Bölüm 10: Çoklu Ajan Sistemleri

Algoritmik Oyun Kuramı

Ajan özellikleri | İşbirliği ve rekabet | Merkezî mi dağıtık mı | Sözleşme ağı | Koalisyon oyunları | Shapley değeri | Çekirdek | Güç endeksleri | Kamu malı ve beleşçilik

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/agt/slides/Bolum_10_Coklu_Ajan_Sistemleri.html

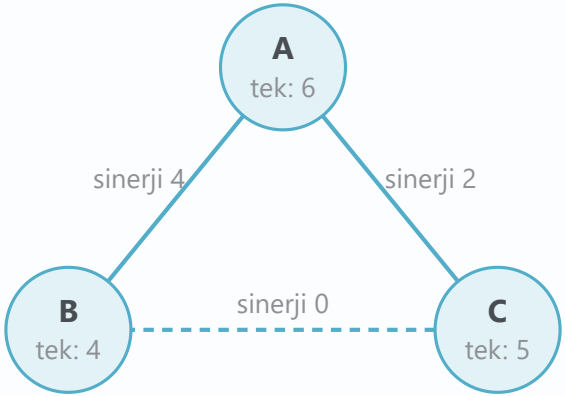
Oyunla Başlangıç: Shapley Koalisyon Oyunu

Oyunun kuralı

Her oyuncunun tek başına bir değeri vardır; bir koalisyonun değeri, üyelerin tek başına değerleri ile içindeki her ikilinin sinerji bonusunun toplamıdır. Büyük koalisyonun değeri v(N) oyunculara paylaştırılır.

Merak sorusu

- Yandaki örnek turda 21 birim eşit bölünürse (7'şer) A haksızlığa uğrar mı?
- Tek başına en değerli oyuncu her zaman en büyük payı mı almalı?



v(ABC) = 15 + 6 = 21

Shapley Koalisyon Oyunu →

Öğrenme Çıktıları

Ajan ve MAS

Ajanın temel özelliklerini ve çoklu ajan sistemlerinde işbirliği, rekabet ve koordinasyon sorunlarını açıklamak.

Görev tahsisi

Sözleşme ağı protokolünü (duyuru, teklif, ödül) küçük bir örnek üzerinde çalıştırmak ve optimumla karşılaştırmak.

Koalisyon oyunları

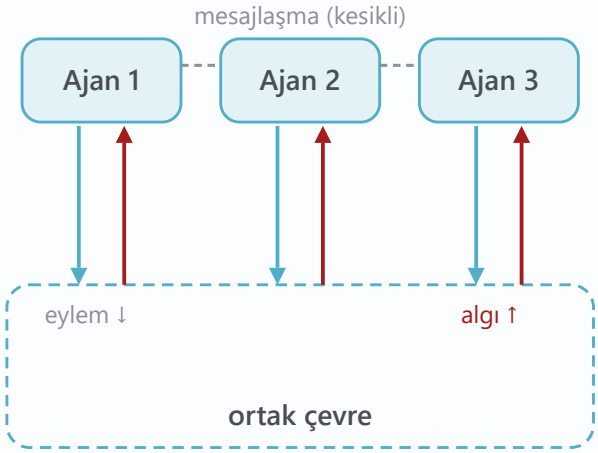
Karakteristik fonksiyonu yazmak; Shapley değerini tüm sıralamalardaki marjinal katkılardan hesaplamak; çekirdeği sınamak; ağırlıklı oylamada güç endeksini bulmak.

Kamu malı ve beleşçilik

Kamu malı oyununda katkı vermemenin neden baskın strateji olduğunu sayısal olarak göstermek ve çoklu ajan sistemlerindeki karşılıklarını tanımak.

Çoklu Ajan Sistemi (Multi-Agent System, MAS) Nedir?

- Aynı çevrede **algılayan, karar veren ve eylem yapan** birden fazla otonom ajandan oluşan sistem.
- Ajanlar ortak hedefe sahip olabilir ya da kendi çıkarları doğrultusunda **rekabet** edebilir.
- Her ajanın bilgisi **yereldir**; bu yüzden koordinasyon ve iletişim temel tasarım sorunlarıdır.
- Örnekler: robot takımları, otonom araç filoları, dağıtık sensör ağları, trafik ve ekonomi simülasyonları.



Oyun kuramı bağlantısı: ajanlar birbirinin kararından etkileniyorsa etkileşim bir **oyundur**; ajanları tasarlayan kişi ise kuralları koyan **mekanizma tasarımcısıdır**.

Ajan Kavramı ve Özellikleri

Wooldridge ve Jennings'in (1995) "zayıf ajan" tanımındaki dört özellik:

Otonomi (autonomy)
İnsan müdahalesi olmadan kendi kararını verir, kendi durumunu yönetir.
`drone rotasını kendisi seçer`

Reaktivite (reactivity)
Çevredeki değişikliği algılar ve zamanında tepki verir.
`engel görünce frenler`

Proaktivite (pro-activeness)
Yalnızca tepki vermez; hedefe ulaşmak için inisiyatif alır, plan yapar.
`şarjı bitmeden istasyona gider`

Sosyal yetenek (social ability)
Diğer ajanlarla iletişim kurar, pazarlık eder, işbirliği yapar.
`görev için teklif verir`

Oyun kuramında ajan = **rasyonel oyuncu**: tercihleri bir fayda fonksiyonuyla ifade edilir ve beklenen faydasını maksimize etmeye çalışır.

Ajan Nasıl Karar Verir? Mimariler

Mimari	Karar mekanizması	Güçlü yanı	Örnek
Reaktif	Durum → eylem kuralları; iç model yok	Hızlı, sağlam	Süpürge robotu, sürü davranışı
Planlayan (deliberative)	Çevre modeli + planlama / arama	Uzun vadeli hedefler	Keşif robotu, lojistik planlayıcı
BDI	İnanç (belief), arzu (desire), niyet (intention)	İnsan benzeri akıl yürütme	Hava trafik yönetimi, sanal asistan
Öğrenen	Deneyimden politika öğrenir	Bilinmeyen çevreye uyum	Oyun oynayan ajanlar
Hibrit	Reaktif alt katman + planlayıcı üst katman	Hız ve öngörü birlikte	Otonom araç

Mimari, ajanın içini anlatır; oyun kuramı ise ajanlar arasındaki **etkileşimi** inceler.

Gerçek Hayat Uygulamaları

Otonom araçlar
Kavşakta geçiş sırası, şerit değiştirme ve konvoy sürüşü için koordinasyon.

Drone takımları
Arama-kurtarmada bölgeleri paylaşma: görev tahsisi problemi.

Depo robotları
Yüzlerce robotun sipariş toplaması; çarpışmasız rota ve iş dağıtımı.

Çevrim içi pazar yerleri
Alıcı ve satıcı ajanlar, fiyat teklifleri ve otomatik pazarlık.

Akıllı şebeke
Evler, depolar ve üreticiler arasında enerji alışverişi; talep yönetimi.

Simülasyon
Trafik, salgın ve ekonomi modellerinde bireylerin ajan olarak canlandırılması.

Ortak soru: kendi hedefi olan ajanlar bir araya geldiğinde **sistem düzeyinde** iyi bir sonuç nasıl elde edilir?

İşbirliği, Rekabet ve Karma Sistemler

İşbirlikçi MAS

Ortak hedef: ajanlar bilgi, görev ve kaynak paylaşır. Sorun *koordinasyon*dur.

`arama-kurtarma drone takımı`

Rekabetçi MAS

Amaçlar farklı: her ajan diğerlerinin davranışını hesaba katar. Sorun *strateji*dir.

`açık artırmada teklifçiler`

Karma (mixed-motive)

Bazı hedefler ortak, bazı kaynaklar rekabet konusu. Gerçek sistemlerin çoğu buradadır.

`aynı şebekeyi paylaşan evler`

Hangisi daha verimli?	Koşul
İşbirliği	Hedefler örtüşüyor, iletişim ucuz, tek ajanın yapamayacağı iş var
Rekabet	Özel bilgi var ve fiyat/teklif ile açığa çıkarılabilir (piyasa, açık artırma)

Yanıt, sistemin amacına ve kaynak yapısına bağlıdır; mekanizma tasarımı rekabeti iyi sonuca yönlendirmeye çalışır.

Çıkar Yelpazesi: Üç Küçük Oyun



- Solda çıkarlar **aynı**: iletişim ya da ortak bir kural (konvansiyon, ör. "herkes sağdan") yeter (koordinasyon sorunu).
- Ortada çıkarlar **kısmen çatışır**: bireysel teşvik ortak iyiye ters düşer, teşvik tasarımı gerekir.
- Sağda biri kazanırken öteki kaybeder (sıfır toplamlı): öngörülemez olmak, yani **karışık strateji** (mixed strategy) önemlidir.

Turuncu hücreler: saf Nash dengeleri (her iki oyuncu da en iyi tepkide).

Bireysel Rasyonellik ve Ortak Kötü Sonuç

Mahkûmun İkilemi

Ortadaki oyunda İhanet baskın stratejidir; denge (−2, −2), oysa (Sessiz, Sessiz) ikisine de (−1, −1) verirdi. Bireysel rasyonellik **ortak kötü sonuç** doğurur.

Bant genişliği

Her ajan kendi akışını artırınca ağ herkes için yavaşlar.

Ortak kaynak

Paylaşılan işlemci, enerji ya da veri havuzu aşırı kullanılır.

Bilgi paylaşımı

Paylaşmak ortak modeli iyileştirir, saklamak ajana kısa vadede avantaj sağlar.

Çok oyunculu sürümü **kamu malı oyunudur**; ortak kaynağın tükenmesi ise ortak mallar trajedisidir.

Karar Alma: Merkezî mi, Dağıtık mı?

Ajanlar çevresel bilgiye göre plan yapar. Kararı kim verir: tek bir merkez mi, her ajan kendisi mi?

	Merkezî kontrol	**Dağıtık (tam otonomi)**
Avantaj	Tüm bilgi tek noktada: **optimum** hesaplanabilir; kolay koordinasyon, düşük çatışma riski	Esnek, ölçeklenebilir, yerel olaylara hızlı tepki; bir parça bozulunca sistem çalışmaya devam eder
Dezavantaj	Tek nokta hatası ve darboğaz; ajan sayısı artınca hesaplama patlar; ajanlar özel bilgisini dürüst bildirmeyebilir	Yerel bilgiyle alınan kararlar uyumsuz olabilir; bencil ajanlar verimsiz dengeye gidebilir (Anarşinin Bedeli)
Örnek	Trafik ışığı yönetim merkezi; sensör ağında merkezî sunucu	Drone takımları, otonom araçlar, robotik keşif

Çoğu gerçek MAS **hibrittir**: ajanlar otonom karar verir, merkez kuralları, fiyatları ya da ihaleleri belirler. Sözleşme ağı buna bir örnektir.

Dağıtık Koordinasyon ve Konvansiyonlar

- Ajanlar merkez olmadan **yerel bilgi ve mesajlaşma** ile karar verir.
- Protokol; iletişim gecikmesi, hata toleransı ve ölçeklenebilirliği birlikte ele almalıdır.
- Ortak bilgi eksikliği uyumsuz planlara yol açar: iki robot dar koridorda karşılaşınca ikisi de aynı tarafa kaçabilir.
- Çözüm: **konvansiyon / sosyal yasa** ("herkes sağdan") — koordinasyon oyununda bir dengeyi önceden seçer.

R1 ╲ R2	Sağ	Sol
Sağ	(1, 1)	(−1, −1)
Sol	(−1, −1)	(1, 1)

Koridor oyunu: iki saf Nash dengesi (turuncu)

İki denge de eşit iyidir; sorun hangisinin oynanacağıdır. Trafik ışığı herkese ortak bir sinyal verir ve iki dengeden hangisinin oynanacağını seçer (**korelasyonlu denge**).

MAS'te Oyun Kuramı ve Teşvikler

Analiz: Nash dengesi

Ajanların stratejik kararları dengede nereye varır? Denge verimli mi?

Tasarım: mekanizma

Kurallar, ödemeler ve açık artırmalar ajanları dürüst bildirime ve işbirliğine yönlendirir.

Enerji paylaşımı

Dengeli teşvik (ör. fazla üretime adil ödeme) → herkes yeterli enerji alır; yanlış teşvik → herkes pik saatte çeker.

Dağıtık ağ kaynakları

Bant genişliği ve hesaplama gücü: kullanım ücreti ya da kota olmazsa aşırı kullanım.

Doğru teşvik

+ İşbirliğini artırır, çatışmayı azaltır.

Yanlış teşvik

– Bencil davranış, aşırı risk ve beleşçilik doğurur.

İletişim, Öğrenme ve Ortaya Çıkan Davranış

İletişim ve bilgi

Ajanların bilgisi farklıdır; kimin neyi bildiği (bilgi yapısı) dengeyi değiştirir (eksik bilgili oyunlar). Bunun için ortak bir dil ve mesajlaşma protokolü gerekir.

Öğrenme ve uyum

Ajanlar geri bildirimle stratejisini günceller: en iyi tepki dinamiği, kurgusal oyun (her ajan rakibinin geçmiş hamlelerinin ortalamasına en iyi tepki verir), pekiştirmeli öğrenme. Diğer ajanlar da öğrendiği için her ajanın çevresi *durağan olmayan* bir çevredir; yakınsama garanti değildir.

Ortaya çıkan davranış

Basit yerel kurallar sistem düzeyinde beklenmedik örüntüler doğurur: Schelling'in (1971) modelinde hafif bir komşu tercihi belirgin mahalle ayrışmasını üretir.

Sistemin davranışı tek tek ajanlara bakarak okunamaz: denge analizi ile simülasyon birlikte kullanılır. Trafik sıkışıklığı ve fiyat dalgalanmaları da ajanlar arası etkileşimden doğan örüntülerdir.

Görev ve Kaynak Tahsisi (Task Allocation)

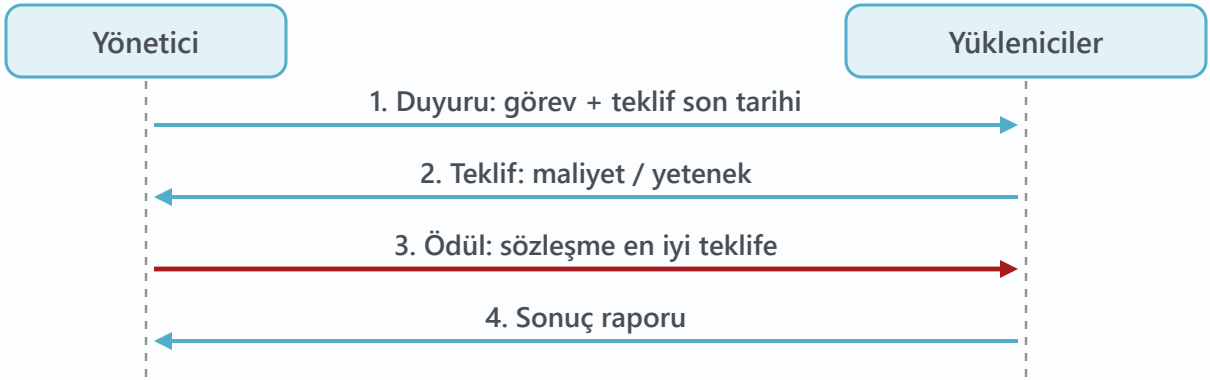
- Amaç: yetenek ve maliyetleri dikkate alarak işleri **uygun ajanlara** dağıtmak (ör. toplam süreyi en aza indirmek).
- Zorluk: maliyetleri çoğu zaman yalnızca ajanın kendisi bilir (**özel bilgi**); ajan sayısı büyüktür; görevler zamanla gelir.
- İyi bir yöntem çözüm kalitesini iletişim ve hesaplama maliyetiyle dengeler.

Yöntem	Nasıl çalışır?	Artı / eksi
Merkezî optimizasyon	Tüm maliyetler toplanır, atama problemi çözülür (Macar algoritması: n görevi n ajana en ucuz biçimde atayan, yaklaşık n^3 adımlı klasik yöntem)	Optimum / tüm bilgi gerekir
Sözleşme ağı	Görev duyurulur, ajanlar teklif verir, en iyi teklif kazanır	Basit, dağıtık / açgözlü olabilir
Açık artırma	Görevler/kaynaklar teklifle satılır; kombinatoryal açık artırmada paketler	Teşvik uyumlu tasarlanabilir
Pazar tabanlı	Fiyatlar arz–talebe göre ayarlanır, ajanlar fiyata tepki verir	Ölçeklenir / yakınsama yavaş olabilir

GÖREV TAHSİSİ

Sözleşme Ağı Protokolü (Contract Net Protocol)

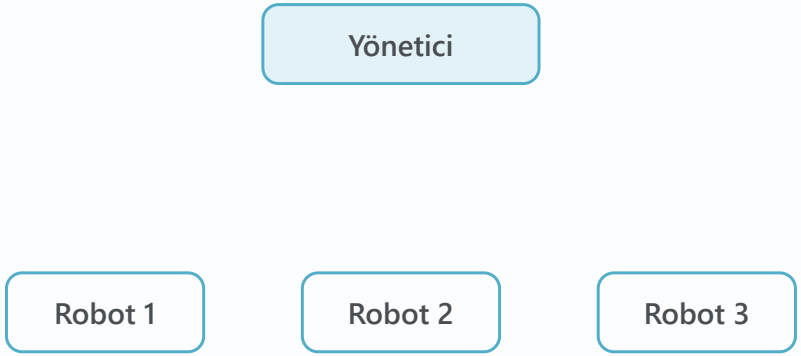
Smith (1980): görevi olan ajan **yönetici** (manager), işi yapabilecek ajanlar **yüklenici** (contractor) rolündedir. Roller görev başına değişebilir.



Yönetici maliyetleri bilmez; teklifler bu **özel bilgiyi** açığa çıkarır. Teklif verme ters bir açık artırmadır (en düşük teklif kazanır). Kazanana kendi teklifi ödenirse yüklenici maliyetini şişirmek ister; ikinci en düşük teklif ödenirse (Vickrey) dürüst teklif en iyisidir.

GÖREV TAHSİSİ

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

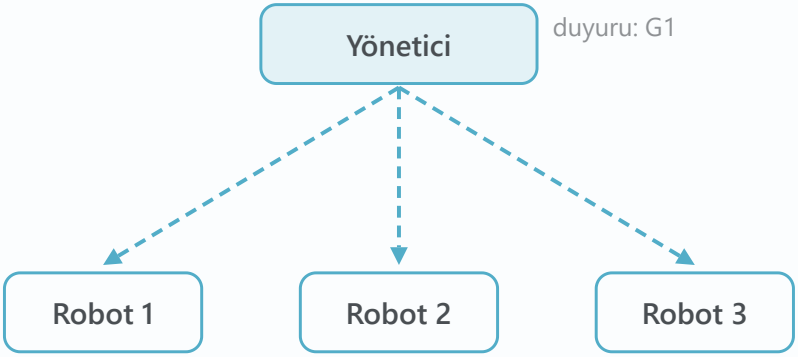
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

Yönetici üç görevi (G1–G3) sırayla ihaleye çıkaracak. Her robot en çok bir görev alabilir; kendi maliyetini (tahmini süre) **yalnızca robot bilir** — yönetici tabloyu görmez.

Adım 1 / 11

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

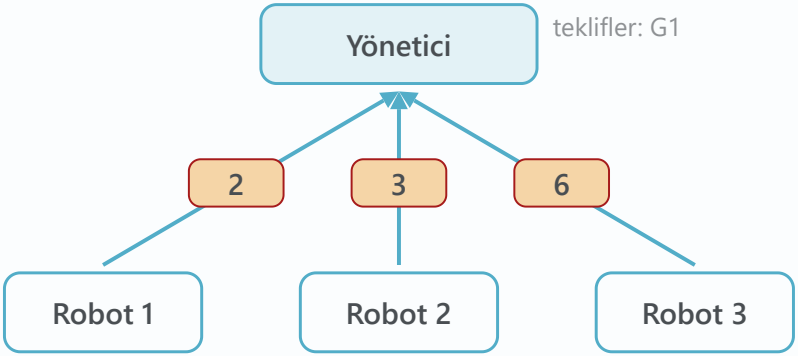
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 0 dk

Duyuru (task announcement): yönetici G1 görevini boştaki robotlara yayınlar: Robot 1, Robot 2, Robot 3.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

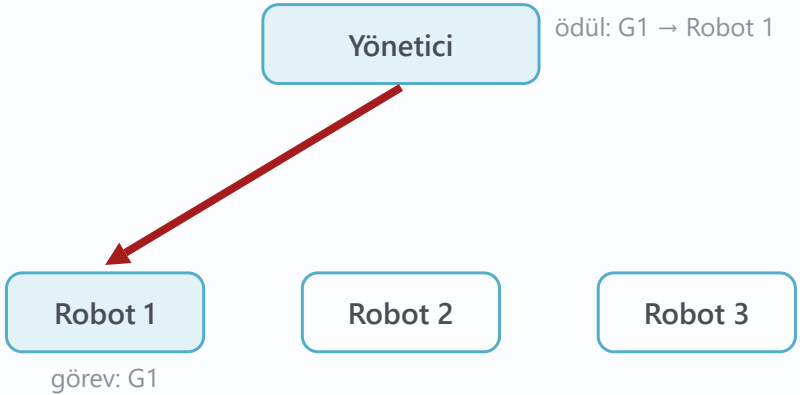
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 0 dk

Teklifler (bid): her boş robot G1 için kendi maliyetini bildirir: R1 → 2, R2 → 3, R3 → 6.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

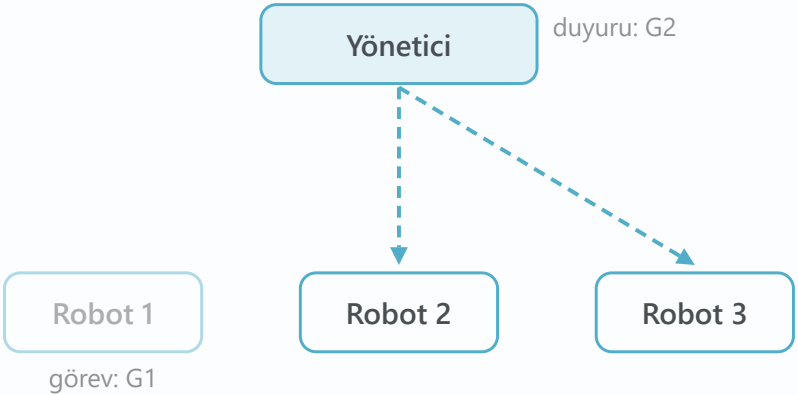
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 2 dk

Ödül (award): en düşük teklif Robot 1 (2 dk) — G1 ona verilir; robot artık meşgul.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

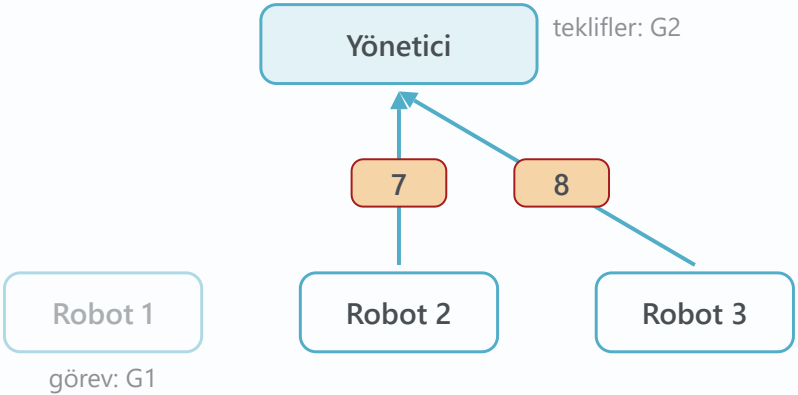
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 2 dk

Duyuru (task announcement): yönetici G2 görevini boştaki robotlara yayınlar: Robot 2, Robot 3.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

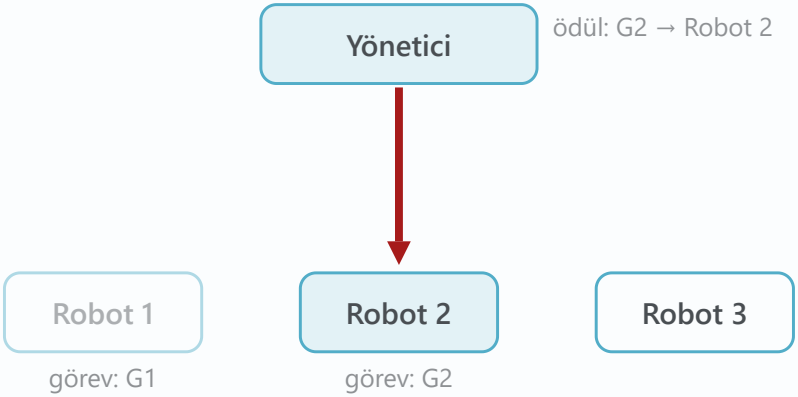
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 2 dk

Teklifler (bid): her boş robot G2 için kendi maliyetini bildirir: R2 → 7, R3 → 8.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

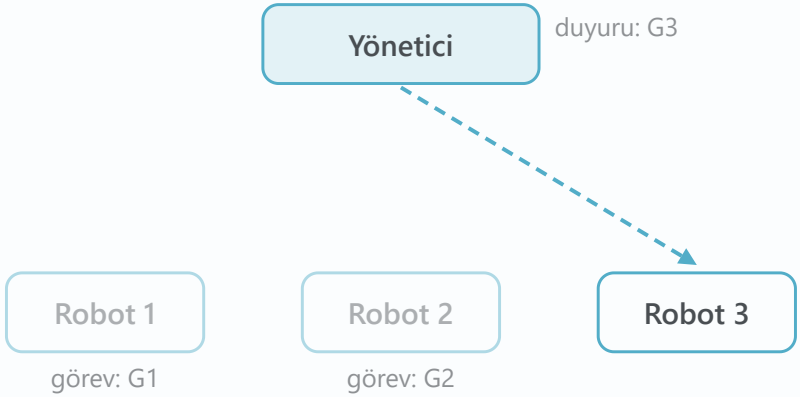
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 9 dk

Ödül (award): en düşük teklif Robot 2 (7 dk) — G2 ona verilir; robot artık meşgul.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

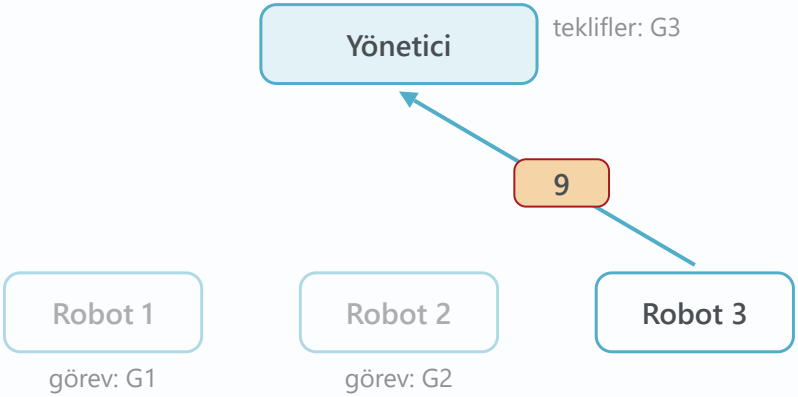
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 9 dk

Duyuru (task announcement): yönetici G3 görevini boştaki robotlara yayınlar: Robot 3.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

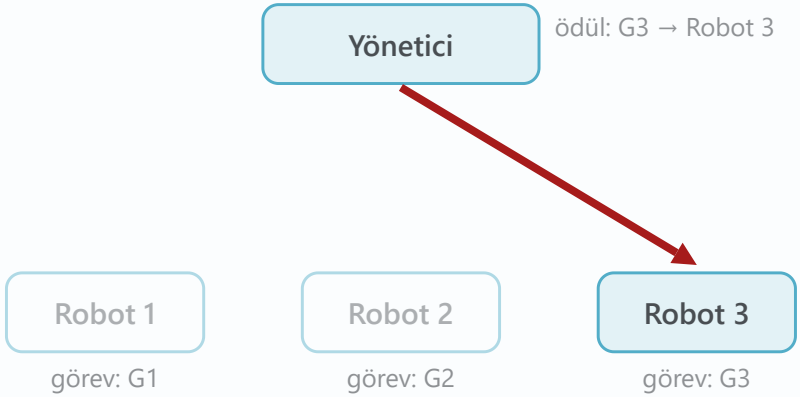
	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

toplam: 9 dk

Teklifler (bid): her boş robot G3 için kendi maliyetini bildirir: R3 → 9.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

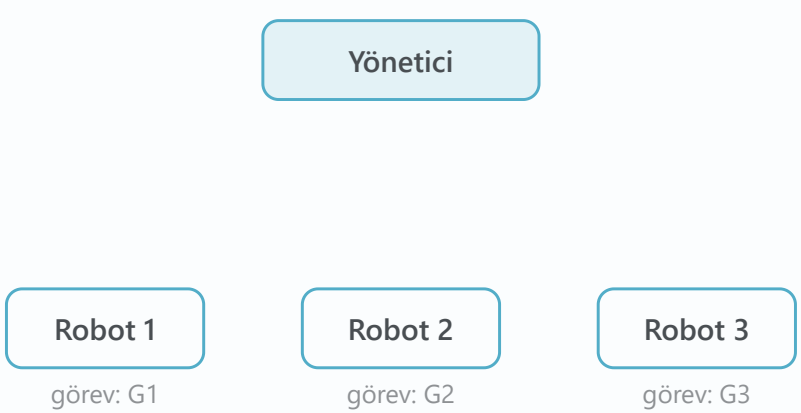
toplam: 18 dk

Ödül (award): en düşük teklif Robot 3 (9 dk) — G3 ona verilir; robot artık meşgul. Tek teklif kaldığı için seçim zorunlu.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

GÖREV TAHSİSİ

Sözleşme Ağı Adım Adım: 3 Görev, 3 Robot



maliyet (dk) = teklif

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

sözleşme ağı: 18 dk

optimum: 13 dk

Sonuç: sözleşme ağı toplam **18 dk** (kırmızı). Tüm 6 atamayı deneyen optimum **13 dk** (kesikli mavi): G1→R3, G2→R1, G3→R2. Açgözlü sıralı ihale %38 daha pahalı.

Gösterim: kesikli ok = duyuru · turuncu = teklif · kalın kırmızı ok / çerçeve = ödül · soluk robot = meşgul · kesikli mavi çerçeve = optimum atama.

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → ?

G2 → ?

G3 → ?

toplam = 0 dk

gorevler = [G1, G2, G3] sırayla ihaleye çıkar. Tüm robotlar boş; atama henüz yok.

Adım 1 / 11

```
▶ def sozlesme_agi(gorevler, maliyet):
▶     bos = set(maliyet)
▶     atama, toplam = {}, 0
4     for g in gorevler:
5         teklif = {r: maliyet[r][g] for r in bos}
6         kazanan = min(teklif, key=teklif.get)
7         atama[g] = kazanan
8         toplam += teklif[kazanan]
9         bos.remove(kazanan)
10    return atama, toplam
```

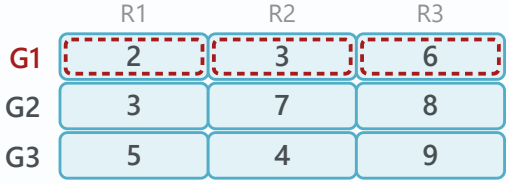
DEĞİŞKENLER

`bos = {R1, R2, R3}`

`atama = {}` `toplam = 0`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`



maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → ?

G2 → ?

G3 → ?

toplam = 0 dk

Duyuru: G1. Boştaki robotlar teklif verir: R1 → 2, R2 → 3, R3 → 6.

Adım 2 / 11

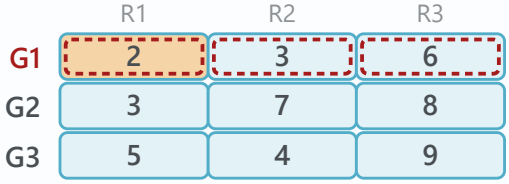
```
1  def sozlesme_agi(gorevler, maliyet):
2      bos = set(maliyet)
3      atama, toplam = {}, 0
▶      for g in gorevler:
▶          teklif = {r: maliyet[r][g] for r in bos}
6          kazanan = min(teklif, key=teklif.get)
7          atama[g] = kazanan
8          toplam += teklif[kazanan]
9          bos.remove(kazanan)
10     return atama, toplam
```

DEĞİŞKENLER

`g = G1` `bos = {R1, R2, R3}` `teklif = {R1: 2, R2: 3, R3: 6}` `toplam = 0`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`



maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → ?

G2 → ?

G3 → ?

toplam = 0 dk

Ödül: en düşük teklif **R1** (2 dk).

Adım 3 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
4    for g in gorevler:
5      teklif = {r: maliyet[r][g] for r in bos}
▶      kazanan = min(teklif, key=teklif.get)
7      atama[g] = kazanan
8      toplam += teklif[kazanan]
9      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G1`

`teklif = {R1: 2, R2: 3, R3: 6}`

`kazanan = R1` `toplam = 0`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`

	R1 (meşgul)	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → ?

G3 → ?

toplam = 2 dk

atama[G1] = R1, toplam = 2; R1 artık meşgul.

Adım 4 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
4    for g in gorevler:
5      teklif = {r: maliyet[r][g] for r in bos}
6      kazanan = min(teklif, key=teklif.get)
▶      atama[g] = kazanan
▶      toplam += teklif[kazanan]
▶      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G1` `kazanan = R1`

`atama = {G1: R1}` `toplam = 2`

`bos = {R2, R3}`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`

	R1 (meşgul)	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → ?

G3 → ?

toplam = 2 dk

Duyuru: G2. Boştaki robotlar teklif verir: R2 → 7, R3 → 8.

Adım 5 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
▶    for g in gorevler:
▶      teklif = {r: maliyet[r][g] for r in bos}
6      kazanan = min(teklif, key=teklif.get)
7      atama[g] = kazanan
8      toplam += teklif[kazanan]
9      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G2` `bos = {R2, R3}` `teklif = {R2: 7, R3: 8}` `toplam = 2`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`



maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → ?

G3 → ?

toplam = 2 dk

Ödül: en düşük teklif **R2** (7 dk).

Adım 6 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
4    for g in gorevler:
5      teklif = {r: maliyet[r][g] for r in bos}
▶      kazanan = min(teklif, key=teklif.get)
7      atama[g] = kazanan
8      toplam += teklif[kazanan]
9      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G2`

`teklif = {R2: 7, R3: 8}`

`kazanan = R2` `toplam = 2`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`

	R1 (meşgul)	R2 (meşgul)	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → R2 (7 dk)

G3 → ?

toplam = 9 dk

atama[G2] = R2, toplam = 9; R2 artık meşgul. G1'i ucuza alan R1, G2 için de en ucuzdu (3) ama artık yok.

Adım 7 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
4    for g in gorevler:
5      teklif = {r: maliyet[r][g] for r in bos}
6      kazanan = min(teklif, key=teklif.get)
▶      atama[g] = kazanan
▶      toplam += teklif[kazanan]
▶      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G2` `kazanan = R2`

`atama = {G1: R1, G2: R2}`

`toplam = 9` `bos = {R3}`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`



atama

G1 → R1 (2 dk)

G2 → R2 (7 dk)

G3 → ?

toplam = 9 dk

Duyuru: G3. Boştaki robotlar teklif verir: R3 → 9.

Adım 8 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
▶    for g in gorevler:
▶      teklif = {r: maliyet[r][g] for r in bos}
6      kazanan = min(teklif, key=teklif.get)
7      atama[g] = kazanan
8      toplam += teklif[kazanan]
9      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G3` `bos = {R3}`

`teklif = {R3: 9}` `toplam = 9`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`



maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → R2 (7 dk)

G3 → ?

toplam = 9 dk

Ödül: en düşük teklif **R3** (9 dk). Tek teklif: seçim zorunlu.

Adım 9 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
4    for g in gorevler:
5      teklif = {r: maliyet[r][g] for r in bos}
▶      kazanan = min(teklif, key=teklif.get)
7      atama[g] = kazanan
8      toplam += teklif[kazanan]
9      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G3` `teklif = {R3: 9}` `kazanan = R3` `toplam = 9`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`

	R1 (meşgul)	R2 (meşgul)	R3 (meşgul)
G1	2	3	6
G2	3	7	8
G3	5	4	9

maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → R2 (7 dk)

G3 → R3 (9 dk)

toplam = 18 dk

atama[G3] = R3, toplam = 18; R3 artık meşgul.

Adım 10 / 11

```
1  def sozlesme_agi(gorevler, maliyet):
2    bos = set(maliyet)
3    atama, toplam = {}, 0
4    for g in gorevler:
5      teklif = {r: maliyet[r][g] for r in bos}
6      kazanan = min(teklif, key=teklif.get)
▶      atama[g] = kazanan
▶      toplam += teklif[kazanan]
▶      bos.remove(kazanan)
10   return atama, toplam
```

DEĞİŞKENLER

`g = G3` `kazanan = R3`

`atama = {G1: R1, G2: R2, G3: R3}`

`toplam = 18` `bos = set()`

GÖREV TAHSİSİ

Kod İzleme: `sozlesme_agi(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

maliyet[r][g] (dk) — yalnız robot bilir

atama

G1 → R1 (2 dk)

G2 → R2 (7 dk)

G3 → R3 (9 dk)

toplam = 18 dk

Sonuç 18 dk: her adımda yerel en iyi karar, ama toplam optimum değil.

Adım 11 / 11

```
1 def sozlesme_agi(gorevler, maliyet):
2   bos = set(maliyet)
3   atama, toplam = {}, 0
4   for g in gorevler:
5     teklif = {r: maliyet[r][g] for r in bos}
6     kazanan = min(teklif, key=teklif.get)
7     atama[g] = kazanan
8     toplam += teklif[kazanan]
9     bos.remove(kazanan)
▶   return atama, toplam
```

DEĞİŞKENLER

`atama = {G1: R1, G2: R2, G3: R3}`

`toplam = 18`

GÖREV TAHSİSİ

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) · (R2, R3, R1) ·
(R1, R3, R2) · (R3, R1, R2) ·
(R2, R1, R3) · (R3, R2, R1) ·

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

Merkezî çözüm: tüm maliyetleri bilen yönetici, 3 robotun 3! = 6 sıralamasını dener.

Adım 1 / 8

```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
▶   en_iyi = None
4   for p in permutations(maliyet):
5     t = sum(maliyet[r][g]
6             for g, r in zip(gorevler, p))
7     if en_iyi is None or t < en_iyi[0]:
8       en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`en_iyi = None`

GÖREV TAHSİSİ

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) ·
(R1, R3, R2) · (R3, R1, R2) ·
(R2, R1, R3) · (R3, R2, R1) ·

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

p = (R1, R2, R3): t = 2 + 7 + 9 = **18** ⇒ yeni en iyi.

Adım 2 / 8

```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
    for p in permutations(maliyet):
      t = sum(maliyet[r][g]
              for g, r in zip(gorevler, p))
      if en_iyi is None or t < en_iyi[0]:
        en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`p = (R1, R2, R3)` `t = 18`

`en_iyi = (18, (R1, R2, R3))`

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) ·
(R1, R3, R2) 14 (R3, R1, R2) ·
(R2, R1, R3) · (R3, R2, R1) ·

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

p = (R1, R3, R2): t = 2 + 8 + 4 = **14** ⇒ yeni en iyi.

Adım 3 / 8

```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
▶   for p in permutations(maliyet):
▶     t = sum(maliyet[r][g]
▶             for g, r in zip(gorevler, p))
▶     if en_iyi is None or t < en_iyi[0]:
▶       en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`p = (R1, R3, R2)` `t = 14`

`en_iyi = (14, (R1, R3, R2))`

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) ·
(R1, R3, R2) 14 (R3, R1, R2) ·
(R2, R1, R3) 15 (R3, R2, R1) ·

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

p = (R2, R1, R3): t = 3 + 3 + 9 = 15 ≥ 14: değişmez.


```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
▶   for p in permutations(maliyet):
▶     t = sum(maliyet[r][g]
▶             for g, r in zip(gorevler, p))
▶     if en_iyi is None or t < en_iyi[0]:
8       en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`p = (R2, R1, R3)` `t = 15`

`en_iyi = (14, (R1, R3, R2))`

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) 16
(R1, R3, R2) 14 (R3, R1, R2) ·
(R2, R1, R3) 15 (R3, R2, R1) ·

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

p = (R2, R3, R1): t = 3 + 8 + 5 = 16 ≥ 14: değişmez.

Adım 5 / 8

```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
▶   for p in permutations(maliyet):
▶     t = sum(maliyet[r][g]
▶             for g, r in zip(gorevler, p))
▶     if en_iyi is None or t < en_iyi[0]:
8       en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`p = (R2, R3, R1)` `t = 16`

`en_iyi = (14, (R1, R3, R2))`

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) 16
(R1, R3, R2) 14 (R3, R1, R2) 13
(R2, R1, R3) 15 (R3, R2, R1) ·

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

p = (R3, R1, R2): t = 6 + 3 + 4 = **13** ⇒ yeni en iyi.


```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
▶   for p in permutations(maliyet):
▶     t = sum(maliyet[r][g]
▶             for g, r in zip(gorevler, p))
▶     if en_iyi is None or t < en_iyi[0]:
▶       en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`p = (R3, R1, R2)` `t = 13`

`en_iyi = (13, (R3, R1, R2))`

GÖREV TAHSİSİ

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) 16
(R1, R3, R2) 14 (R3, R1, R2) 13
(R2, R1, R3) 15 (R3, R2, R1) 18

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

p = (R3, R2, R1): t = 6 + 7 + 5 = 18 ≥ 13: değişmez.

Adım 7 / 8

```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
▶   for p in permutations(maliyet):
▶     t = sum(maliyet[r][g]
▶             for g, r in zip(gorevler, p))
▶     if en_iyi is None or t < en_iyi[0]:
8       en_iyi = (t, p)
9   return en_iyi
```

DEĞİŞKENLER

`p = (R3, R2, R1)` `t = 18`

`en_iyi = (13, (R3, R1, R2))`

GÖREV TAHSİSİ

Kod İzleme: `optimum(gorevler, maliyet)`

	R1	R2	R3
G1	2	3	6
G2	3	7	8
G3	5	4	9

(R1, R2, R3) 18 (R2, R3, R1) 16
(R1, R3, R2) 14 (R3, R1, R2) 13
(R2, R1, R3) 15 (R3, R2, R1) 18

p = (G1, G2, G3) görevlerini alan robotlar; turuncu = en iyi

Optimum 13 dk: G1→R3, G2→R1, G3→R2; sözleşme ağı 18 dk (%38 fazla). Ama n robot için n! atama: n = 10'da 3.628.800 (Macar algoritması $O(n^3)$ ile çözer).

Adım 8 / 8

```
1 from itertools import permutations
2 def optimum(gorevler, maliyet):
3   en_iyi = None
4   for p in permutations(maliyet):
5     t = sum(maliyet[r][g]
6             for g, r in zip(gorevler, p))
7     if en_iyi is None or t < en_iyi[0]:
8       en_iyi = (t, p)
▶   return en_iyi
```

DEĞİŞKENLER

`en_iyi = (13, (R3, R1, R2))`

GÖREV TAHSİSİ

Sözleşme Ağı: Güçlü ve Zayıf Yanlar

Güçlü yanlar

+ Dağıtık: yönetici maliyet tablosunu bilmeden iyi bir atama yapar.
+ Görev başına mesaj sayısı O(n): duyuru + n teklif + ödül.
+ Yeni robot eklemek kolay; bozulan robotun görevi yeniden ihaleye çıkar.

Zayıf yanlar

– **Açgözlü ve sıralı**: örnekte 18 dk, optimum 13 dk (%38 fazla). G1'i ucuza alan R1, G2 için en iyi aday olduğu hâlde meşgul kaldı.
– Robotlar maliyetini stratejik olarak şişirebilir.
– Görev sırası sonucu değiştirir.

İyileştirmeler: **yeniden ihale** (atamaları takasla düzeltme), **kombinatoryal açık artırma** (robotlar görev paketlerine teklif verir), dürüst teklif için **ikinci fiyat / VCG ödemeleri**.

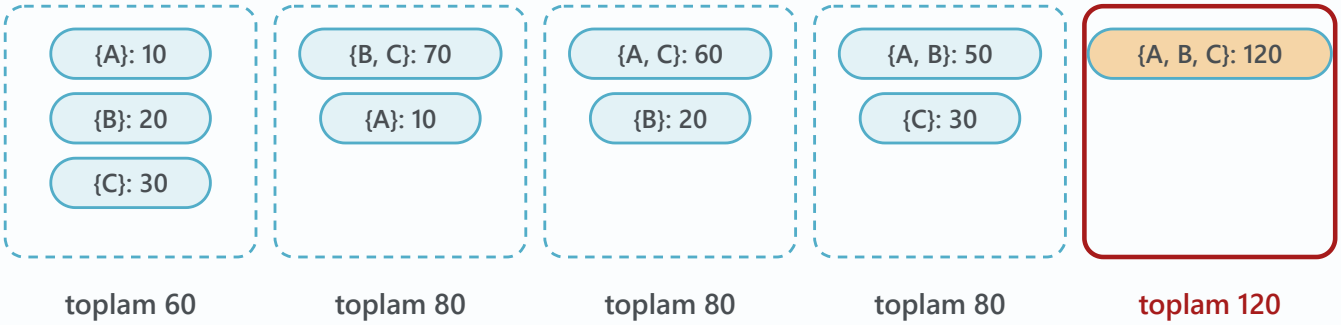
Koalisyon Oluşumu (Coalition Formation)

- Tek başına hedefe ulaşamayan ajanlar geçici veya kalıcı **koalisyonlar** kurabilir: kargo firmaları araç paylaşır, drone'lar ağır yükü birlikte taşır.
- Bu oyunlar **işbirlikçidir**: bağlayıcı anlaşma yapılabilir; soru "hangi strateji?" değil, "kim kiminle birleşir ve kazanç **nasıl paylaşılır**?"
- Paylaşım adil değilse koalisyon dağılır: kararlılık doğrudan paylaşım kuralına bağlıdır.

Shapley değeri
Adalet: herkes ortalama *marjinal katkısı* kadar alır.

Çekirdek (core)
Kararlılık: hiçbir alt grup ayrılarak daha fazlasını alamaz.

Koalisyon Yapıları: Kim Kiminle?



- **Koalisyon yapısı**: ajanların ayrık koalisyonlara bölünmesi. Toplum için soru: toplam değeri en büyük yapı hangisi?
- Yapı sayısı, n ajanı boş olmayan ayrık gruplara bölmenin yol sayısıdır (Bell sayısı): n = 3 için 5 n = 4 için 15 n = 8 için 4.140 n = 10 için 115.975.
- Örnekte oyun **süperadditif**: birleşmek hiç zarar vermez, en iyi yapı büyük koalisyon {A, B, C} (120).
- Genel durumda en iyi yapıyı bulmak NP-zordur (Sandholm vd., 1999); geriye kalan soru **120'nin paylaşımı**.

Koalisyon Oyunu ve Karakteristik Fonksiyon

- Oyuncular kümesi N = {1, …, n}; her **koalisyon** S ⊆ N için **v(S)**: S'nin üyeleri birlikte, diğerlerinden bağımsız olarak elde edebileceği toplam değer.
- v her koalisyona, o grubun birlikte kazanabileceği toplam değeri atar; boş koalisyonun değeri 0'dır. v'ye **karakteristik fonksiyon** denir. Kazanç grup içinde serbestçe paylaştırılabildiği için bu oyunlara aktarılabilir faydalı (TU) oyun denir.
- **Süperadditif**: ayrık S, T için v(S ∪ T) ≥ v(S) + v(T) → birleşmek zarar vermez; büyük koalisyon N oluşur.
- Örnekte üç lojistik firması A, B, C: birlikte 120, en iyi ikili 70, tek başına en fazla 30 kazanır.

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S) — kâr (bin TL)

120 Nasıl Paylaşılmalı?

Bir **paylaşım** (imputation) $x = (x_A, x_B, x_C)$: toplamı $v(N) = 120$ (verimlilik) ve herkes en az tek başına değerini alır (bireysel rasyonellik).

Öneri	x_A	x_B	x_C	Gerekçe / sorun
Eşit bölüşüm	40	40	40	Basit; ama C'nin daha fazla katkısını yok sayar
Tek başına değerle orantılı (10 : 20 : 30)	20	40	60	Yalnızca $v(\{i\})$'ye bakar; sinerjileri görmez
Güçlü C'nin önerisi	10	25	85	A ve B birlikte 50 alabilir; kabul etmezler
Shapley değeri	30	40	50	Ortalama marjinal katkı

İki ölçüt: paylaşım **adil** mi (Shapley) ve **kararlı** mı (çekirdek)?

KOALİSYON OYUNLARI

Shapley Değeri (Shapley Value)

- Oyuncular **rastgele bir sırayla** gelip koalisyona katılır. Her oyuncuya, geldiği anda kattığı değer (marjinal katkı) verilir.
- Shapley değeri: n! sıranın hepsi üzerinden **ortalama marjinal katkı**.
- Hesap maliyeti: n! sıra (ya da 2^{n-1} alt küme); büyük n için örnekleme ile yaklaşık hesaplanır.
- Shapley (1953): yandaki dört aksiyomu sağlayan **tek** kural. Model açıklamada da kullanılır (SHAP: bir modelin kararını özelliklerin katkılarına bölen yöntem).

Verimlilik

$\Sigma \varphi_i = v(N)$.

Simetri

Aynı katkıyı yapan iki oyuncu aynı payı alır.

Kukla oyuncu

Hiçbir koalisyona katkısı yoksa $\varphi_i = 0$.

Toplamsallık

İki oyunun değerleri toplanırsa Shapley payları da toplanır.

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	·	·	·
A → C → B	·	·	·
B → A → C	·	·	·
B → C → A	·	·	·
C → A → B	·	·	·
C → B → A	·	·	·
toplam	·	·	·
φ = toplam / 6	·	·	·

Her sırada ajanlar tek tek gelir; her birine **geldiği anda koalisyona kattığı değer** (marjinal katkı) yazılır.

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	·	·	·
B → A → C	·	·	·
B → C → A	·	·	·
C → A → B	·	·	·
C → B → A	·	·	·
toplam	·	·	·
φ = toplam / 6	·	·	·

Sıra A → B → C — A: 10 − 0 = 10; B: 50 − 10 = 40; C: 120 − 50 = 70. Satır toplamı 120 = v(ABC).

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	10	60	50
B → A → C	·	·	·
B → C → A	·	·	·
C → A → B	·	·	·
C → B → A	·	·	·
toplam	·	·	·
φ = toplam / 6	·	·	·

Sıra A → C → B — A: 10 − 0 = 10; C: 60 − 10 = 50; B: 120 − 60 = 60. Satır toplamı 120 = v(ABC).

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	10	60	50
B → A → C	30	20	70
B → C → A	·	·	·
C → A → B	·	·	·
C → B → A	·	·	·
toplam	·	·	·
φ = toplam / 6	·	·	·

Sıra B → A → C — B: 20 − 0 = 20; A: 50 − 20 = 30; C: 120 − 50 = 70. Satır toplamı 120 = v(ABC).

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	10	60	50
B → A → C	30	20	70
B → C → A	50	20	50
C → A → B	·	·	·
C → B → A	·	·	·
toplam	·	·	·
φ = toplam / 6	·	·	·

Sıra B → C → A — B: 20 − 0 = 20; C: 70 − 20 = 50; A: 120 − 70 = 50. Satır toplamı 120 = v(ABC).

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	10	60	50
B → A → C	30	20	70
B → C → A	50	20	50
C → A → B	30	60	30
C → B → A	·	·	·
toplam	·	·	·
φ = toplam / 6	·	·	·

Sıra C → A → B — C: 30 − 0 = 30; A: 60 − 30 = 30; B: 120 − 60 = 60. Satır toplamı 120 = v(ABC).

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	10	60	50
B → A → C	30	20	70
B → C → A	50	20	50
C → A → B	30	60	30
C → B → A	50	40	30
toplam	·	·	·
φ = toplam / 6	·	·	·

Sıra C → B → A — C: 30 − 0 = 30; B: 70 − 30 = 40; A: 120 − 70 = 50. Satır toplamı 120 = v(ABC).

Shapley Değeri Adım Adım: 6 Sıralama

S	v(S)
{A}	10
{B}	20
{C}	30
{A, B}	50
{A, C}	60
{B, C}	70
{A, B, C}	120

v(S)

sıra	A	B	C
A → B → C	10	40	70
A → C → B	10	60	50
B → A → C	30	20	70
B → C → A	50	20	50
C → A → B	30	60	30
C → B → A	50	40	30
toplam	180	240	300
φ = toplam / 6	**30**	**40**	**50**

Ortalama marjinal katkı: **φ = (30, 40, 50)**. Toplam 120: verimlilik aksiyomu sağlanır.

Kod İzleme: `shapley(N, v)` (1/2)

sıra (katılım sırası)

S = ∅

φA 0 | φB 0 | φC 0

phi (toplam katkı, henüz bölünmedi)

phi sözlüğü sıfırlanır; `permutations` (itertools) 3! = 6 sırayı üretir.

Adım 1 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
▶    phi = {i: 0 for i in N}
▶    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

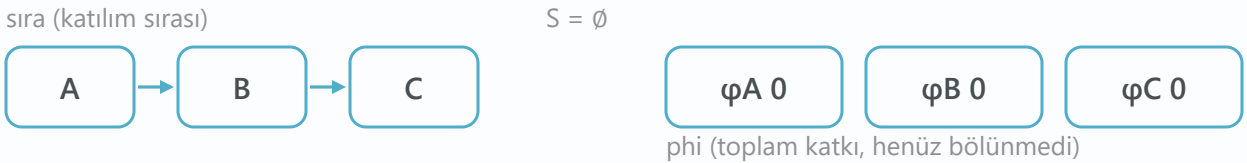
DEĞİŞKENLER

`N = [A, B, C]`

`siralar = 6 sıra` | `phi[A] = 0`

`phi[B] = 0` | `phi[C] = 0`

Kod İzleme: `shapley(N, v)` (1/2)



1. sıra (A, B, C): ajanlar bu sırayla gelir. S boş koalisyonla başlar.

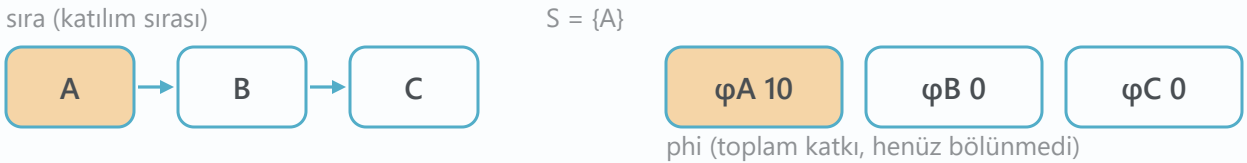
Adım 2 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (A, B, C)` `S = ∅` `phi[A] = 0` `phi[B] = 0` `phi[C] = 0`

Kod İzleme: `shapley(N, v)` (1/2)



A katılır: katkı = v(A) − v(∅) = 10 − 0 = **10**; phi[A] = 10.

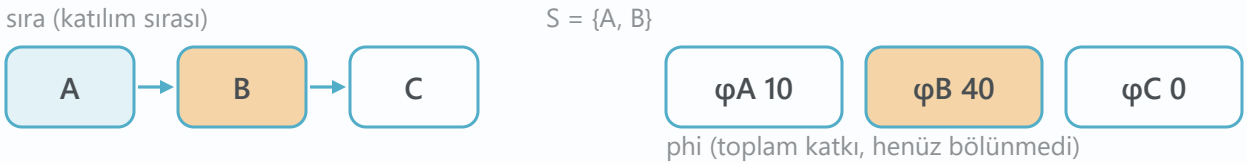
Adım 3 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (A, B, C)` `S = {A}` `i = A` `katki = 10` `phi[A] = 10` `phi[B] = 0` `phi[C] = 0`

Kod İzleme: `shapley(N, v)` (1/2)



B katılır: katkı = v(AB) − v(A) = 50 − 10 = **40**; phi[B] = 40.

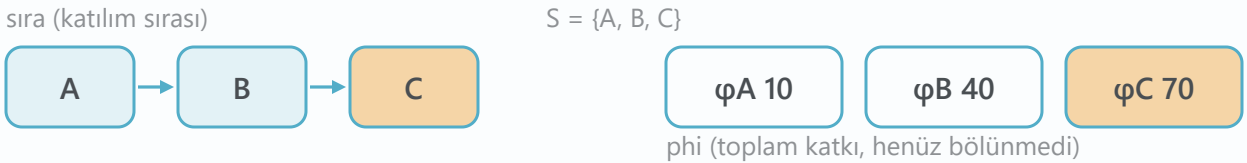
Adım 4 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (A, B, C)` `S = {A, B}` `i = B` `katki = 40` `phi[A] = 10` `phi[B] = 40` `phi[C] = 0`

Kod İzleme: `shapley(N, v)` (1/2)



C katılır: katkı = v(ABC) − v(AB) = 120 − 50 = **70**; phi[C] = 70.

Adım 5 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

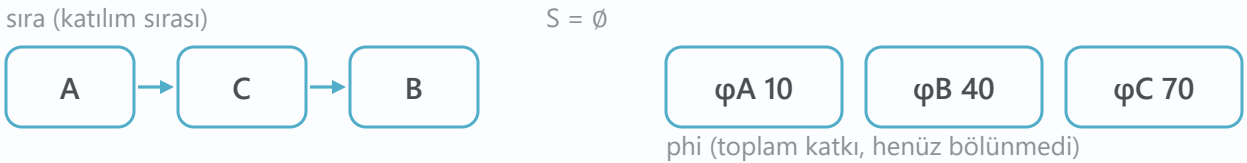
`sira = (A, B, C)`

`S = {A, B, C}` `i = C`

`katki = 70` `phi[A] = 10`

`phi[B] = 40` `phi[C] = 70`

Kod İzleme: `shapley(N, v)` (1/2)



2. sıra (A, C, B): ajanlar bu sırayla gelir. S boş koalisyonla başlar.

Adım 6 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

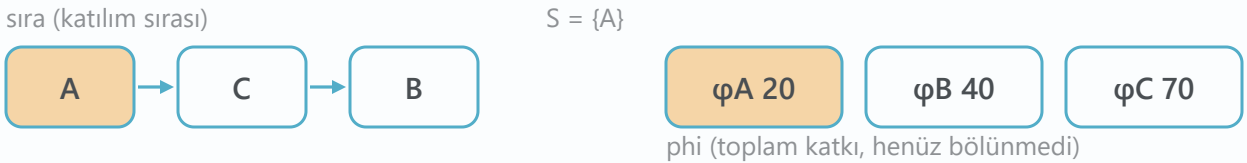
DEĞİŞKENLER

`sira = (A, C, B)` `S = ∅`

`phi[A] = 10` `phi[B] = 40`

`phi[C] = 70`

Kod İzleme: `shapley(N, v)` (1/2)



A katılır: katkı = v(A) − v(∅) = 10 − 0 = **10**; phi[A] = 20.

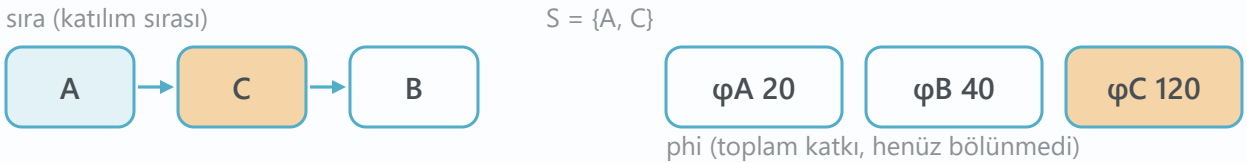
Adım 7 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (A, C, B)` `S = {A}` `i = A` `katki = 10` `phi[A] = 20` `phi[B] = 40` `phi[C] = 70`

Kod İzleme: `shapley(N, v)` (1/2)



C katılır: katkı = v(AC) − v(A) = 60 − 10 = **50**; phi[C] = 120.

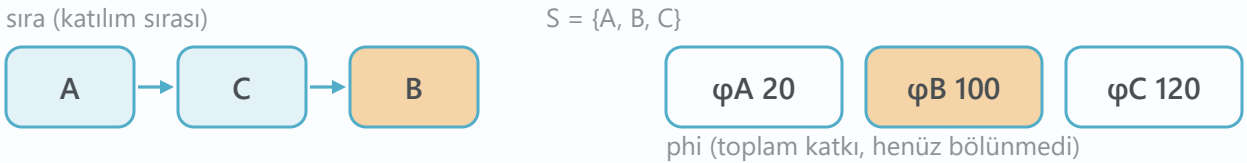
Adım 8 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (A, C, B)` `S = {A, C}` `i = C` `katki = 50` `phi[A] = 20` `phi[B] = 40` `phi[C] = 120`

Kod İzleme: `shapley(N, v)` (1/2)



B katılır: katkı = v(ABC) − v(AC) = 120 − 60 = **60**; phi[B] = 100.


```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

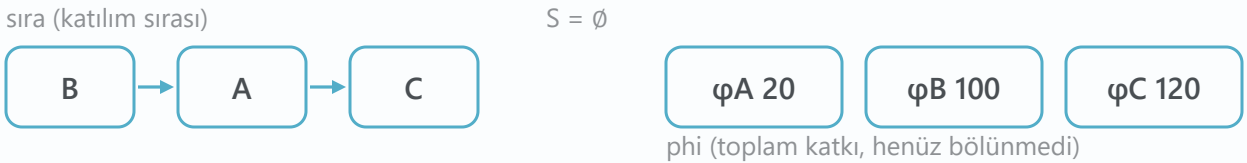
`sira = (A, C, B)`

`S = {A, B, C}` `i = B`

`katki = 60` `phi[A] = 20`

`phi[B] = 100` `phi[C] = 120`

Kod İzleme: `shapley(N, v)` (1/2)



3. sıra (B, A, C): ajanlar bu sırayla gelir. S boş koalisyonla başlar.

Adım 10 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

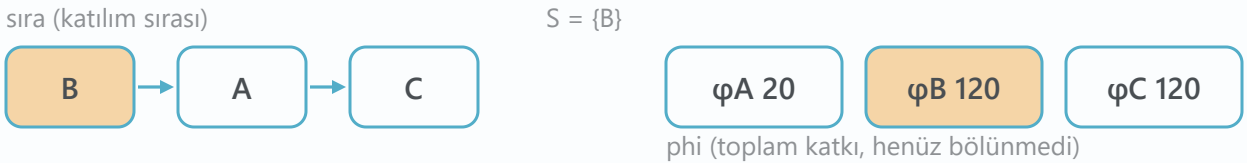
DEĞİŞKENLER

`sira = (B, A, C)` `S = ∅`

`phi[A] = 20` `phi[B] = 100`

`phi[C] = 120`

Kod İzleme: `shapley(N, v)` (1/2)



B katılır: katkı = v(B) − v(∅) = 20 − 0 = **20**; phi[B] = 120.

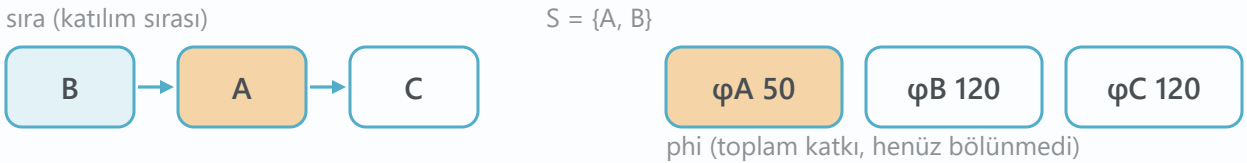
Adım 11 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (B, A, C)` `S = {B}` `i = B` `katki = 20` `phi[A] = 20` `phi[B] = 120` `phi[C] = 120`

Kod İzleme: `shapley(N, v)` (1/2)



A katılır: katkı = v(AB) − v(B) = 50 − 20 = **30**; phi[A] = 50.

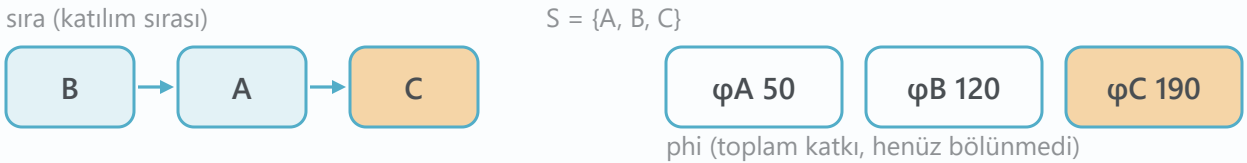
Adım 12 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (B, A, C)` `S = {A, B}` `i = A` `katki = 30` `phi[A] = 50` `phi[B] = 120` `phi[C] = 120`

Kod İzleme: `shapley(N, v)` (1/2)



C katılır: katkı = v(ABC) − v(AB) = 120 − 50 = **70**; phi[C] = 190.

Adım 13 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

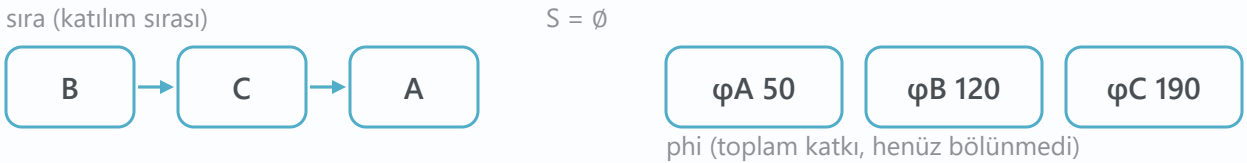
`sira = (B, A, C)`

`S = {A, B, C}` `i = C`

`katki = 70` `phi[A] = 50`

`phi[B] = 120` `phi[C] = 190`

Kod İzleme: `shapley(N, v)` (2/2)



4. sıra (B, C, A): ajanlar bu sırayla gelir. S boş koalisyonla başlar.

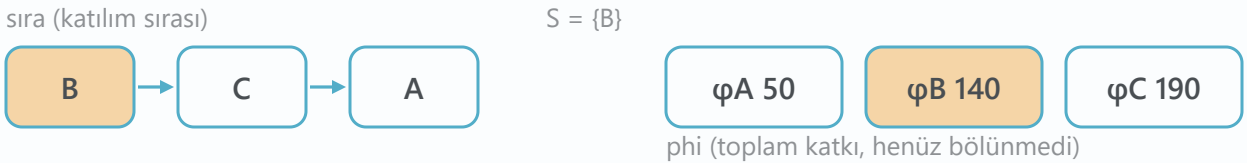
Adım 1 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (B, C, A)` `S = ∅` `phi[A] = 50` `phi[B] = 120` `phi[C] = 190`

Kod İzleme: `shapley(N, v)` (2/2)



B katılır: katkı = v(B) − v(∅) = 20 − 0 = **20**; phi[B] = 140.

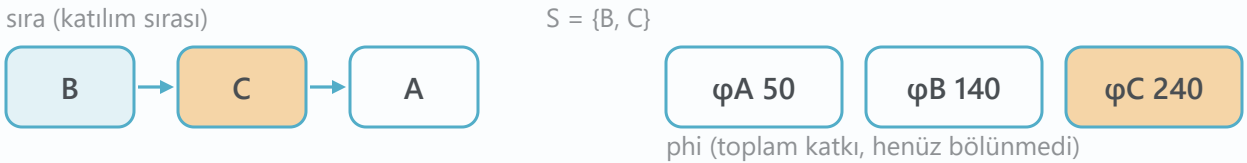
Adım 2 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (B, C, A)` `S = {B}` `i = B` `katki = 20` `phi[A] = 50` `phi[B] = 140` `phi[C] = 190`

Kod İzleme: `shapley(N, v)` (2/2)



C katılır: katkı = v(BC) − v(B) = 70 − 20 = **50**; phi[C] = 240.

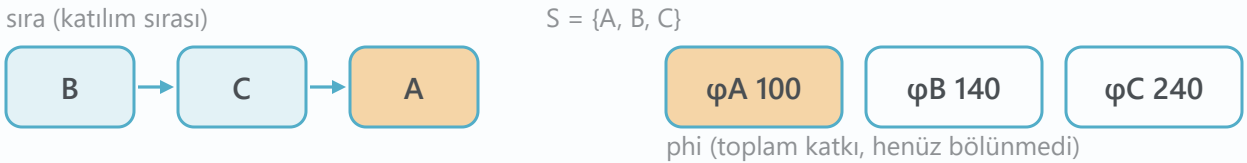
Adım 3 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (B, C, A)` `S = {B, C}` `i = C` `katki = 50` `phi[A] = 50` `phi[B] = 140` `phi[C] = 240`

Kod İzleme: `shapley(N, v)` (2/2)



A katılır: katkı = v(ABC) − v(BC) = 120 − 70 = **50**; phi[A] = 100.

Adım 4 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

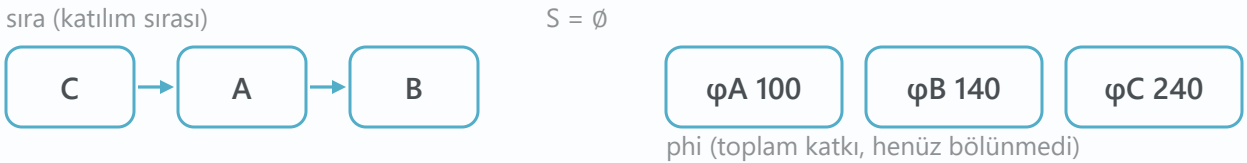
`sira = (B, C, A)`

`S = {A, B, C}` `i = A`

`katki = 50` `phi[A] = 100`

`phi[B] = 140` `phi[C] = 240`

Kod İzleme: `shapley(N, v)` (2/2)



5. sıra (C, A, B): ajanlar bu sırayla gelir. S boş koalisyonla başlar.

Adım 5 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

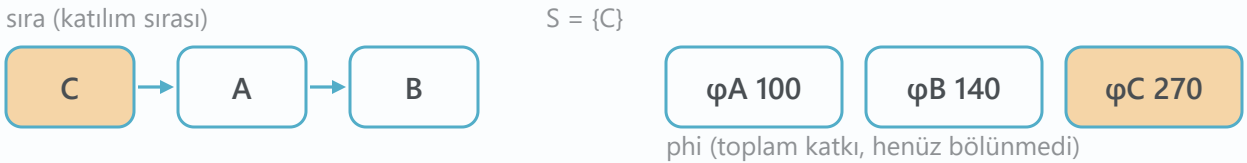
DEĞİŞKENLER

`sira = (C, A, B)` `S = ∅`

`phi[A] = 100` `phi[B] = 140`

`phi[C] = 240`

Kod İzleme: `shapley(N, v)` (2/2)



C katılır: katkı = v(C) − v(∅) = 30 − 0 = **30**; phi[C] = 270.

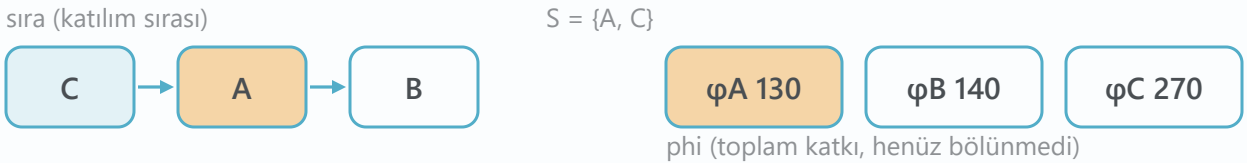
Adım 6 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (C, A, B)` `S = {C}` `i = C` `katki = 30` `phi[A] = 100` `phi[B] = 140` `phi[C] = 270`

Kod İzleme: `shapley(N, v)` (2/2)



A katılır: katkı = v(AC) − v(C) = 60 − 30 = **30**; phi[A] = 130.

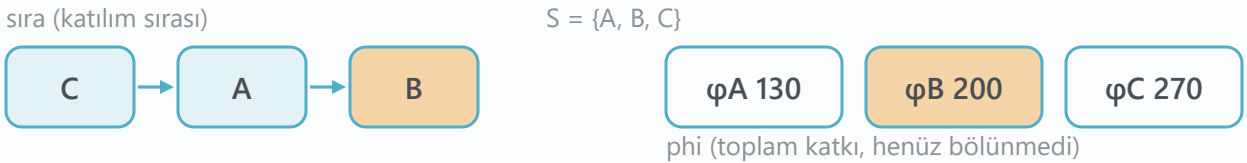
Adım 7 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (C, A, B)` `S = {A, C}` `i = A` `katki = 30` `phi[A] = 130` `phi[B] = 140` `phi[C] = 270`

Kod İzleme: `shapley(N, v)` (2/2)



B katılır: katkı = v(ABC) − v(AC) = 120 − 60 = **60**; phi[B] = 200.

Adım 8 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

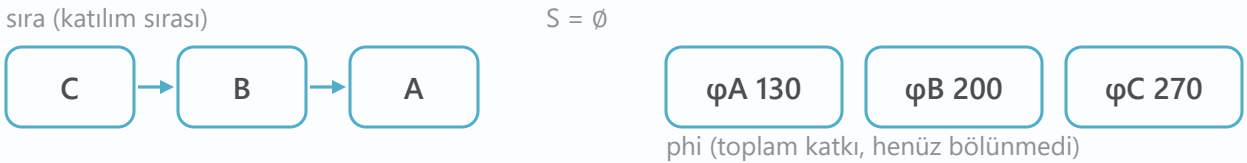
`sira = (C, A, B)`

`S = {A, B, C}` `i = B`

`katki = 60` `phi[A] = 130`

`phi[B] = 200` `phi[C] = 270`

Kod İzleme: `shapley(N, v)` (2/2)



6. sıra (C, B, A): ajanlar bu sırayla gelir. S boş koalisyonla başlar.

Adım 9 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

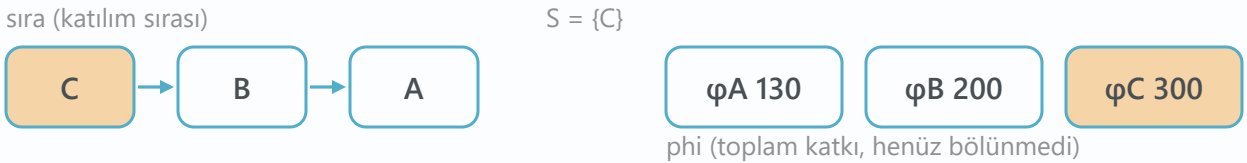
DEĞİŞKENLER

`sira = (C, B, A)` `S = ∅`

`phi[A] = 130` `phi[B] = 200`

`phi[C] = 270`

Kod İzleme: `shapley(N, v)` (2/2)



C katılır: katkı = v(C) − v(∅) = 30 − 0 = **30**; phi[C] = 300.

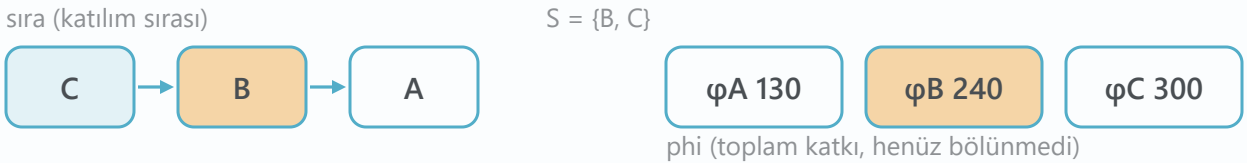
Adım 10 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (C, B, A)` `S = {C}` `i = C` `katki = 30` `phi[A] = 130` `phi[B] = 200` `phi[C] = 300`

Kod İzleme: `shapley(N, v)` (2/2)



B katılır: katkı = v(BC) − v(C) = 70 − 30 = **40**; phi[B] = 240.

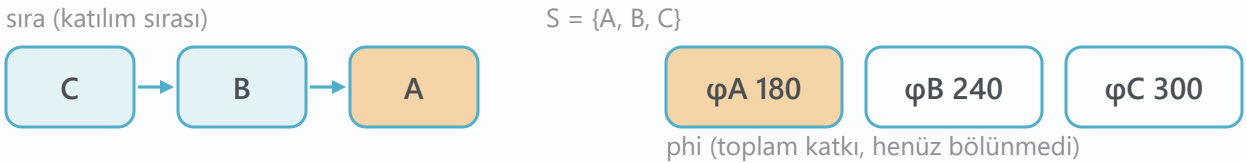
Adım 11 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (C, B, A)` `S = {B, C}` `i = B` `katki = 40` `phi[A] = 130` `phi[B] = 240` `phi[C] = 300`

Kod İzleme: `shapley(N, v)` (2/2)



A katılır: katkı = v(ABC) − v(BC) = 120 − 70 = **50**; phi[A] = 180.

Adım 12 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
▶      for i in sira:
▶        katki = v[S | {i}] - v[S]
▶        phi[i] += katki
▶        S = S | {i}
11   return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (C, B, A)`

`S = {A, B, C}` `i = A`

`katki = 50` `phi[A] = 180`

`phi[B] = 240` `phi[C] = 300`

Kod İzleme: `shapley(N, v)` (2/2)

sıra (katılım sırası)

S = ∅

φA 30	φB 40	φC 50

φ = toplam / 6

Toplamlar 6'ya bölünür: φA = 180/6 = 30, φB = 240/6 = 40, φC = 300/6 = 50. Toplam 120 = v(ABC).

Adım 13 / 13

```
1  from itertools import permutations
2  def shapley(N, v):
3    phi = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      S = frozenset()
7      for i in sira:
8        katki = v[S | {i}] - v[S]
9        phi[i] += katki
10       S = S | {i}
▶    return {i: phi[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`phi[A] = 180` `phi[B] = 240` `phi[C] = 300`

`sonuç = {A: 30, B: 40, C: 50}`

Oyuna Dönüş: Shapley Koalisyon Oyunu

Kuramın cevabı

Oyunun ikili sinerji modelinde marjinal katkıların ortalaması kısa bir kurala iner: φ_i = tek başına değer + içinde yer aldığı her sinerjinin yarısı.

Oyuncu	Tek	Sinerji payı	φ	Eşit pay
A	6	4/2 + 2/2 = 3	**9**	7
B	4	4/2 + 0 = 2	**6**	7
C	5	2/2 + 0 = 1	**6**	7
Toplam	15	6	**21**	21

Altı katılım sırasının ortalaması da (9, 6, 6) verir. Eşit bölüşümde A, katkısının 2 altında kalır.

Gerçekte

Üç kişilik koalisyon oyunlarında başkaları için "adil" paylaşım öneren deneklerin seçimleri, Shapley değeri ile eşit bölüşümün bir karışımı olarak en iyi açıklanmıştır (de Clippel ve Rozen, 2022). Örnekte yarı yarıya karışım (8; 6,5; 6,5) verir.

Oyundaki **eşit bölüşüm hatası** da bu farkı ölçer: $|7 - 9| + |7 - 6| + |7 - 6| = 4$.

Katkıya göre paylaşım (Shapley) ile eşit paylaşım çoğu zaman ayrışır; hangisinin adil sayılacağı bağlama bağlıdır.

Çekirdek (Core)

- x paylaşımı **çekirdektedir** $\Leftrightarrow \Sigma_{i\in N}\, x_i = v(N)$ ve her $S \subseteq N$ için $\Sigma_{i\in S}\, x_i \geq v(S)$.
- Yani hiçbir koalisyon büyük koalisyondan **ayrılarak** (block) üyelerine daha fazlasını sağlayamaz.
- Çekirdek bir doğrusal eşitsizlikler sistemidir (Gillies, 1959): boş, tek nokta ya da bir bölge olabilir.

Tekli kısıtlar

$x_A \geq 10$, $x_B \geq 20$, $x_C \geq 30$

İkili kısıtlar

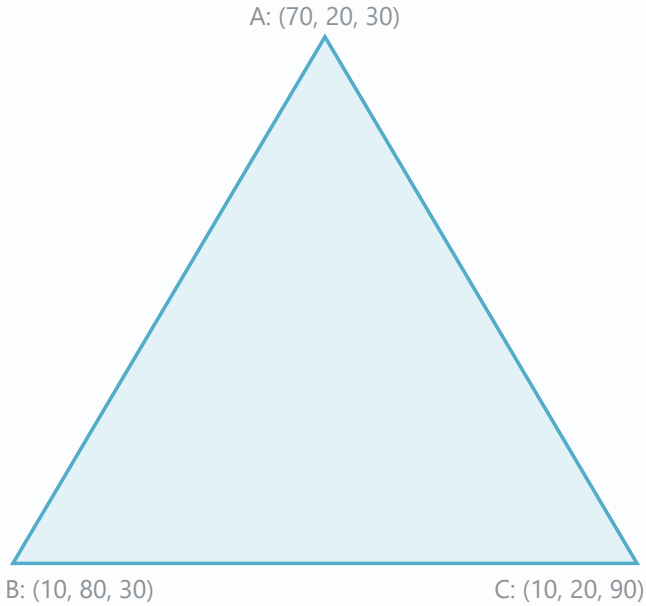
$x_A+x_B \geq 50$, $x_A+x_C \geq 60$, $x_B+x_C \geq 70$

Verimlilik

$x_A+x_B+x_C = 120$

Shapley adalet, çekirdek kararlılık ölçütüdür; ikisi her zaman örtüşmez. **Dışbükey** (convex) oyun: bir oyuncunun katkısı, katıldığı koalisyon büyüdükçe azalmaz. Örneğimiz böyle bir oyun; bu oyunlarda Shapley paylaşımı her zaman çekirdektedir (Shapley, 1971).

Çekirdeğin Geometrisi Adım Adım



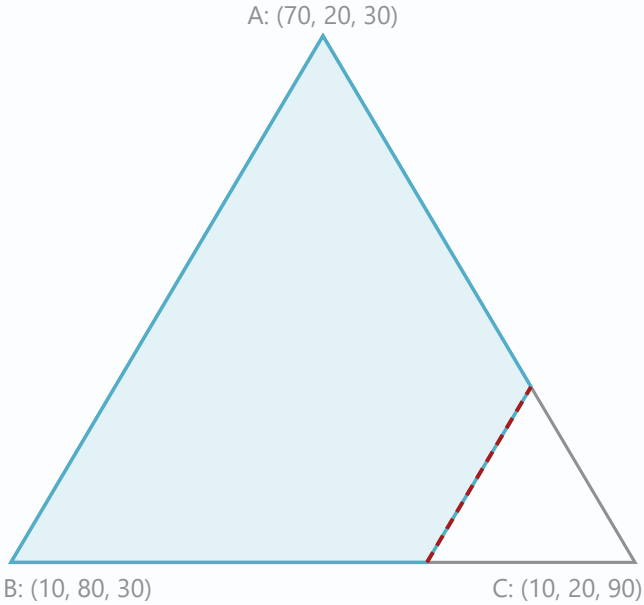
paylaşım x = (xA, xB, xC), toplam 120

üçgen: xA ≥ 10, xB ≥ 20, xC ≥ 30

- A+B ≥ 50 ⇔ xC ≤ 70
- A+C ≥ 60 ⇔ xB ≤ 60
- B+C ≥ 70 ⇔ xA ≤ 50

Bireysel rasyonel ve verimli tüm paylaşımlar bir üçgendir; köşede bir ajan artan 60'ın hepsini alır.

Çekirdeğin Geometrisi Adım Adım



paylaşım x = (xA, xB, xC), toplam 120

üçgen: xA ≥ 10, xB ≥ 20, xC ≥ 30

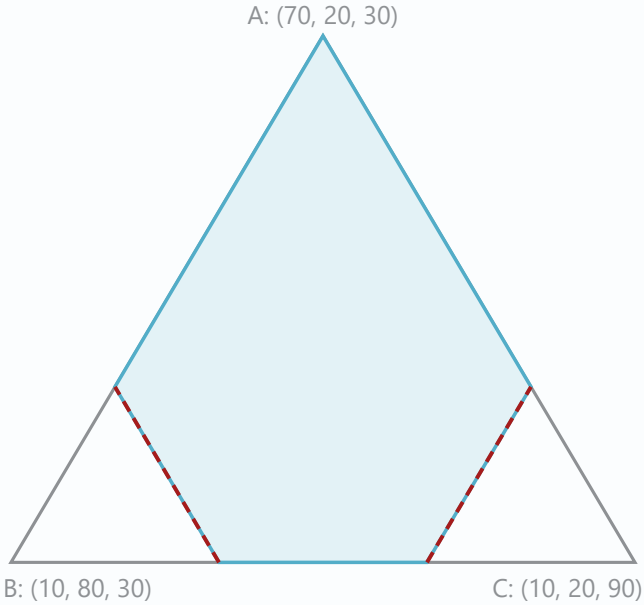
✓ A+B ≥ 50 ⇔ xC ≤ 70

· A+C ≥ 60 ⇔ xB ≤ 60

· B+C ≥ 70 ⇔ xA ≤ 50

{A, B}: $x_A + x_B \geq 50 \Leftrightarrow x_C \leq 70$: C'ye 70'ten fazlası verilirse A ve B ayrılıp kendi 50'lerini alır. C köşesi kesilir.

Çekirdeğin Geometrisi Adım Adım



paylaşım x = (xA, xB, xC), toplam 120

üçgen: xA ≥ 10, xB ≥ 20, xC ≥ 30

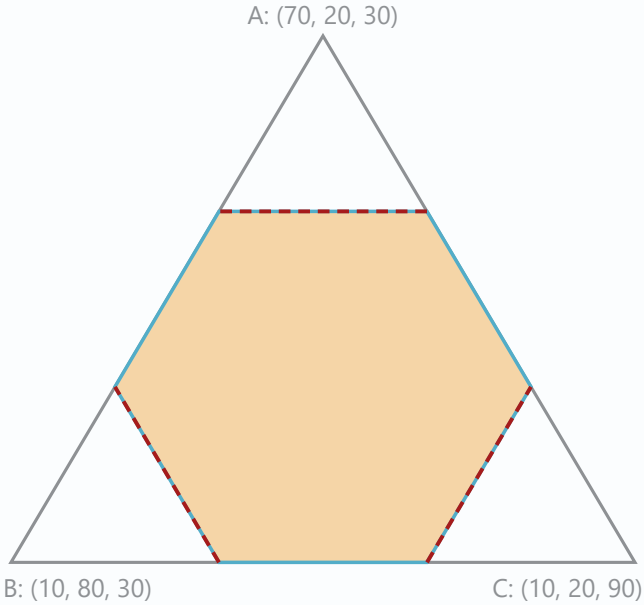
✓ A+B ≥ 50 ⇔ xC ≤ 70

✓ A+C ≥ 60 ⇔ xB ≤ 60

· B+C ≥ 70 ⇔ xA ≤ 50

{A, C}: $x_A + x_C \geq 60 \Leftrightarrow x_B \leq 60$: B köşesi kesilir.

Çekirdeğin Geometrisi Adım Adım



paylaşım x = (xA, xB, xC), toplam 120

üçgen: xA ≥ 10, xB ≥ 20, xC ≥ 30

✓ A+B ≥ 50 ⇔ xC ≤ 70

✓ A+C ≥ 60 ⇔ xB ≤ 60

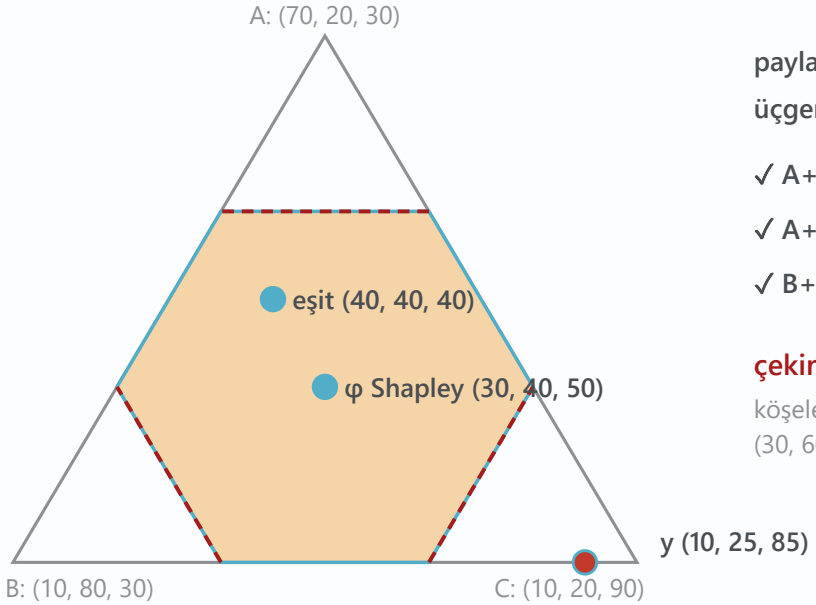
✓ B+C ≥ 70 ⇔ xA ≤ 50

çekirdek: 6 köşeli çokgen

köşeler: (10, 40, 70), (10, 60, 50), (30, 20, 70)
(30, 60, 30), (50, 20, 50), (50, 40, 30)

{B, C}: $x_B + x_C \geq 70 \Leftrightarrow x_A \leq 50$: A köşesi de kesilir. Kalan turuncu altıgen **çekirdektir**.

Çekirdeğin Geometrisi Adım Adım



paylaşım x = (xA, xB, xC), toplam 120

üçgen: xA ≥ 10, xB ≥ 20, xC ≥ 30

✓ A+B ≥ 50 ⇔ xC ≤ 70

✓ A+C ≥ 60 ⇔ xB ≤ 60

✓ B+C ≥ 70 ⇔ xA ≤ 50

çekirdek: 6 köşeli çokgen

köşeler: (10, 40, 70), (10, 60, 50), (30, 20, 70)
(30, 60, 30), (50, 20, 50), (50, 40, 30)

φ = (30, 40, 50) ve eşit bölüşüm çekirdekte (mavi). y = (10, 25, 85) dışarıda (kırmızı): $x_A + x_B = 35 < 50$.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	·	·
{B}	Σ ≥ 20	·	·
{C}	Σ ≥ 30	·	·
{A, B}	Σ ≥ 50	·	·
{A, C}	Σ ≥ 60	·	·
{B, C}	Σ ≥ 70	·	·
{A, B, C}	toplam = 120	·	·

Bir paylaşım çekirdekteyse **hiçbir koalisyon** ayrılıp kendi değerini (v(S)) alarak daha iyi duruma geçemez. İki paylaşımı 7 kısıtla sınayalım.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	·	·
{C}	Σ ≥ 30	·	·
{A, B}	Σ ≥ 50	·	·
{A, C}	Σ ≥ 60	·	·
{B, C}	Σ ≥ 70	·	·
{A, B, C}	toplam = 120	·	·

{A}: v(A) = 10. x için 30 ≥ 10 ✓; y için 10 ≥ 10 ✓.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	·	·
{A, B}	Σ ≥ 50	·	·
{A, C}	Σ ≥ 60	·	·
{B, C}	Σ ≥ 70	·	·
{A, B, C}	toplam = 120	·	·

{B}: v(B) = 20. x için 40 ≥ 20 ✓; y için 25 ≥ 20 ✓.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	50 ✓	85 ✓
{A, B}	Σ ≥ 50	·	·
{A, C}	Σ ≥ 60	·	·
{B, C}	Σ ≥ 70	·	·
{A, B, C}	toplam = 120	·	·

{C}: v(C) = 30. x için 50 ≥ 30 ✓; y için 85 ≥ 30 ✓.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	50 ✓	85 ✓
{A, B}	Σ ≥ 50	70 ✓	35 ✗
{A, C}	Σ ≥ 60	·	·
{B, C}	Σ ≥ 70	·	·
{A, B, C}	toplam = 120	·	·

{A, B}: v(AB) = 50. x için 70 ≥ 50 ✓; y için 35 ≥ 50 ✗. {A, B} birlikte ayrılırsa 50 alır: y'yi **engeller** (block).

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	50 ✓	85 ✓
{A, B}	Σ ≥ 50	70 ✓	35 ✗
{A, C}	Σ ≥ 60	80 ✓	95 ✓
{B, C}	Σ ≥ 70	·	·
{A, B, C}	toplam = 120	·	·

{A, C}: v(AC) = 60. x için 80 ≥ 60 ✓; y için 95 ≥ 60 ✓.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	50 ✓	85 ✓
{A, B}	Σ ≥ 50	70 ✓	35 ✗
{A, C}	Σ ≥ 60	80 ✓	95 ✓
{B, C}	Σ ≥ 70	90 ✓	110 ✓
{A, B, C}	toplam = 120	·	·

{B, C}: v(BC) = 70. x için 90 ≥ 70 ✓; y için 110 ≥ 70 ✓.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	50 ✓	85 ✓
{A, B}	Σ ≥ 50	70 ✓	35 ✗
{A, C}	Σ ≥ 60	80 ✓	95 ✓
{B, C}	Σ ≥ 70	90 ✓	110 ✓
{A, B, C}	toplam = 120	120 ✓	120 ✓

{A, B, C}: v(ABC) = 120. x için 120 = 120 ✓; y için 120 = 120 ✓.

Çekirdek Kontrolü Adım Adım

koalisyon S	kısıt	x = (30, 40, 50)	y = (10, 25, 85)
{A}	Σ ≥ 10	30 ✓	10 ✓
{B}	Σ ≥ 20	40 ✓	25 ✓
{C}	Σ ≥ 30	50 ✓	85 ✓
{A, B}	Σ ≥ 50	70 ✓	35 ✗
{A, C}	Σ ≥ 60	80 ✓	95 ✓
{B, C}	Σ ≥ 70	90 ✓	110 ✓
{A, B, C}	toplam = 120	120 ✓	120 ✓

Sonuç: Shapley paylaşımı x **çekirdekte**; y ise {A, B} tarafından engellendiği için **çekirdek dışında**. (Oyun dışbükey olduğundan Shapley değeri çekirdektedir.)

Çekirdeği Boş Bir Oyun: Çoğunluk Oyunu

- Üç ajan 1 birimlik ödülü paylaşacak; herhangi **iki** ajan anlaşırsa ödülü alır: $v(S) = 1$ ($|S| \geq 2$), aksi hâlde 0.
- Çekirdek kısıtları: $x_A+x_B \geq 1$, $x_A+x_C \geq 1$, $x_B+x_C \geq 1$.
- Toplayınca $2(x_A+x_B+x_C) \geq 3$, yani toplam $\geq 3/2$. Ama toplam = 1 olmalı ⇒ **çekirdek boş**.
- Her paylaşımda en az bir çift $\leq 2/3$ alır ve üçüncüyü dışlayıp 1'i paylaşmak ister: sonsuz pazarlık.

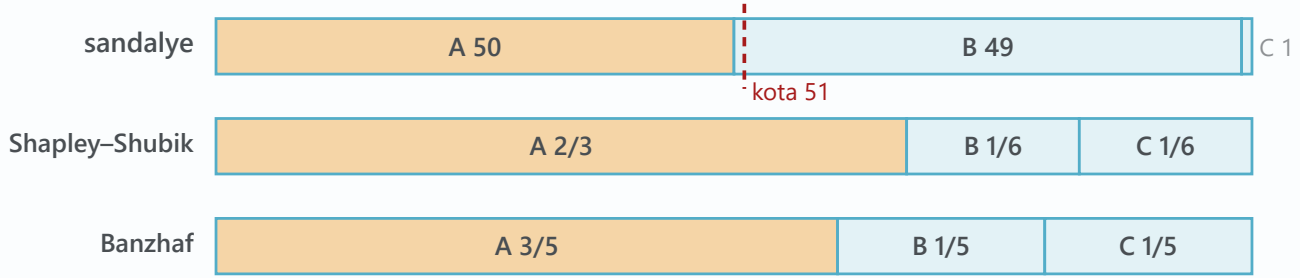
Shapley

$\varphi = (1/3, 1/3, 1/3)$: adil ama kararlı değil.

Ders

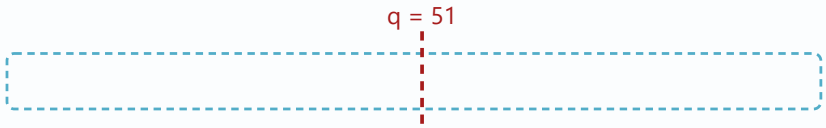
Kararlı paylaşım her oyunda yoktur. O zaman "hiçbir grup ayrılarak çok fazla kazanamasın" gibi gevşetilmiş kurallar kullanılır (ör. ε-çekirdek: hiçbir koalisyon ayrılarak ε'dan fazla kazanamaz).

Ağırlıklı Oylama: Ağırlık ≠ Güç



- **Ağırlıklı oylama oyunu** [q; w_1, …, w_n]: koalisyon ağırlıkları toplamı q'ya ulaşırsa kazanır, v(S) = 1; yoksa 0.
- **Shapley–Shubik endeksi** (1954) = bu oyunun Shapley değeri: i'nin **kilit** (pivotal) olduğu sıraların oranı.
- Meclis [51; 50, 49, 1]: B ile C, 48 sandalye farkına rağmen **eşit** güçtedir (1/6); Banzhaf endeksi kritik üyelikleri sayar: A üç kazanan koalisyonda ({A, B}, {A, C}, {A, B, C}), B ve C birer koalisyonda kilittir ⇒ (3/5, 1/5, 1/5).
- 1958 AET Bakanlar Konseyi [12; 4, 4, 4, 2, 2, 1]: Lüksemburg'un oyu hiçbir koalisyonun sonucunu değiştiremiyordu, güç = 0.

Kod İzleme: `guc_endeksi(N, w, q)`



kilit sayacı

A 0

B 0

C 0

w = {A: 50, B: 49, C: 1}, q = 51. 3! = 6 geliş sırası; her sırada kotayı geçiren oyuncu kilittir.

Adım 1 / 14

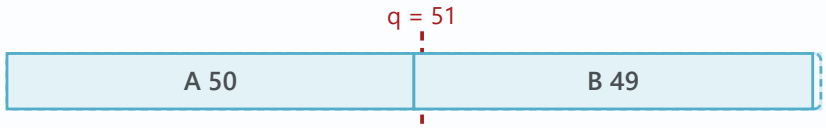
```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
▶    kilit = {i: 0 for i in N}
▶    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`kilit` = **{A: 0, B: 0, C: 0}**

`siralar` = **6 sıra**

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: A → B → C

kilit sayacı

A	0
B	0
C	0

Sıra A → B → C: toplam 50 < 51 iken B gelir, toplam 99 ≥ 51.

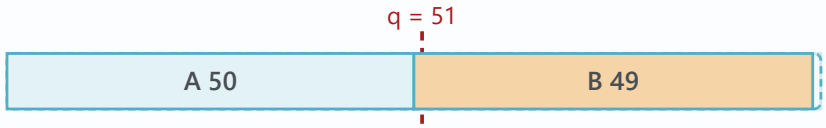
Adım 2 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      toplam = 0
▶      for i in sira:
▶        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (A, B, C)` `i = B` `toplam = 99` `kilit = {A: 0, B: 1, C: 0}`

Kod İzleme: `guc_endeksi(N, w, q)`



kilit sayacı

A 0

B 1

C 0

sıra: A → B → C kilit: B

B kilit: kilit[B] = 1; `break` ile sıradaki geliş sırasına geçilir.

Adım 3 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
▶        if toplam >= q:
▶          kilit[i] += 1
▶          break
12   return {i: kilit[i] / len(siralar) for i in N}
```

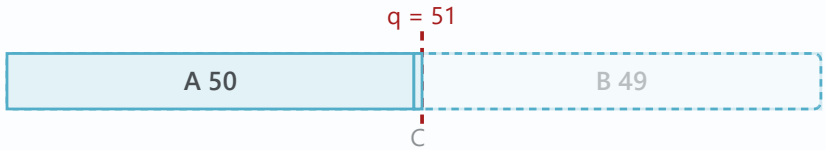
DEĞİŞKENLER

`sira = (A, B, C)` `i = B`

`toplam = 99`

`kilit = {A: 0, B: 1, C: 0}`

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: A → C → B

kilit sayacı

A 0

B 1

C 0

Sıra A → C → B: toplam 50 < 51 iken C gelir, toplam 51 ≥ 51.

Adım 4 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      toplam = 0
▶      for i in sira:
▶        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

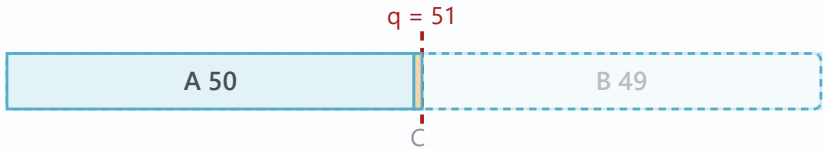
DEĞİŞKENLER

`sira = (A, C, B)` `i = C`

`toplam = 51`

`kilit = {A: 0, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



kilit sayacı

A 0

B 1

C 1

sıra: A → C → B kilit: C

C kilit: kilit[C] = 1; `break` ile sıradaki geliş sırasına geçilir.

Adım 5 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
▶        if toplam >= q:
▶          kilit[i] += 1
▶          break
12   return {i: kilit[i] / len(siralar) for i in N}
```

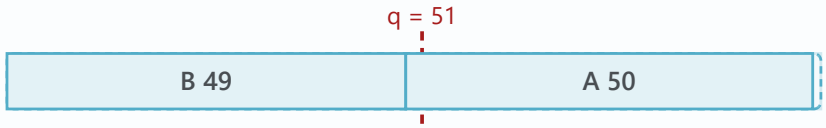
DEĞİŞKENLER

`sira = (A, C, B)` `i = C`

`toplam = 51`

`kilit = {A: 0, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: B → A → C

kilit sayacı

A 0

B 1

C 1

Sıra B → A → C: toplam 49 < 51 iken A gelir, toplam 99 ≥ 51.

Adım 6 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      toplam = 0
▶      for i in sira:
▶        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

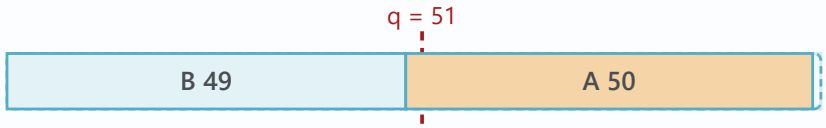
DEĞİŞKENLER

`sira = (B, A, C)` `i = A`

`toplam = 99`

`kilit = {A: 1, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



kilit sayacı

A	1
B	1
C	1

sıra: B → A → C kilit: A

A kilit: kilit[A] = 1; `break` ile sıradaki geliş sırasına geçilir.

Adım 7 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
▶        if toplam >= q:
▶          kilit[i] += 1
▶          break
12   return {i: kilit[i] / len(siralar) for i in N}
```

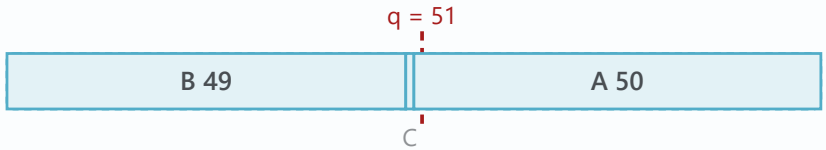
DEĞİŞKENLER

`sira = (B, A, C)` `i = A`

`toplam = 99`

`kilit = {A: 1, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: B → C → A

kilit sayacı

A	1
B	1
C	1

Sıra B → C → A: toplam 50 < 51 iken A gelir, toplam 100 ≥ 51.

Adım 8 / 14

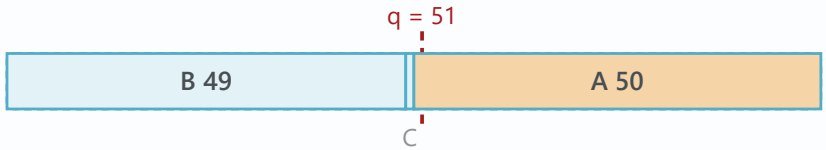
```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      toplam = 0
▶      for i in sira:
▶        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (B, C, A)` `i = A` `toplam = 100`

`kilit = {A: 2, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: B → C → A kilit: A

kilit sayacı

A	2
B	1
C	1

A kilit: kilit[A] = 2; `break` ile sıradaki geliş sırasına geçilir.

Adım 9 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
▶        if toplam >= q:
▶          kilit[i] += 1
▶          break
12   return {i: kilit[i] / len(siralar) for i in N}
```

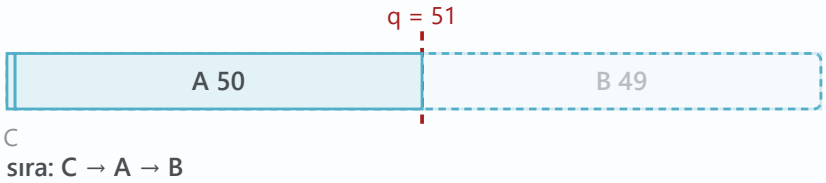
DEĞİŞKENLER

`sira = (B, C, A)` `i = A`

`toplam = 100`

`kilit = {A: 2, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



kilit sayacı

A	2
B	1
C	1

Sıra C → A → B: toplam 1 < 51 iken A gelir, toplam 51 ≥ 51.

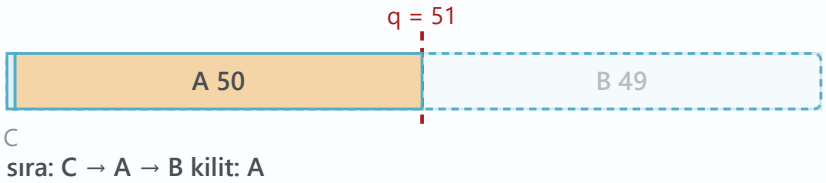
Adım 10 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      toplam = 0
▶      for i in sira:
▶        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`sira = (C, A, B)` `i = A` `toplam = 51` `kilit = {A: 3, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: C → A → B kilit: A



A kilit: kilit[A] = 3; `break` ile sıradaki geliş sırasına geçilir.

Adım 11 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
▶        if toplam >= q:
▶          kilit[i] += 1
▶          break
12   return {i: kilit[i] / len(siralar) for i in N}
```

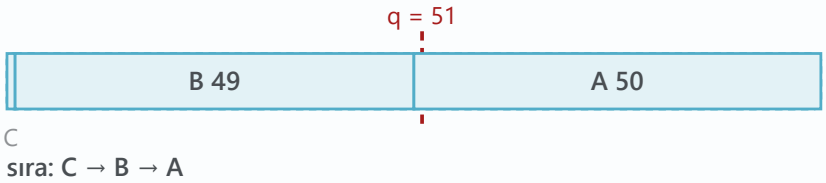
DEĞİŞKENLER

`sira = (C, A, B)` `i = A`

`toplam = 51`

`kilit = {A: 3, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`





Sıra C → B → A: toplam 50 < 51 iken A gelir, toplam 100 ≥ 51.

Adım 12 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
▶    for sira in siralar:
▶      toplam = 0
▶      for i in sira:
▶        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
12   return {i: kilit[i] / len(siralar) for i in N}
```

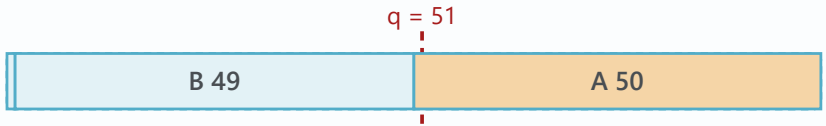
DEĞİŞKENLER

`sira = (C, B, A)` `i = A`

`toplam = 100`

`kilit = {A: 4, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`



sıra: C → B → A kilit: A

kilit sayacı



A kilit: kilit[A] = 4; `break` ile sıradaki geliş sırasına geçilir.

Adım 13 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
▶        if toplam >= q:
▶          kilit[i] += 1
▶          break
12   return {i: kilit[i] / len(siralar) for i in N}
```

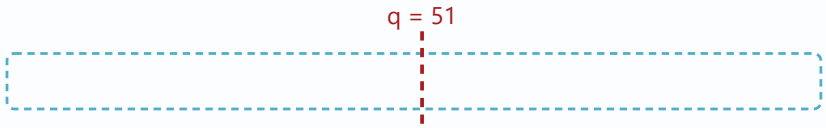
DEĞİŞKENLER

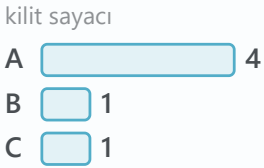
`sira = (C, B, A)` `i = A`

`toplam = 100`

`kilit = {A: 4, B: 1, C: 1}`

Kod İzleme: `guc_endeksi(N, w, q)`





Güç = kilit / 6 ⇒ (2/3, 1/6, 1/6). B'nin 49 sandalyesi C'nin 1 sandalyesinden fazla güç getirmiyor.

Adım 14 / 14

```
1  from itertools import permutations
2  def guc_endeksi(N, w, q):
3    kilit = {i: 0 for i in N}
4    siralar = list(permutations(N))
5    for sira in siralar:
6      toplam = 0
7      for i in sira:
8        toplam += w[i]
9        if toplam >= q:
10         kilit[i] += 1
11         break
▶    return {i: kilit[i] / len(siralar) for i in N}
```

DEĞİŞKENLER

`kilit = {A: 4, B: 1, C: 1}`

`sonuc = {A: 0.667, B: 0.167, C: 0.167}`

Kamu Malı Oyunu (Public Goods Game)

- **Kamu malı**: kimse kullanımdan dışlanamaz ve bir kişinin kullanımı diğerininkini azaltmaz (ör. açık kaynak yazılım, temiz hava, ortak veri kümesi).
- Oyun: n oyuncu, her biri e jeton. Oyuncu i ortak havuza 0 ile e arasında k_i jeton katar. Havuz r çarpanıyla büyür ve **herkese eşit** dağıtılır.
- Kazanç: $u_i = e - k_i + (r/n)\cdot\Sigma_j\ k_j$ ($\Sigma_j k_j$: havuza giren toplam jeton)
- r/n: bir jetonun katkı yapana geri dönen kısmı (MPCR: kişi başı marjinal getiri). **1/n < r/n < 1** ise ikilem vardır.

Örneğimiz

n = 4, e = 10, r = 1,6 ⇒ r/n = 0,4

Bireysel

1 jetonluk katkı: katkı verene 0,4 döner ⇒ net −0,6

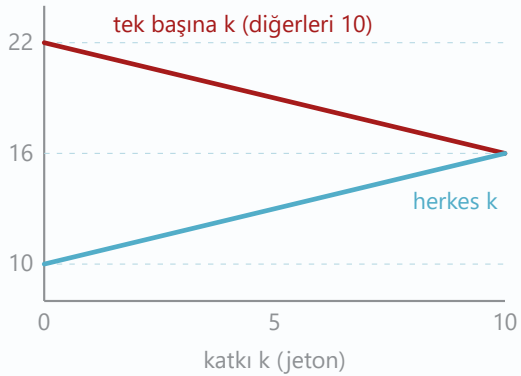
Sosyal

1 jetonluk katkı: gruba 1,6 döner ⇒ net +0,6

KAMU MALI OYUNU

Neden Beleşçilik (Free Riding)?

- $u_1 = 10 - k_1 + 0{,}4 \cdot (k_1 + \text{diğerleri})$. O1 bir jeton daha verirse cebinden 1 jeton çıkar, havuzdan 0,4 geri gelir: net **−0,6**.
- Diğerleri ne yaparsa yapsın her jetonluk katkı O1'e 0,6 kaybettirir: **k = 0 baskın strateji**.
- Tek Nash dengesi: herkes 0 → kişi başı 10. Oysa herkes 10 verse kişi başı 16: denge **Pareto optimal değil**.
- Kırmızı doğru: diğerleri tam katkı verirken O1'in kazancı 22'den 16'ya düşer. Mavi: herkes aynı k'yi verirse kişi başı 10'dan 16'ya çıkar.
- Çoklu ajan karşılığı: dosya paylaşım ağında yükleme yapmadan yalnızca indiren eş, federe öğrenmede güncelleme göndermeden ortak modeli kullanan istemci.



Mahkûmun İkilemi'nin n kişilik hâli: bireysel teşvik (−0,6) ile sosyal refah (+0,6) ters yönde.

ÖZET

Özet

Ajan ve MAS

Otonom, reaktif, proaktif ve sosyal ajanlar; işbirlikçi, rekabetçi ya da karma sistemler; çoğu gerçek sistem hibrit kontrol kullanır.

Sözleşme ağı

Duyuru → teklif → ödül. Dağıtık ve basit, ama açgözlü: örnekte 18 dk vs optimum 13 dk.

Koalisyon oyunları

v(S) koalisyon değerini verir. Shapley = ortalama marjinal katkı: (30, 40, 50). Ağırlıklı oylamada güç ≠ ağırlık.

Çekirdek

Hiçbir koalisyonun ayrılmak istemediği paylaşımlar; boş olabilir (çoğunluk oyunu).

Kamu malı

$r/n < 1$ ise katkı vermemek baskın: denge 10, optimum 16 (kişi başı).

Beleşçilik

Ortak kaynağa katkı vermeyen ajan (yalnızca indiren eş) bireysel olarak kazanır, sistem kaybeder.

ÖZET

Etkileşimli Oyunlar

Shapley Koalisyon Oyunu

Karakteristik fonksiyondan marjinal katkılar ve Shapley değeri.

Kamu Malı Oyunu

Turlar boyunca katkı ve beleşçilerin etkisi.

ÖZET

Örnek Problemler

1. Sözleşme ağı örneğinde görevler G3, G2, G1 sırasıyla ihaleye çıksaydı atama ve toplam süre ne olurdu?
2. Eldiven oyunu: A'nın sol, B ve C'nin sağ eldiveni var; bir çift 1 değerinde. v'yi yazın, Shapley değerini hesaplayın.
3. Aynı eldiven oyununun çekirdeği nedir?
4. Ana örnekte (40, 40, 40) çekirdekte mi? (20, 25, 75) çekirdekte mi?
5. Kamu malı: n = 5, e = 10, r = 2. Herkes tam katkı verirse ve biri beleşçi olursa kazançlar?
6. Kamu malı oyununda r ≥ n olsaydı baskın strateji ne olurdu?

▶ **Cevapları göster**

Kaynaklar

- M. Wooldridge, *An Introduction to MultiAgent Systems*, 2. baskı, Wiley, 2009.
- Y. Shoham, K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge University Press, 2009 (Bölüm 12).
- G. Chalkiadakis, E. Elkind, M. Wooldridge, *Computational Aspects of Cooperative Game Theory*, Morgan & Claypool, 2011.
- R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver", *IEEE Trans. Computers* C-29(12), 1980.
- L. S. Shapley, "A value for n-person games", *Contributions to the Theory of Games II*, Princeton, 1953; "Cores of convex games", *Int. J. Game Theory* 1, 1971; L. S. Shapley, M. Shubik, *APSR* 48(3), 1954.
- G. de Clippel, K. Rozen, "Fairness through the lens of cooperative game theory: An experimental approach", *American Economic Journal: Microeconomics* 14(3), 2022.
- T. Sandholm vd., "Coalition structure generation with worst case guarantees", *Artificial Intelligence* 111, 1999.

ÖZET

Kaynaklar: Kitaplar, Dersler ve Araçlar

- A. R. Karlin, Y. Peres, *Game Theory, Alive*, American Mathematical Society, 2017 (koalisyonlar ve Shapley değeri bölümü).
- N. Nisan, T. Roughgarden, É. Tardos, V. Vazirani (ed.), *Algorithmic Game Theory*, Cambridge University Press, 2007 (Bölüm 15: maliyet paylaşımı, çekirdek ve Shapley değeri).
- C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model", *SIGGRAPH '87*, 1987; T. C. Schelling, "Dynamic models of segregation", *J. Mathematical Sociology* 1(2), 1971.
- FIPA, *Contract Net Interaction Protocol Specification* (SC00029H), 2002.
- M. Wooldridge, N. R. Jennings, "Intelligent agents: Theory and practice", *Knowledge Engineering Review* 10(2), 1995; D. B. Gillies, "Solutions to general non-zero-sum games", *Contributions to the Theory of Games IV*, Princeton, 1959.
- O. N. Bondareva, "Some applications of linear programming methods to the theory of cooperative games", *Problemy Kibernetiki* 10, 1963; L. S. Shapley, "On balanced sets and cores", *Naval Research Logistics Quarterly* 14(4), 1967; J. Schaeffer vd., "Checkers is solved", *Science* 317(5844), 2007.
- Ders sayfaları: N. Shah, CSC304 (Toronto) · A. Procaccia, 15-896 (CMU) · T. Roughgarden, CS364A (Stanford) · M. Hoefer, AGT (Frankfurt)
- Araçlar: Nashpy (iki oyunculu oyunlarda Nash dengesi; ör. çıkar yelpazesindeki üç oyun) · Gambit (açık kaynak oyun kuramı araçları).

Son

Bölüm 10: Çoklu Ajan Sistemleri

Sonraki bölüm: Bölüm 11: Algoritmik Adalet ve Etik

Çalışma soruları · Terim sözlüğü