s_2 = (B→M, Ç→K).

Hücrelerde (P1 kazancı, P2 kazancı). 2 × 2 × 2 = 8 saf strateji profilinin hepsi denendi.

Adım Adım: Bayes–Nash Dengesini Bulmak

Tip B · olasılık 1/2	P2: K	P2: M
P1: K	(2, 1)	(0, 0)
P1: M	(0, 0)	(1, 2)

Tip Ç · olasılık 1/2	P2: K	P2: M
P1: K	(2, 0)	(0, 2)
P1: M	(0, 1)	(1, 0)

Aday profil	P1: K beklenen	P1: M beklenen	Denge?
P1 = K; B→K, Ç→M	½·2 + ½·0 = 1	½·0 + ½·1 = 0,5	✓ **evet**
P1 = M; B→M, Ç→K	½·0 + ½·2 = 1	½·1 + ½·0 = 0,5	✗ **K'ye sapar**

P1: K → ½·0 + ½·2 = 1; M → ½·1 + ½·0 = 0,5. P1 M'den K'ye sapar → denge değil. **Tek saf Bayes–Nash dengesi: (K; B→K, Ç→M)**.

Hücrelerde (P1 kazancı, P2 kazancı). 2 × 2 × 2 = 8 saf strateji profilinin hepsi denendi.

Bayesçi Bakışla Mekanizmalar

Örnek: birinci fiyat

Dürüst teklif kazanınca fayda 0 bırakır; dengede herkes değerinin altında teklif verir. Bu denge Bayesçidir: rakiplerin değerleri hakkındaki inanca dayanır.

DSIC mi, BIC mi?

DSIC diğerlerinin inançlarından bağımsız ve sağlamdır. BIC daha zayıftır ama daha fazla hedefe izin verir: ör. verimli ve tam bütçe dengeli kamu malı mekanizması (d'Aspremont ve Gérard-Varet, 1979; Arrow, 1979).

En yüksek gelir

Satıcının beklenen gelirini maksimize eden mekanizma da Bayesçi bir sorudur; cevabı rezerv fiyatlı (satıcının belirlediği en düşük fiyatın altında satılmayan) bir açık artırmadır (Myerson, 1981).

Poker ve blöf

Rakibin eli (tipi) gizlidir; blöf, karışık Bayesçi stratejinin parçasıdır.

DEĞERLENDİRME

Tartışma: Davranış, Adalet ve Etik

İnsanlar her zaman teşviklere göre mi davranır?

- Kuram: oyuncular faydasını maksimize eder.
- Gerçekte duygular, ahlak, alışkanlık ve adalet duygusu da etkilidir (ör. Ültimatom oyununda düşük teklifler reddedilir).
- Teşvikler güçlüdür, ama tek belirleyici değildir; basit ve anlaşılır kurallar önemlidir.

Mekanizma tasarımında etik sorunlar

- Manipülasyon ve bilgi saklama; güçlü oyuncuların avantaj sağlaması.
- Adil olmayan gelir dağılımı; ödeme gücü olmayanların dışlanması.
- Etik sınırlar toplum ve kurumlarca belirlenmeli: iyi mekanizma yalnızca verimli değil, adil ve etik de olmalıdır.

DEĞERLENDİRME

Özet: Bu Bölümün Mekanizmaları

Mekanizma	Para	Teşvik özelliği	Ne sağlar?
Orantılı maliyet paylaşımı	var	IC değil (beleşçilik)	—
VCG (Clarke pivotu)	var	DSIC, IR	Verimli sonuç; bütçe fazlası olabilir
Vickrey (tek nesne VCG)	var	DSIC, IR	En yüksek değerli kazanır
Gale–Shapley	yok	Teklif verenler için DSIC	Kararlı eşleştirme, $O(n^2)$
Medyan (tesis yerleşimi)	yok	DSIC (stratejiye dayanıklı)	Toplam uzaklık en az
TTC (ev takası)	yok	DSIC	Pareto optimal, çekirdekte
Bayesçi mekanizmalar	var	BIC (Bayes–Nash)	Beklenen hedefler (ör. gelir)

Açıklama ilkesi sayesinde doğrudan, dürüst bildirimli mekanizmalara odaklanmak yeterlidir; ama IC + IR + verimlilik + bütçe dengesi her zaman birlikte sağlanamaz.

DEĞERLENDİRME

Örnek Problemler

1. İki özdeş nesne, üç teklifçi (birer nesne ister), değerler 10, 8, 5. VCG'de kim kazanır, kim ne öder?
2. Maliyet paylaşımı örneğinde (60; 30, 25, 20) diğerleri dürüstse Bora'nın en kârlı bildirimi nedir?
3. Köyler 3, 7, 8, 12, 30: medyan ve ortalama? Ortalamada 30'daki köy tesisi kendine çekmek için ne bildirir?
4. Gale–Shapley örneğinde projeler teklif verseydi sonuç ne olurdu?
5. Bayesçi örnekte B tipinin olasılığı 1/4 olsaydı (K; B→K, Ç→M) yine denge olur muydu?
6. TTC, tercihler A: $B \succ C \succ A$, B: $A \succ C \succ B$, C: $A \succ B \succ C$. Kim hangi odayı alır?

▶ **Cevapları göster**

DEĞERLENDİRME

Etkileşimli Oyunlar ve Simülatörler

Teşvik Uyumluluğu Laboratuvarı

Dürüst ve stratejik bildirimlerde ödemeler ve fayda.

Kararlı Eşleştirme

Adım adım ertelenmiş kabul; kararlılık ve toplam memnuniyet.

Ültimatom Oyunu

Pazarlık, ret ve adalet duygusu: teşviklerin sınırı.

Tesis Yerleşimi

Ortalama ve medyan kuralı: hangisinde yalan kazandırır?

DEĞERLENDİRME

Kaynaklar

- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (ed.), *Algorithmic Game Theory*, Cambridge University Press, 2007.
- T. Roughgarden, *Twenty Lectures on Algorithmic Game Theory*, Cambridge University Press, 2016.
- Y. Shoham, K. Leyton-Brown, *Multiagent Systems*, Cambridge University Press, 2009.
- D. Gale, L. S. Shapley, "College Admissions and the Stability of Marriage", *American Mathematical Monthly*, 1962.
- H. Moulin, "On Strategy-Proofness and Single Peakedness", *Public Choice*, 1980 · M. J. Osborne, *An Introduction to Game Theory*, Oxford University Press, 2004 (Bayesçi oyunlar).
- R. Myerson, "Optimal Auction Design", *Mathematics of Operations Research*, 1981 · L. Shapley, H. Scarf, "On Cores and Indivisibility", *Journal of Mathematical Economics*, 1974.
- A. E. Roth, T. Sönmez, M. U. Ünver, "Kidney Exchange", *Quarterly Journal of Economics*, 2004.
- A. E. Roth, A. Postlewaite, "Weak versus Strong Domination in a Market with Indivisible Goods", *J. Math. Econ.* 4(2), 1977 · C. d'Aspremont, L.-A. Gérard-Varet, "Incentives and Incomplete Information", *J. Public Econ.* 11(1), 1979 · A. Abdulkadiroğlu, T. Sönmez, "School Choice: A Mechanism Design Approach", *AER* 93(3), 2003.
- E. Maskin, "Nash Equilibrium and Welfare Optimality", *Review of Economic Studies* 66(1), 1999.
- A. R. Karlin, Y. Peres, *Game Theory, Alive*, AMS, 2017 (mekanizma tasarımı, kararlı eşleştirme) · D. Easley, J. Kleinberg, *Networks, Crowds, and Markets*, Cambridge, 2010 (Bölüm 10 ve 15).
- Ders sayfaları: Roughgarden CS364A · Shah CSC304 · Procaccia 15-896 · Dinitz AGT

Son

Bölüm 4: Mekanizma Tasarımı

Sonraki bölüm: Bölüm 5: Açık Artırmalar

Çalışma soruları · Terim sözlüğü