Bölüm 6: Yaklaşım Algoritmaları

İleri Algoritmalar

Yaklaşım oranı | Gezgin satıcı (TSP) | En yakın komşu ve MST | 2-opt yerel arama | İş çizelgeleme | Düğüm kapsama | Sırt çantası | Power Iteration

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü:
https://sercankulcu.github.io/files/advanced_algorithms/slides/Bolum_06_Yaklasim_Algoritmalari.html

GİRİŞ

Öğrenme Çıktıları

Kavram

Yaklaşım (approximation) algoritmasının ne olduğunu, **yaklaşım oranını** ve sezgisel yöntemlerden farkını açıklamak.

TSP

Gezgin satıcı problemi için en yakın komşu, MST tabanlı 2-yaklaşım ve 2-opt yerel aramasını adım adım uygulamak.

Çizelgeleme ve Kapsama

Makine çizelgeleme (liste çizelgeleme, LPT) ve düğüm kapsama 2-yaklaşımını Java ile yazıp izlemek; oranlarını kanıtlamak.

Sayısal Yakınsama

Power Iteration ile en büyük özdeğere ve özvektöre yinelemeyle nasıl yakınsandığını hesaplamak.

GİRİŞ

Yaklaşım Algoritmaları (Approximation Algorithms)

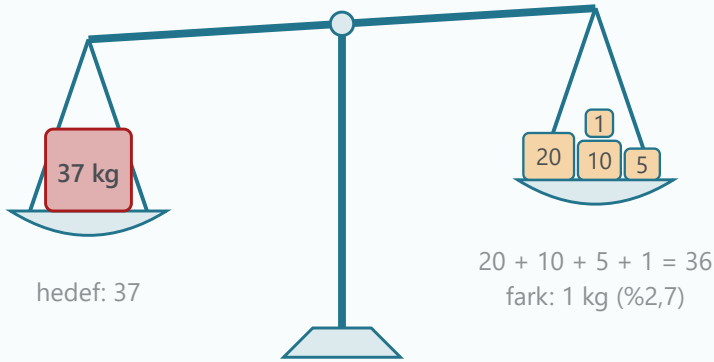
- Bazı problemlerin **kesin (optimal)** çözümünü bulmak çok zor ya da pratikte **imkânsız** olabilir: TSP, düğüm kapsama, makine çizelgeleme ve sırt çantası **NP-zordur**; P ≠ NP ise polinom zamanlı kesin algoritmaları yoktur.
- Yaklaşım algoritmaları **mükemmel çözümü** aramak yerine, optimal çözüme **mümkün olduğunca yakın** çözümler üretir.
- **Polinom zamanda** çalışırlar: karmaşık problemler makul sürede çözülebilir.
- Çözümün optimal çözüme **belli bir oran ya da yüzde** içinde yakın olduğu çoğu zaman **kanıtlanabilir**.
- Hesaplama kaynakları (zaman, bellek) **kısıtlı** olduğunda tercih edilir.

Takas: **optimallikten** biraz vazgeçilir, karşılığında **verimlilik** ve çözüm kalitesi için bir **garanti** kazanılır.

GİRİŞ

Terazi Benzetmesi

- Terazinin bir kefesine bir ağırlık konduğunda, diğer kefeye **tam olarak aynı** ağırlığı koymak zor olabilir.
- Bu durumda, eldeki çeşitli **küçük ağırlıklar** kullanılarak büyük ağırlığa **olabildiğince yakın** bir denge kurulur.
- Elimizde yalnızca 20, 10, 5 ve 1 kg varsa ve her biri bir kez kullanılabiliyorsa, 37 kg'a en yakın denge 36 kg'dır.



Yaklaşım algoritması da böyle çalışır: kesin dengeyi (optimumu) aramak yerine, **hızlıca** bulunan ve hatası **sınırlı** olan bir dengeyle yetinir.

GİRİŞ

Yaklaşım Oranı (Approximation Ratio)

C: algoritmanın bulduğu çözümün maliyeti, C*: optimal çözümün maliyeti. Algoritma her girdi için

$$\max(C / C^*, C^* / C) \le \rho(n)$$

koşulunu sağlıyorsa bir **ρ(n)-yaklaşım algoritmasıdır** (ρ ≥ 1; ρ = 1 ise algoritma optimaldir).

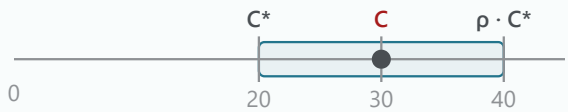
En küçükleme (minimization)

C ≤ ρ · C*. Örnek: 2-yaklaşım bir düğüm kapsama algoritması, en fazla optimumun **iki katı** düğüm seçer.

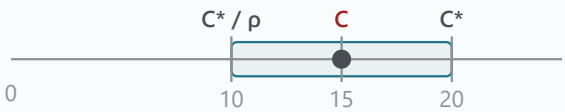
En büyükleme (maximization)

C ≥ C* / ρ. Örnek: 2-yaklaşım bir sırt çantası algoritması, optimal değerin en az **yarısını** garanti eder.

En küçükleme (ρ = 2): C* ≤ C ≤ ρ · C*



En büyükleme (ρ = 2): C* / ρ ≤ C ≤ C*



Oran bir **en kötü durum** garantisidir; pratikte çözümler çoğu zaman optimuma çok daha yakındır.

GİRİŞ

Yaklaşım Algoritması, Sezgisel ve Yaklaşım Şemaları

Yaklaşım	Garanti	Örnek
Kesin algoritma	Optimal; ama NP-zor problemlerde üstel zaman	Kaba kuvvet TSP: $(n-1)!/2$ tur
Yaklaşım algoritması	Polinom zaman + kanıtlanmış oran ρ	Düğüm kapsama: $\rho = 2$
PTAS	Her sabit $\varepsilon > 0$ için $(1 + \varepsilon)$ oranı, n'de polinom (ε'da üstel olabilir)	Öklid TSP (Arora, 1998)
FPTAS	$(1 + \varepsilon)$ oranı, hem n'de hem $1/\varepsilon$'da polinom	Sırt çantası: $O(n^3/\varepsilon)$
Sezgisel (heuristic)	Garanti yok; pratikte genellikle iyi	2-opt, en yakın komşu
Metasezgisel	Garanti yok; problemden bağımsız arama çerçevesi	Genetik algoritma, karınca kolonisi

Rastgele yuvarlama (randomized rounding) gibi rastgele yaklaşım algoritmaları da vardır; beklenen değerde bir oran garanti ederler.

GİRİŞ

Yaklaşım Algoritmalarının Uygulamaları

Gezgin satıcı (TSP)

Satıcının tüm müşterilere en kısa toplam yolla uğrayıp geri döndüğü rotaya karar vermek: lojistik, kargo, devre kartı delme.

Sırt çantası (knapsack)

Çantaya sığacak ve en büyük faydayı sağlayacak eşyaları seçmek: bütçe, yük ve portföy planlama.

İş (job) çizelgeleme

Birden fazla işin ya da sürecin makinelerde/işlemcilerde en verimli şekilde yürütülmesi.

Düğüm kapsama ve sayısal yöntemler

Ağlara en az sayıda izleme noktası yerleştirmek; büyük matrislerin özdeğerine yinelemeyle yakınsamak (PageRank).

GİRİŞ

Yaklaşım Algoritmalarının Türleri

Basitleştirme

Problemin **daha basit bir sürümü** çözülür (ör. tam sayı kısıtı gevşetilir: LP gevşetmesi, MST ile TSP alt sınırı).

Açgözlü (greedy)

Her adımda **o an en iyi görünen** seçenek seçilerek ilerlenir (en yakın komşu, LPT, oran sıralı sırt çantası).

Yerel arama (local search)

Mevcut çözüm **küçük değişikliklerle** iyileştirilir; daha iyi komşu kalmayınca durulur (2-opt).

Primal–dual

LP'nin birincil ve eşlenik (dual) çözümleri birlikte büyütülür; kapsama problemlerinde sık kullanılır.

Rastgele

Rastgele yuvarlama ya da örnekleme; beklenen oran garantisi.

Metasezgisel

Benzetimli tavlama, genetik algoritma, karınca kolonisi.

Gezgin Satıcı Problemi (TSP)

- **En iyileme problemi:** satıcı, bir dizi şehri **en kısa toplam uzaklıkta** birer kez ziyaret edip başladığı şehre dönmek ister.
- **Kombinatoryal problem:** çözüm, tüm şehirlerin **gezilme sırasıdır** (bir permütasyon).
- **NP-zor:** çok sayıda şehir için en iyi rotayı bulmak pratikte zordur. İki düğüm arasındaki en kısa yol (Dijkstra) ise polinom zamanda çözülür; iki problem karıştırılmamalıdır.
- **Girdi:** n şehir (0, 1, 2, … ya da A, B, C, …) ve $\mathbf{d_{ij}}$: i ve j şehirleri arasındaki uzaklık.
- Simetrik TSP: $d_{ij} = d_{ji}$; çizge kuramında: tam çizgedeki en hafif **Hamilton çevrimi**.

d	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

Örnek: 5 şehrin uzaklık matrisi.

Problemin Çözümü

Tam çözüm

Tüm olası rotaları kontrol etmek (kaba kuvvet) ya da dinamik programlama (Held–Karp, $O(n^2 \cdot 2^n)$).

Yaklaşık çözümler

Sezgisel algoritmalar: en yakın komşu (nearest neighbor), MST tabanlı 2-yaklaşım, Christofides, genetik algoritmalar.

Optimizasyon araçları

Yerel arama (2-opt), benzetimli tavlama (simulated annealing), karınca kolonisi optimizasyonu (ant colony optimization).

n	(n−1)!/2 farklı tur
5	12
10	181.440
15	$\approx 4{,}4 \cdot 10^{10}$
20	$\approx 6{,}1 \cdot 10^{16}$
25	$\approx 3{,}1 \cdot 10^{23}$

Başlangıç şehri sabitlenir ve iki yön aynı sayılır. Saniyede 10^9 tur denense bile n = 25 için $\approx 10^7$ yıl gerekir.

En Yakın Komşu (Nearest Neighbor) Sezgiseli

1. Bir başlangıç şehri seçilir ve ziyaret edildi olarak işaretlenir.
2. Bulunulan şehirden **ziyaret edilmemiş en yakın** şehre gidilir (açgözlü seçim).
3. Tüm şehirler ziyaret edilene kadar 2. adım tekrarlanır.
4. Son şehirden **başlangıç şehrine dönülür** ve tur kapanır.

Artıları

+ Çok basit; **O(n^2)** zaman.
+ Genellikle optimumun %25 kadar üzerinde tur verir.
+ Yerel arama için iyi bir **başlangıç turu**.

Eksileri

– Sabit bir yaklaşım oranı **yoktur**; metrik TSP'de bile oran Θ(log n) olabilir.
– Son dönüş kenarı çok uzun kalabilir.

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
▶    int n = d.length, toplam = 0, simdiki = bas;
▶    boolean[] ziyaret = new boolean[n];
▶    int[] tur = new int[n + 1];
▶    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0

Başlangıç şehri 0: `ziyaret[0] = true`, `tur = [0]`, `toplam = 0`.

Adım 1 / 11

DEĞİŞKENLER

`k = –` `simdiki = 0` `enYakin = –`
`enAz = –` `toplam = 0` `tur = [0]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
▶      int enYakin = -1, enAz = Integer.MAX_VALUE;
▶      for (int j = 0; j < n; j++)
▶        if (!ziyaret[j] && d[simdiki][j] < enAz) {
▶          enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0

0. satır taranır, ziyaret edilmemişler: d[0][1] = 10, d[0][2] = 15, d[0][3] = 20, d[0][4] = 25 → en küçüğü **1** (10).

Adım 2 / 11

DEĞİŞKENLER

`k = 1` `simdiki = 0` `enYakin = 1` `enAz = 10` `toplam = 0` `tur = [0]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
▶      tur[k] = enYakin; ziyaret[enYakin] = true;
▶      toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1

`tur[1] = 1`, `toplam += 10` → 10; artık `simdiki = 1`.

Adım 3 / 11

DEĞİŞKENLER

`k = 1` `simdiki = 1` `enYakin = 1`

`enAz = 10` `toplam = 10` `tur = [0, 1]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
▶      int enYakin = -1, enAz = Integer.MAX_VALUE;
▶      for (int j = 0; j < n; j++)
▶        if (!ziyaret[j] && d[simdiki][j] < enAz) {
▶          enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1

1. satır taranır, ziyaret edilmemişler: d[1][2] = 35, d[1][3] = 25, d[1][4] = 30 → en küçüğü **3** (25).

Adım 4 / 11

DEĞİŞKENLER

`k = 2` `simdiki = 1` `enYakin = 3` `enAz = 25` `toplam = 10` `tur = [0, 1]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
▶      tur[k] = enYakin; ziyaret[enYakin] = true;
▶      toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3

`tur[2] = 3`, `toplam += 25` → 35; artık `simdiki = 3`.

Adım 5 / 11

DEĞİŞKENLER

`k = 2` `simdiki = 3` `enYakin = 3` `enAz = 25` `toplam = 35` `tur = [0, 1, 3]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
▶      int enYakin = -1, enAz = Integer.MAX_VALUE;
▶      for (int j = 0; j < n; j++)
▶        if (!ziyaret[j] && d[simdiki][j] < enAz) {
▶          enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3

3. satır taranır, ziyaret edilmemişler: d[3][2] = 30, d[3][4] = 40 → en küçüğü **2** (30).

Adım 6 / 11

DEĞİŞKENLER

`k = 3` `simdiki = 3` `enYakin = 2` `enAz = 30` `toplam = 35` `tur = [0, 1, 3]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
▶      tur[k] = enYakin; ziyaret[enYakin] = true;
▶      toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3 → 2

`tur[3] = 2`, `toplam += 30` → 65; artık `simdiki = 2`.

Adım 7 / 11

DEĞİŞKENLER

`k = 3` `simdiki = 2` `enYakin = 2` `enAz = 30` `toplam = 65` `tur = [0, 1, 3, 2]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
▶      int enYakin = -1, enAz = Integer.MAX_VALUE;
▶      for (int j = 0; j < n; j++)
▶        if (!ziyaret[j] && d[simdiki][j] < enAz) {
▶          enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3 → 2

2. satır taranır, ziyaret edilmemişler: d[2][4] = 35 → en küçüğü **4** (35).

Adım 8 / 11

DEĞİŞKENLER

`k = 4` `simdiki = 2` `enYakin = 4` `enAz = 35` `toplam = 65` `tur = [0, 1, 3, 2]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
▶      tur[k] = enYakin; ziyaret[enYakin] = true;
▶      toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3 → 2 → 4

`tur[4] = 4`, `toplam += 35` → 100; artık `simdiki = 4`.

Adım 9 / 11

DEĞİŞKENLER

`k = 4` `simdiki = 4` `enYakin = 4` `enAz = 35` `toplam = 100` `tur = [0, 1, 3, 2, 4]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
▶    tur[n] = bas; toplam += d[simdiki][bas];
17   return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3 → 2 → 4 → 0

Tüm şehirler ziyaret edildi; başlangıca dönülür: `toplam += d[4][0]` = 25 → **125**.

Adım 10 / 11

DEĞİŞKENLER

`k = 5` `simdiki = 4` `enYakin = –` `enAz = –` `toplam = 125`

`tur = [0, 1, 3, 2, 4, 0]`

En Yakın Komşu: Kodu Adım Adım İzle

```
1  int[] enYakinKomsu(int[][] d, int bas) {
2    int n = d.length, toplam = 0, simdiki = bas;
3    boolean[] ziyaret = new boolean[n];
4    int[] tur = new int[n + 1];
5    tur[0] = bas; ziyaret[bas] = true;
6    for (int k = 1; k < n; k++) {
7      int enYakin = -1, enAz = Integer.MAX_VALUE;
8      for (int j = 0; j < n; j++)
9        if (!ziyaret[j] && d[simdiki][j] < enAz) {
10         enAz = d[simdiki][j]; enYakin = j; }
11     // en yakın şehre git
12     tur[k] = enYakin; ziyaret[enYakin] = true;
13     toplam += enAz; simdiki = enYakin;
14   }
15   // başlangıca dön
16   tur[n] = bas; toplam += d[simdiki][bas];
▶    return tur;
18 }
```

	0	1	2	3	4
0	0	10	15	20	25
1	10	0	35	25	30
2	15	35	0	30	35
3	20	25	30	0	40
4	25	30	35	40	0

tur: 0 → 1 → 3 → 2 → 4 → 0

Tur 0 → 1 → 3 → 2 → 4 → 0, uzunluk **125**. (Kaba kuvvetle doğrulandı: bu örnekte optimumdur.)

Adım 11 / 11

DEĞİŞKENLER

`k = 5` `simdiki = 4` `enYakin = –`

`enAz = –` `toplam = 125`

`tur = [0, 1, 3, 2, 4, 0]`

Kaba Kuvvetle Karşılaştırma: 12 Farklı Tur

Tur	Uzunluk
0 → 1 → 3 → 2 → 4 → 0	125
0 → 1 → 3 → 4 → 2 → 0	125
0 → 1 → 4 → 2 → 3 → 0	125
0 → 1 → 4 → 3 → 2 → 0	125
0 → 2 → 3 → 1 → 4 → 0	125
0 → 2 → 4 → 1 → 3 → 0	125

Tur	Uzunluk
0 → 1 → 2 → 3 → 4 → 0	140
0 → 1 → 2 → 4 → 3 → 0	140
0 → 2 → 1 → 3 → 4 → 0	140
0 → 2 → 1 → 4 → 3 → 0	140
0 → 3 → 1 → 2 → 4 → 0	140
0 → 3 → 2 → 1 → 4 → 0	140

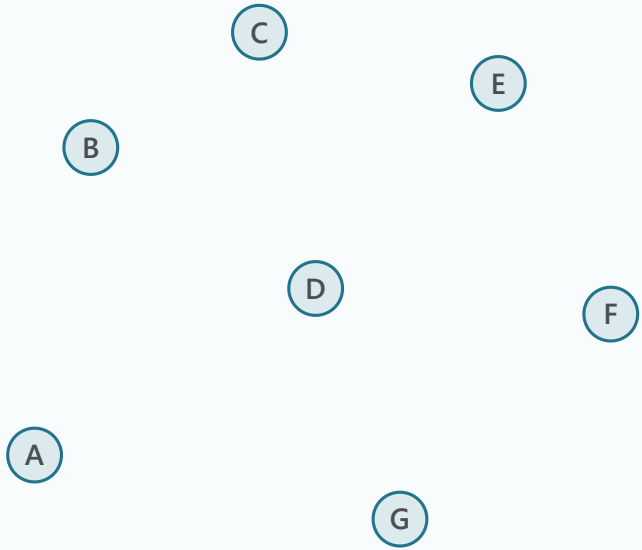
- (5−1)!/2 = 12 tur denendi: optimum **125**; en yakın komşu da 125 buldu (bu örnek için şanslı bir durum).
- Bu matris **üçgen eşitsizliğini** sağlamaz: $d_{12} = 35 > d_{10} + d_{02} = 10 + 15$. Yani örnek **metrik değildir**.

Üçgen Eşitsizliği ve Yaklaşım Sınırları

Metrik TSP: her i, j, k için $d_{ij} \leq d_{ik} + d_{kj}$ (doğrudan gitmek hiçbir zaman daha uzun değildir)

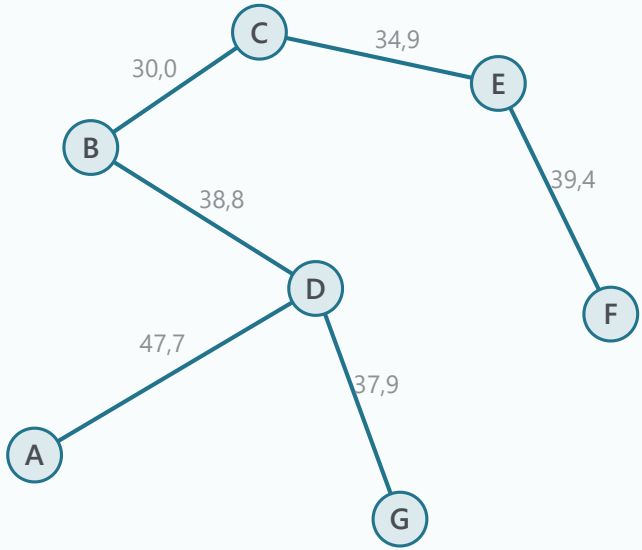
Durum	Oran	Açıklama
Genel TSP	yok	P ≠ NP ise hiçbir sabit ρ için polinom zamanlı ρ-yaklaşım yoktur: böyle bir algoritma Hamilton çevrimi problemini çözerdi.
Metrik TSP, MST tabanlı	2	MST + önce-kök (preorder) gezinme + kısayol; $O(n^2)$.
Metrik TSP, Christofides	1,5	MST + tek dereceli düğümlerde en küçük mükemmel eşleştirme + Euler turu (1976).
Öklid TSP	1 + ε	Düzlemdeki noktalar için PTAS vardır (Arora, Mitchell).
En yakın komşu	Θ(log n)	Metrik durumda bile sabit oran garanti etmez.

Metrik TSP için MST Tabanlı 2-Yaklaşım



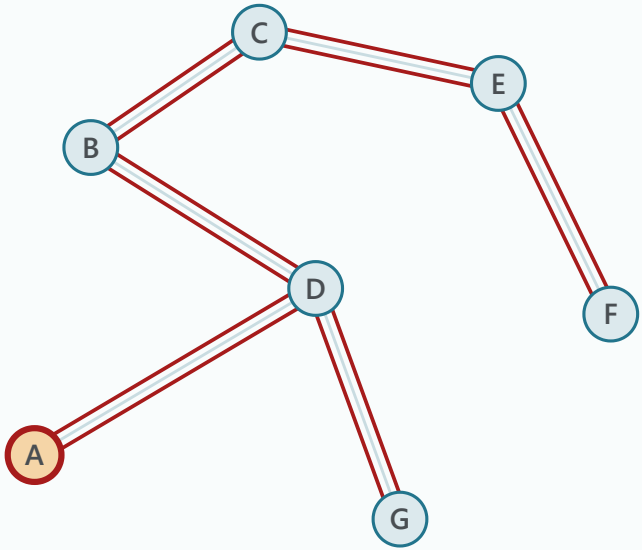
Metrik örnek: düzlemde 7 şehir; uzaklıklar Öklid uzaklığıdır (üçgen eşitsizliği sağlanır).

Metrik TSP için MST Tabanlı 2-Yaklaşım



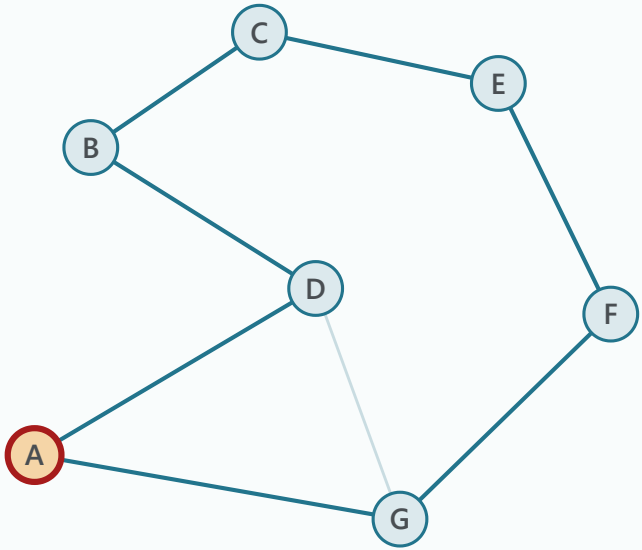
1) Prim ile **minimum kapsayan ağaç** (MST) kurulur: toplam ağırlık w(T) = 228,8. Bir turdan bir kenar silinince kapsayan ağaç kalır → **w(T) ≤ OPT**.

Metrik TSP için MST Tabanlı 2-Yaklaşım



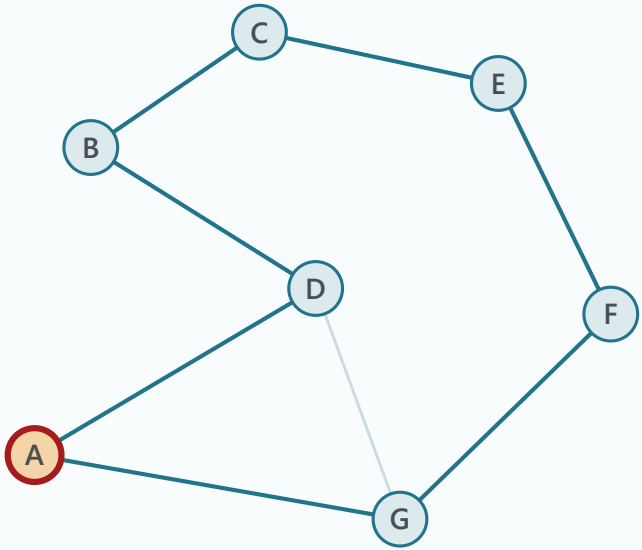
2) Ağaç A'dan derinlik öncelikli **iki kez** gezilir (her kenar bir gidiş bir dönüş): A → D → B → C → E → F → E → C → B → D → G → D → A. Uzunluk 2 · w(T) = 457,6.

Metrik TSP için MST Tabanlı 2-Yaklaşım



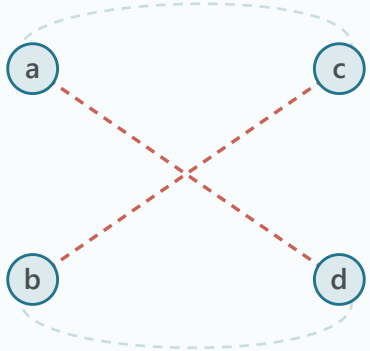
3) Tekrar eden şehirler atlanır (**kısayol**): önce-kök sırası A → D → B → C → E → F → G → A. Üçgen eşitsizliği sayesinde kısayol yolu uzatmaz: tur = 287,7.

Metrik TSP için MST Tabanlı 2-Yaklaşım

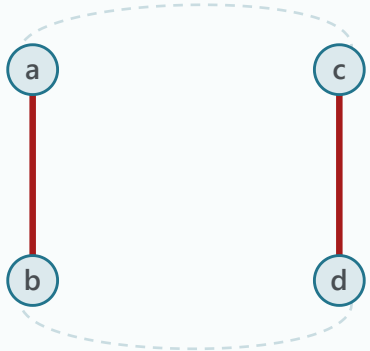


Sonuç: tur ≤ 2 · w(T) ≤ 2 · OPT. Burada tur = 287,7, optimum (kaba kuvvet, 360 tur) = 282,5 → oran 1,02 ≤ 2.

Yerel Arama: 2-opt Hamlesi



Önce: (a, b) ve (c, d) kenarları çaprazlanıyor.

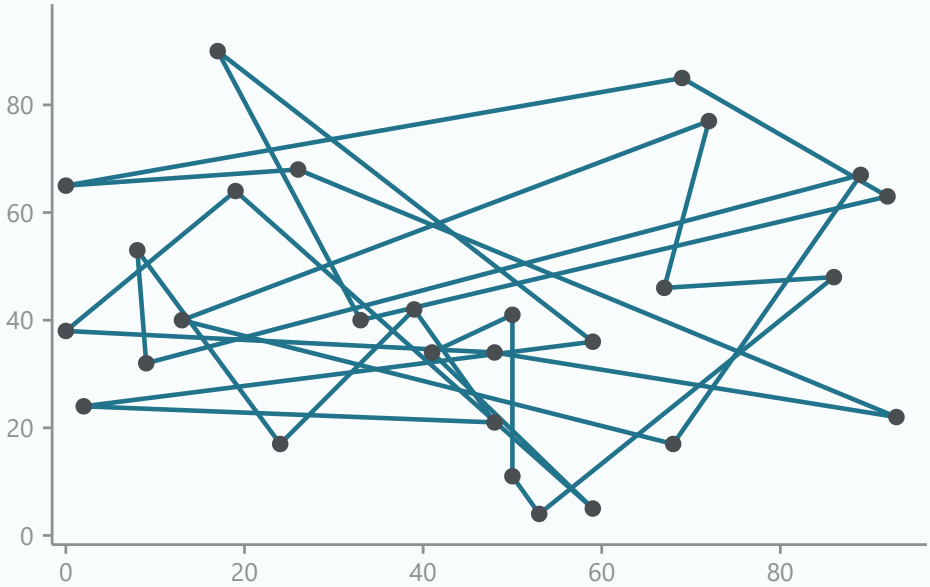


Sonra: (a, c) ve (b, d); aradaki yol ters çevrildi.

- Turdan iki kenar çıkarılır, uçlar **çapraz** bağlanır: b … c arasındaki alt yol **ters çevrilir**.
- Kazanç $\Delta = d(a,c) + d(b,d) - d(a,b) - d(c,d) < 0$ ise hamle yapılır. Öklid düzleminde her **kesişme** böyle kaldırılabilir.
- İyileştiren hamle kalmayınca tur **2-opt yerel optimumudur** (genel optimum olması gerekmez).

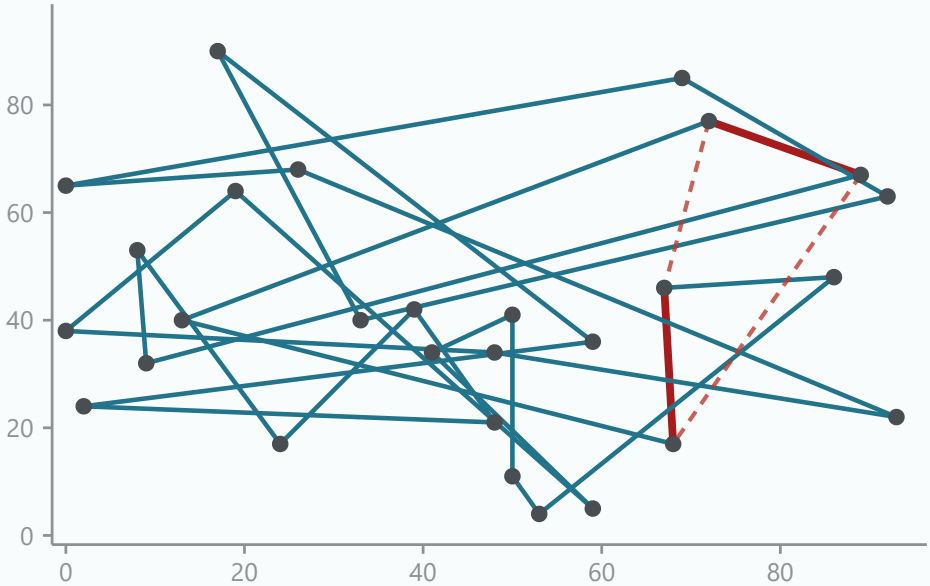
YEREL ARAMA · ADIM ADIM

2-opt ile Turu İyileştirme



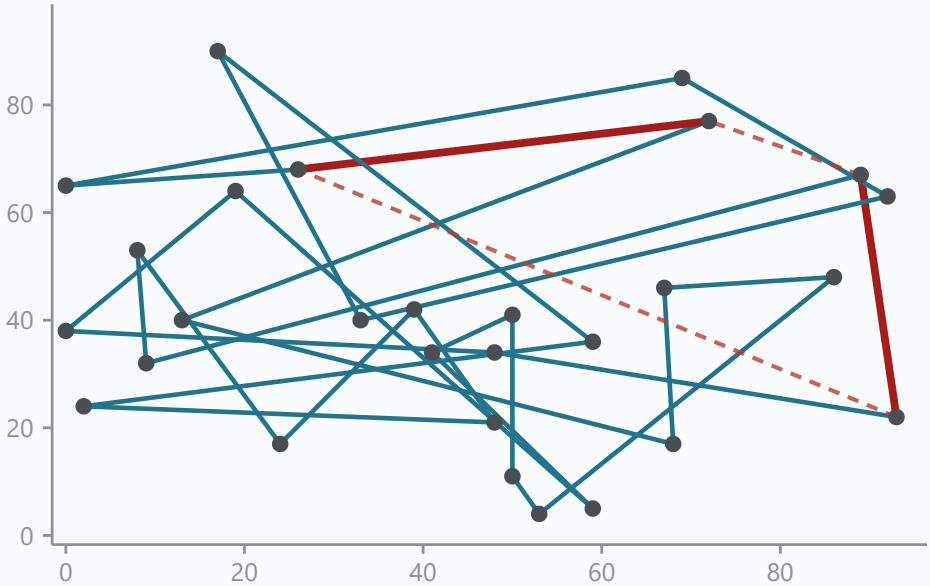
28 şehir, **rastgele** başlangıç turu: uzunluk **1269,8**. Kenarlar sürekli birbirini kesiyor.

2-opt ile Turu İyileştirme



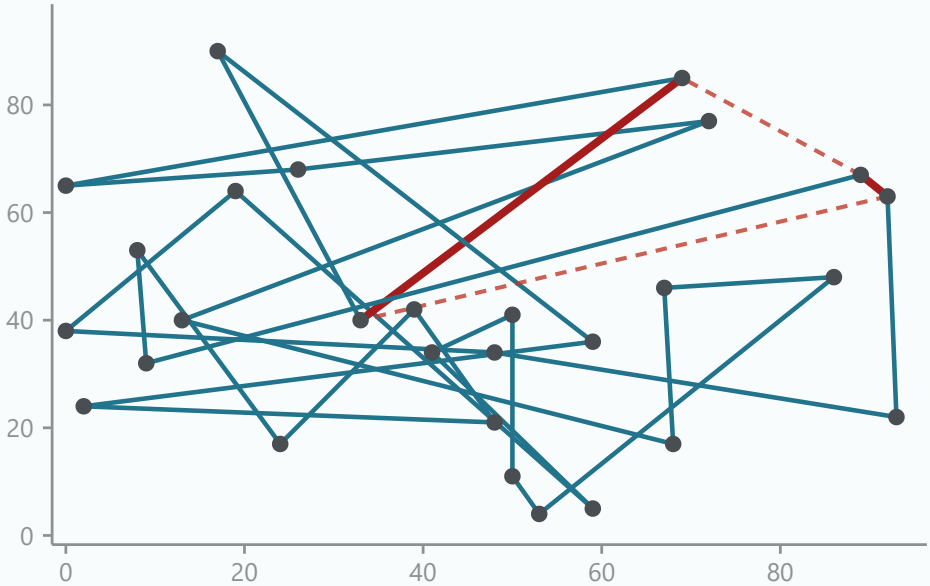
1. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −36,9 → uzunluk **1232,9**.

2-opt ile Turu İyileştirme



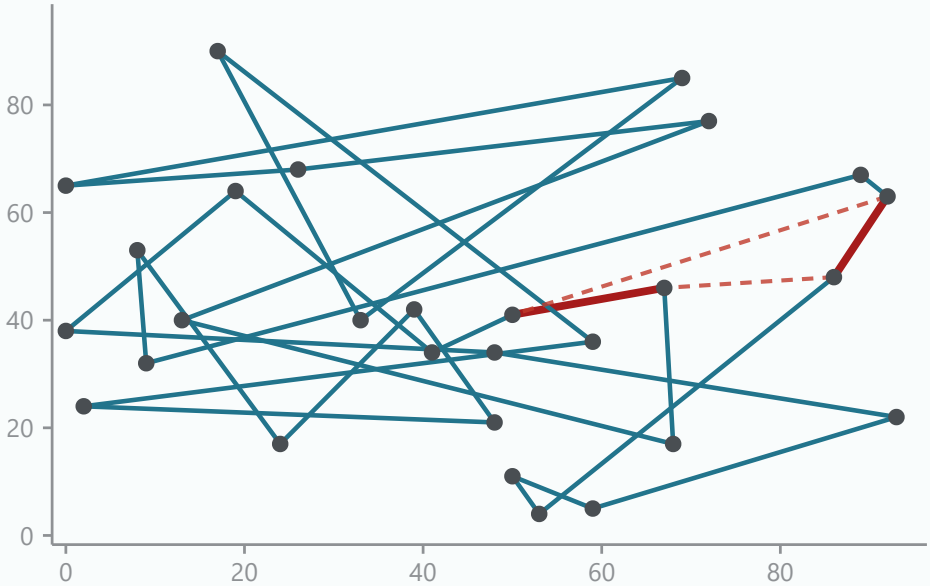
2. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −8,9 → uzunluk **1224,0**.

2-opt ile Turu İyileştirme



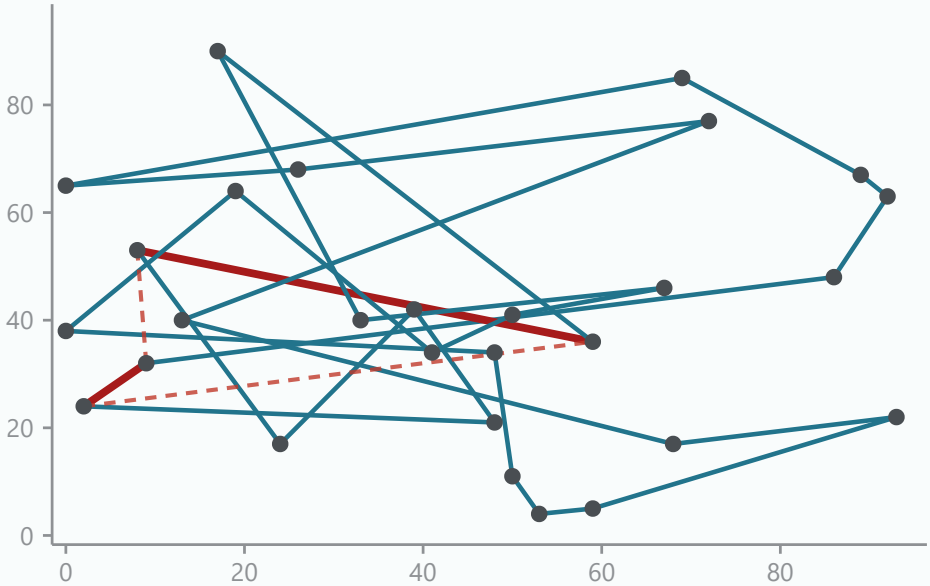
4. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −27,6 → uzunluk **1187,3**.

2-opt ile Turu İyileştirme



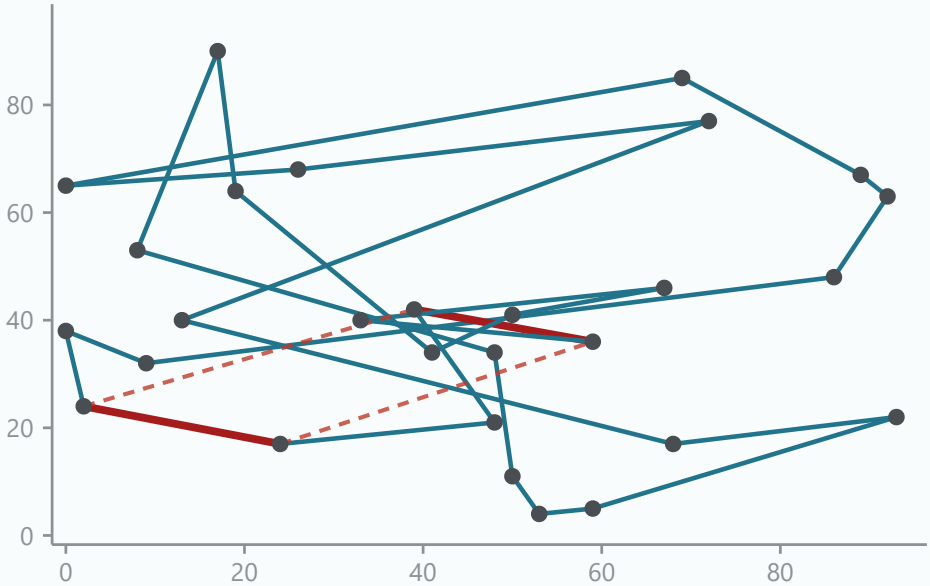
8. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −32,6 → uzunluk **1111,7**.

2-opt ile Turu İyileştirme



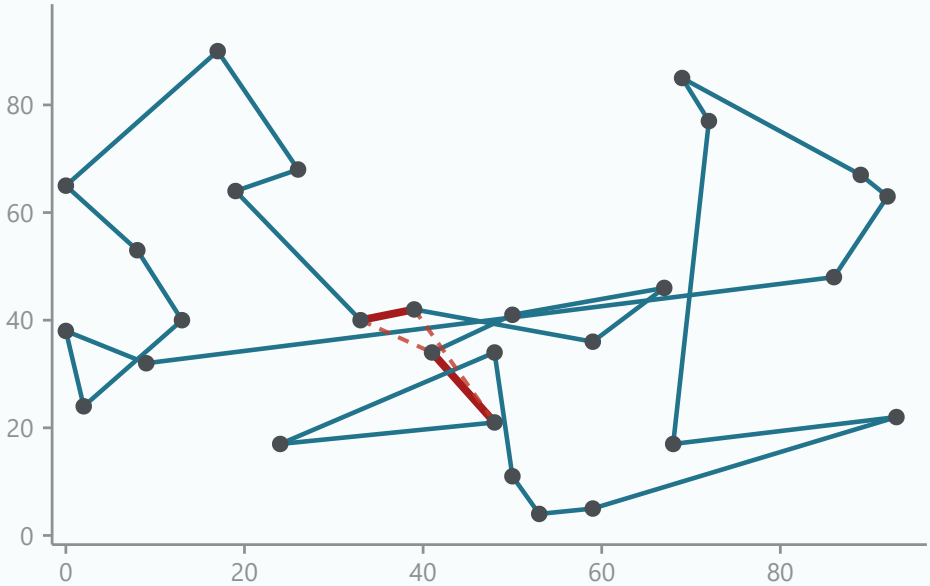
14. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −14,9 → uzunluk **1005,2**.

2-opt ile Turu İyileştirme



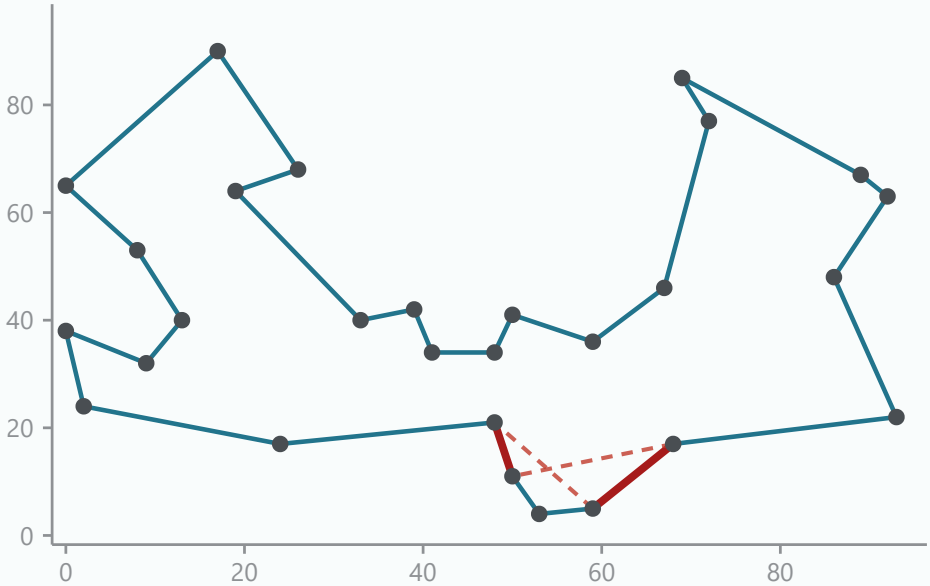
22. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −37,0 → uzunluk **852,8**.

2-opt ile Turu İyileştirme



32. iyileştiren hamleden sonra: (a,b), (c,d) çıkarıldı (kesikli), (a,c), (b,d) eklendi (kalın). Δ = −11,8 → uzunluk **595,9**.

2-opt ile Turu İyileştirme



48. ve son hamle: artık hiçbir 2-opt hamlesi turu kısaltmıyor → **yerel optimum**, uzunluk **430,8** (başlangıcın %34'ü). Kesişen kenar kalmadı.

2-opt Yerel Arama: Java

```java
void ikiOpt(int[] tur, double[][] d) {
  int n = tur.length;
  boolean iyilesti = true;
  while (iyilesti) {
    iyilesti = false;
    for (int i = 0; i < n - 1; i++)
      for (int j = i + 2; j < n; j++) {
        int a = tur[i], b = tur[i + 1];
        int c = tur[j], e = tur[(j + 1) % n];
        double fark = d[a][c] + d[b][e]
                    - d[a][b] - d[c][e];
        if (fark < -1e-9) {   // kısalıyor
          tersCevir(tur, i + 1, j);
          iyilesti = true;
        }
      }
  }
}
```

Bir tur tarama

$O(n^2)$ aday hamle; her hamle O(1)'de değerlendirilir, ters çevirme O(n).

Yerel optimum

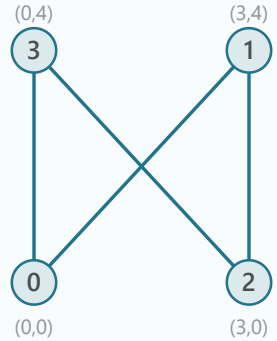
Döngü, hiçbir hamle kazanç sağlamayınca biter. Tur sayısı en kötü durumda üstel olabilir, pratikte azdır.

İyi başlangıç

Rastgele yerine en yakın komşu turuyla başlamak daha az hamle gerektirir. Genelleştirme: 3-opt, Lin–Kernighan.

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
▶    int n = tur.length;
▶    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
8          int a = tur[i], b = tur[i + 1];
9          int c = tur[j], e = tur[(j + 1) % n];
10         double fark = d[a][c] + d[b][e]
11                     - d[a][b] - d[c][e];
12         if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→1→2→3→0

uzunluk = 18

Başlangıç turu 0 → 1 → 2 → 3 → 0, uzunluk **18**. (0, 1) ve (2, 3) kenarları çaprazlanıyor.

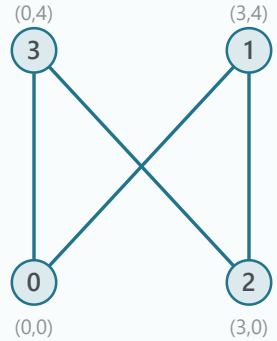
Adım 1 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = true` `i = –` `j = –` `fark = –` `tur = [0, 1, 2, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
▶    while (iyilesti) {
▶      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
8          int a = tur[i], b = tur[i + 1];
9          int c = tur[j], e = tur[(j + 1) % n];
10         double fark = d[a][c] + d[b][e]
11                     - d[a][b] - d[c][e];
12         if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→1→2→3→0

uzunluk = 18

tarama 1

Tarama 1 başlar: `iyilesti = false`. Bu taramada hiç hamle yapılmazsa döngü biter.

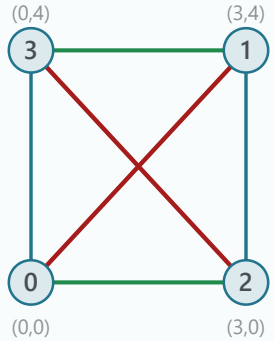
Adım 2 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = false` `i = –` `j = –` `fark = –` `tur = [0, 1, 2, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
▶          int a = tur[i], b = tur[i + 1];
▶          int c = tur[j], e = tur[(j + 1) % n];
▶          double fark = d[a][c] + d[b][e]
▶                      - d[a][b] - d[c][e];
▶          if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→1→2→3→0

uzunluk = 18

çıkan: 5 + 5

giren: 3 + 3

fark = −4

i = 0, j = 2: (0,1) ve (2,3) yerine (0,2) ve (1,3). fark = 3 + 3 − 5 − 5 = **−4** < 0 → tur kısalır.

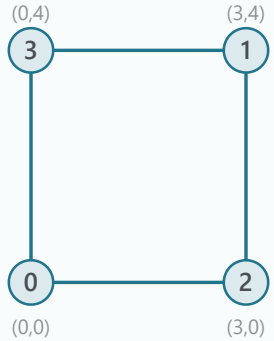
Adım 3 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = false` `i = 0` `j = 2` `a = 0` `b = 1` `c = 2` `e = 3` `fark = -4` `tur = [0, 1, 2, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
8          int a = tur[i], b = tur[i + 1];
9          int c = tur[j], e = tur[(j + 1) % n];
10         double fark = d[a][c] + d[b][e]
11                     - d[a][b] - d[c][e];
12         if (fark < -1e-9) {   // kısalıyor
▶            tersCevir(tur, i + 1, j);
▶            iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

tarama 1

`tersCevir(tur, 1, 2)` : aradaki yol ters çevrilir → `[0, 2, 1, 3]` , uzunluk **14**. `iyilesti = true` .

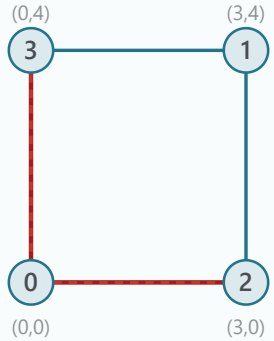
Adım 4 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = true` `i = 0` `j = 2` `a = 0` `b = 1` `c = 2` `e = 3` `fark = -4` `tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
▶          int a = tur[i], b = tur[i + 1];
▶          int c = tur[j], e = tur[(j + 1) % n];
▶          double fark = d[a][c] + d[b][e]
▶                      - d[a][b] - d[c][e];
▶          if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

çıkan: 3 + 4

giren: 4 + 3

fark = 0

i = 0, j = 3: fark = 4 + 3 − 3 − 4 = **0** (iki kenar komşu: e = a) → hamle yapılmaz.

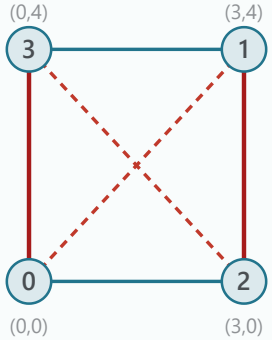
Adım 5 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = true` `i = 0` `j = 3` `a = 0` `b = 2` `c = 3` `e = 0` `fark = 0` `tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
▶          int a = tur[i], b = tur[i + 1];
▶          int c = tur[j], e = tur[(j + 1) % n];
▶          double fark = d[a][c] + d[b][e]
▶                      - d[a][b] - d[c][e];
▶          if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

çıkan: 4 + 4

giren: 5 + 5

fark = 2

i = 1, j = 3: fark = 5 + 5 − 4 − 4 = **2** → hamle yapılmaz.

Adım 6 / 11

DEĞİŞKENLER

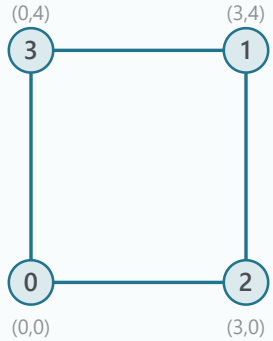
`n = 4` `iyilesti = true` `i = 1` `j = 3`

`a = 2` `b = 1` `c = 3` `e = 0` `fark = 2`

`tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
▶    while (iyilesti) {
▶      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
8          int a = tur[i], b = tur[i + 1];
9          int c = tur[j], e = tur[(j + 1) % n];
10         double fark = d[a][c] + d[b][e]
11                     - d[a][b] - d[c][e];
12         if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

tarama 2

Önceki taramada hamle yapıldı (`iyilesti = true`), tarama baştan yinelenir.

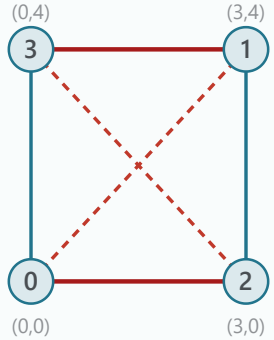
Adım 7 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = false` `i = –` `j = –` `fark = –` `tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
▶          int a = tur[i], b = tur[i + 1];
▶          int c = tur[j], e = tur[(j + 1) % n];
▶          double fark = d[a][c] + d[b][e]
▶                      - d[a][b] - d[c][e];
▶          if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

çıkan: 3 + 3

giren: 5 + 5

fark = 4

i = 0, j = 2: fark = 5 + 5 − 3 − 3 = **4** → hamle yapılmaz.

Adım 8 / 11

DEĞİŞKENLER

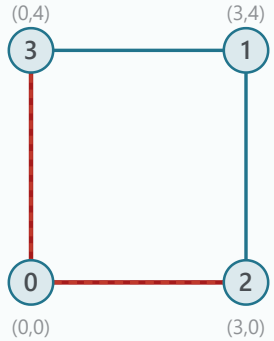
`n = 4` `iyilesti = false` `i = 0` `j = 2`

`a = 0` `b = 2` `c = 1` `e = 3` `fark = 4`

`tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
▶          int a = tur[i], b = tur[i + 1];
▶          int c = tur[j], e = tur[(j + 1) % n];
▶          double fark = d[a][c] + d[b][e]
▶                      - d[a][b] - d[c][e];
▶          if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

çıkan: 3 + 4

giren: 4 + 3

fark = 0

i = 0, j = 3: fark = 4 + 3 − 3 − 4 = **0** (iki kenar komşu: e = a) → hamle yapılmaz.

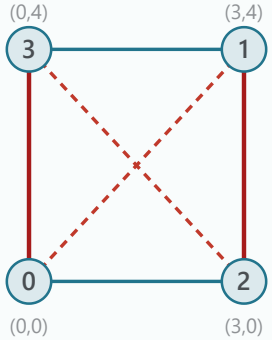
Adım 9 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = false` `i = 0` `j = 3` `a = 0` `b = 2` `c = 3` `e = 0` `fark = 0` `tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
4    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
▶          int a = tur[i], b = tur[i + 1];
▶          int c = tur[j], e = tur[(j + 1) % n];
▶          double fark = d[a][c] + d[b][e]
▶                      - d[a][b] - d[c][e];
▶          if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

çıkan: 4 + 4

giren: 5 + 5

fark = 2

i = 1, j = 3: fark = 5 + 5 − 4 − 4 = **2** → hamle yapılmaz.

Adım 10 / 11

DEĞİŞKENLER

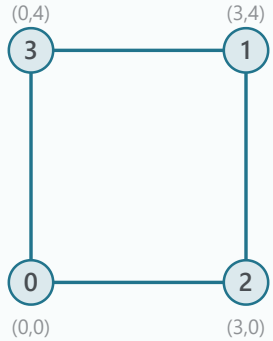
`n = 4` `iyilesti = false` `i = 1` `j = 3`

`a = 2` `b = 1` `c = 3` `e = 0` `fark = 2`

`tur = [0, 2, 1, 3]`

2-opt: Kodu Adım Adım İzle

```
1  void ikiOpt(int[] tur, double[][] d) {
2    int n = tur.length;
3    boolean iyilesti = true;
▶    while (iyilesti) {
5      iyilesti = false;
6      for (int i = 0; i < n - 1; i++)
7        for (int j = i + 2; j < n; j++) {
8          int a = tur[i], b = tur[i + 1];
9          int c = tur[j], e = tur[(j + 1) % n];
10         double fark = d[a][c] + d[b][e]
11                     - d[a][b] - d[c][e];
12         if (fark < -1e-9) {   // kısalıyor
13           tersCevir(tur, i + 1, j);
14           iyilesti = true;
15         }
16       }
17   }
18 }
```



tur: 0→2→1→3→0

uzunluk = 14

yerel optimum

Tüm tarama boyunca kısalma bulunmadı: döngü biter. Tur `[0, 2, 1, 3]`, uzunluk 14: 2-opt yerel optimumu (bu 4 noktada aynı zamanda optimum).

Adım 11 / 11

DEĞİŞKENLER

`n = 4` `iyilesti = false` `i = –` `j = –` `fark = –` `tur = [0, 2, 1, 3]`

Yerel Optimumdan Kaçış: Metasezgiseller

Benzetimli tavlama
(Simulated annealing)
Kötüleştiren bir hamle de $e^{-\Delta/T}$ olasılıkla kabul edilir; "sıcaklık" T yavaşça düşürülür. Yerel optimumdan çıkabilir.

Genetik algoritma
Turlardan oluşan bir **popülasyon**; seçilim, çaprazlama (crossover) ve mutasyonla nesiller boyunca iyileştirilir.

Karınca kolonisi
(Ant colony optimization)
Karıncalar kısa turların kenarlarına daha çok **feromon** bırakır; sonraki karıncalar feromonu yoğun kenarları tercih eder.

- Üçü de TSP, araç rotalama, iş çizelgeleme ve atama problemlerinde yaygındır.
- Hiçbiri yaklaşım oranı **garanti etmez**; sonuç başlangıç çözümüne ve parametrelere duyarlıdır. Tipik ayarlar: $T_0 = 100$, her adımda $T \leftarrow 0{,}997 \cdot T$; 100 bireylik popülasyon, rastgele kesim noktalı çaprazlama, 0,05 mutasyon olasılığı.



Yalnızca iyileştiren hamleleri kabul eden yerel arama soldaki çukurda kalır. Benzetimli tavlama yüksek T'de tepeyi aşabilir; T düştükçe hamleler seçicileşir.

İş Çizelgeleme (Job Scheduling)

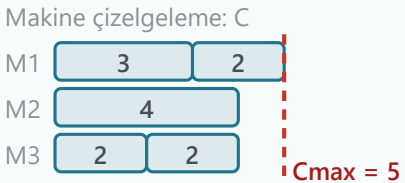
- **En iyileme problemi:** belirli kaynaklarla (işçiler, makineler, işlemciler) işlerin **en verimli** şekilde planlanması.
- **Amaç:** işlerin tamamlanma süresini en aza indirmek ya da belirli ölçütlere (kâr, gecikme, son tarih) göre en iyilemek.

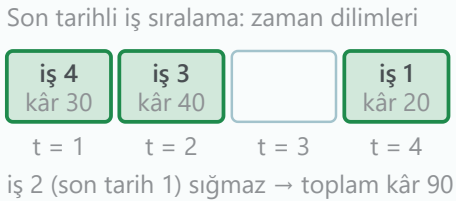
Makine çizelgeleme (en büyük yük)

n iş, m özdeş makine; iş j'nin süresi p_j. Son işin bittiği an $\mathbf{C_{max}}$ (en büyük yük, makespan) en küçük olsun. m ≥ 2 için NP-zor.

Son tarihli iş sıralama

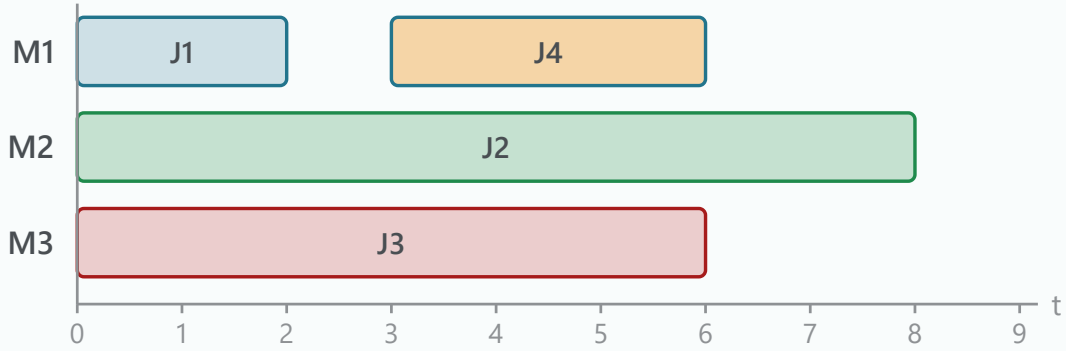
Her iş 1 birim sürer, bir **kârı** ve **son tarihi** (deadline) vardır. Son tarihinden önce biten işlerin toplam kârı en büyük olsun.





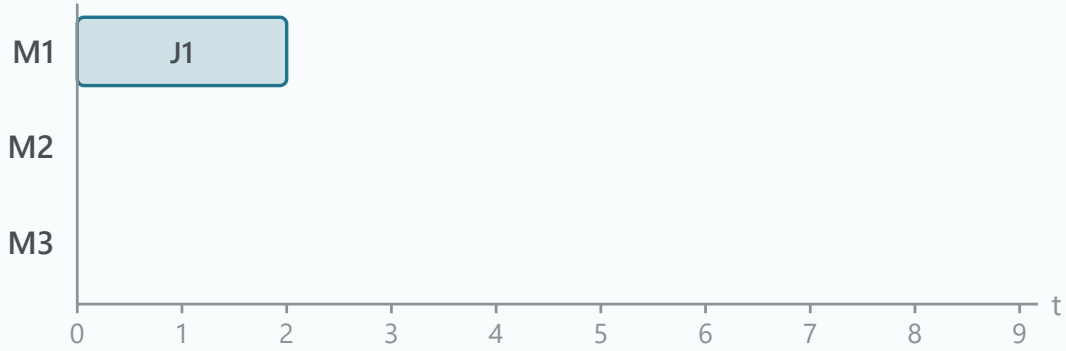
İlk problem için açgözlü **yaklaşım** algoritmaları (liste çizelgeleme, LPT) vardır; ikincisi açgözlü bir algoritmayla **optimal** çözülür.

Makinelere İş Atama: Liste Çizelgeleme



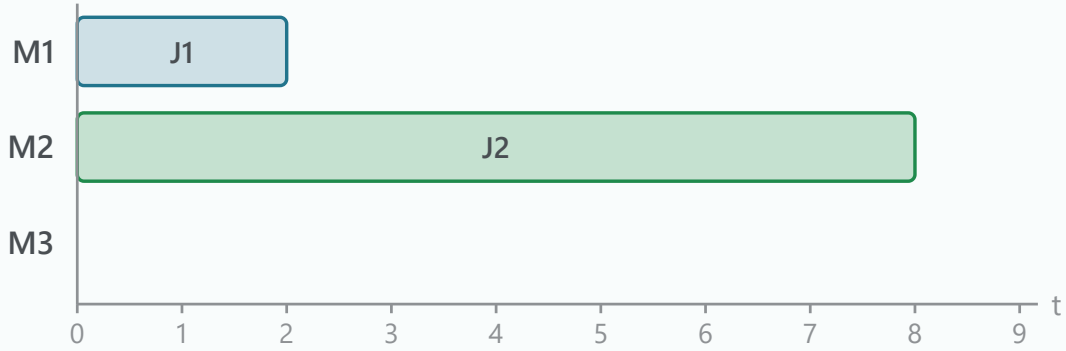
Üç özdeş makine M1–M3. İşler sırayla gelir: J1 (2), J2 (8), J3 (6) t = 0'da hazır; J4 (3) t = 3'te gelir. **Liste çizelgeleme** (Graham): sıradaki iş, **ilk boşalan** makineye atanır.

Makinelere İş Atama: Liste Çizelgeleme



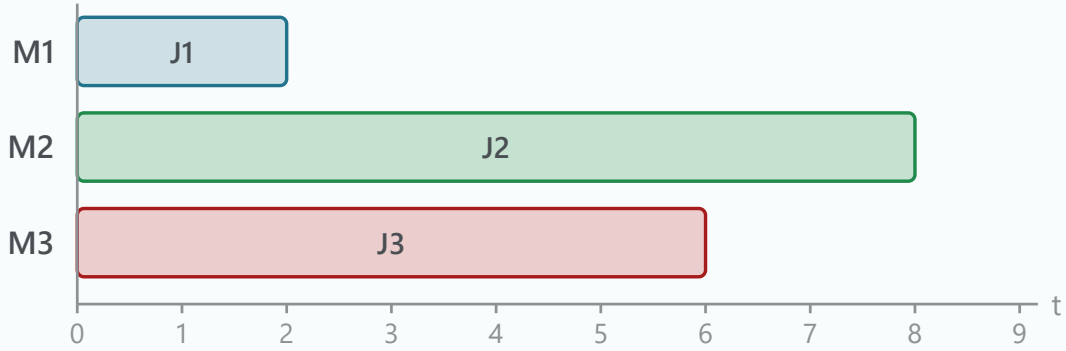
J1 (süre 2) → **M1**, [0, 2] aralığında çalışır (tüm makineler boş; ilk makine seçilir).

Makinelere İş Atama: Liste Çizelgeleme



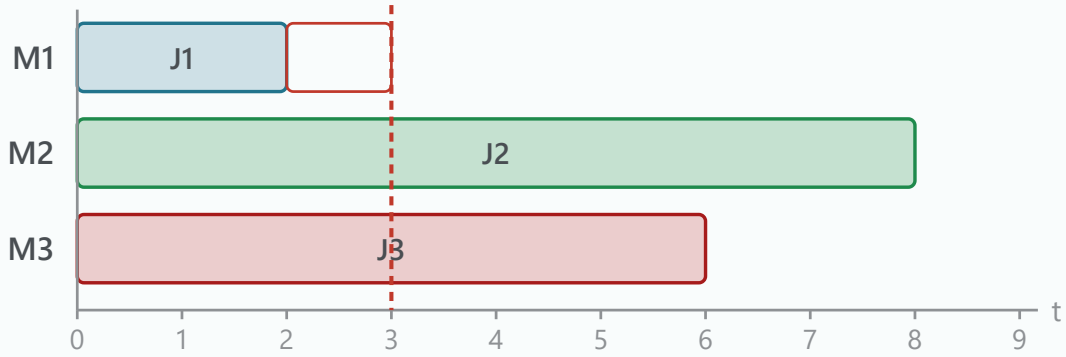
J2 (süre 8) → **M2**, [0, 8] aralığında çalışır (M1 t = 2 anına kadar meşgul).

Makinelere İş Atama: Liste Çizelgeleme



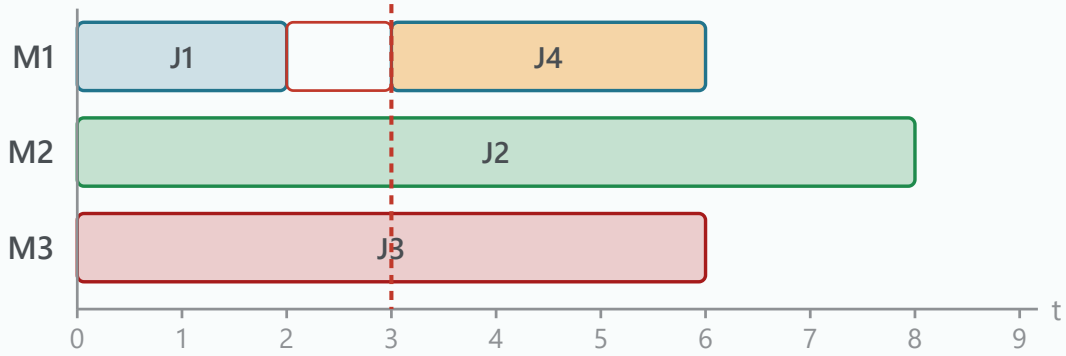
J3 (süre 6) → **M3**, [0, 6] aralığında çalışır (M1 t = 2 anına kadar meşgul; M2 t = 8 anına kadar meşgul).

Makinelere İş Atama: Liste Çizelgeleme



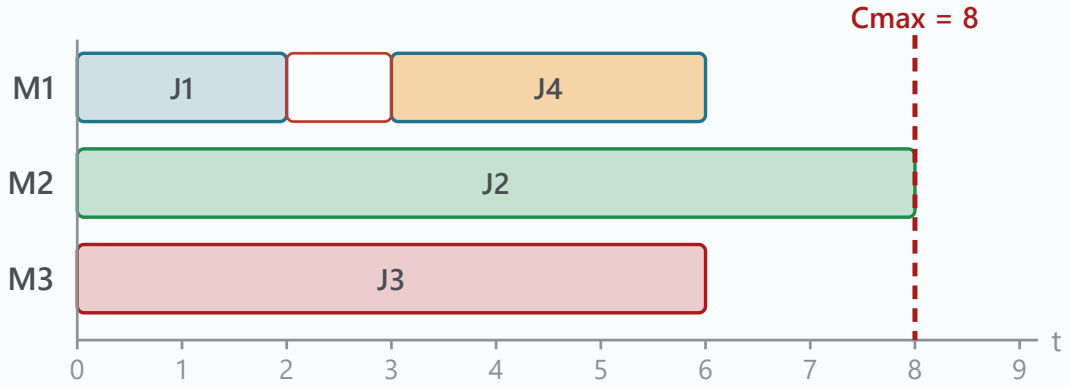
t = 2: M1 boşaldı ama bekleyen iş yok; makine t = 3'e kadar **boşta** kalır (taralı alan).

Makinelere İş Atama: Liste Çizelgeleme



J4 (süre 3) → **M1**, [3, 6] aralığında çalışır (M2 t = 8 anına kadar meşgul; M3 t = 6 anına kadar meşgul).

Makinelere İş Atama: Liste Çizelgeleme



Tüm işler atandı: $\mathbf{C_{max} = 8}$ (M2). Alt sınır: OPT ≥ en uzun iş = 8 → bu çizelge **optimaldir**. Genelde ise liste çizelgeleme yalnızca (2 − 1/m) oranını garanti eder.

Liste Çizelgeleme: (2 − 1/m)-Yaklaşım

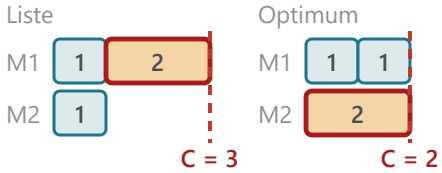
İki alt sınır: $OPT \geq \Sigma p_j / m$ (iş yükü paylaştırılmalı) ve $OPT \geq \max p_j$.

1. En son biten iş j, s_j anında başlamış olsun.
2. s_j'den önce **tüm makineler meşguldü** (yoksa j daha erken başlardı) → $s_j \leq (\Sigma p - p_j) / m$.
3. $C_{max} = s_j + p_j \leq \Sigma p / m + (1 - 1/m) \cdot p_j \leq OPT + (1 - 1/m) \cdot OPT$.

$C_{max} \leq (2 - 1/m) \cdot OPT$; süre $O(n \log m)$ (en erken boşalan makine bir heap'te tutulur).

Sınır sıkıdır

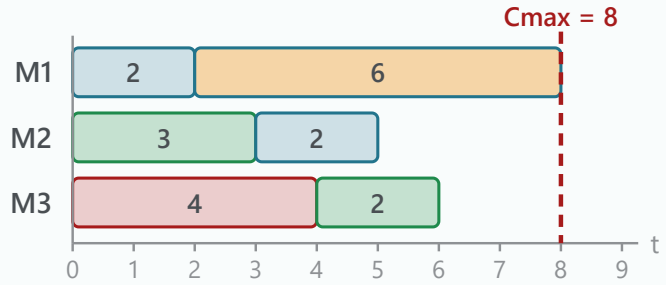
m = 2, işler sırasıyla 1, 1, 2:
• Liste: M1 = 1 + 2 = 3, M2 = 1 → $C_{max} = 3$
• Optimum: {1, 1} ve {2} → 2
Oran 3/2 = 2 − 1/2.



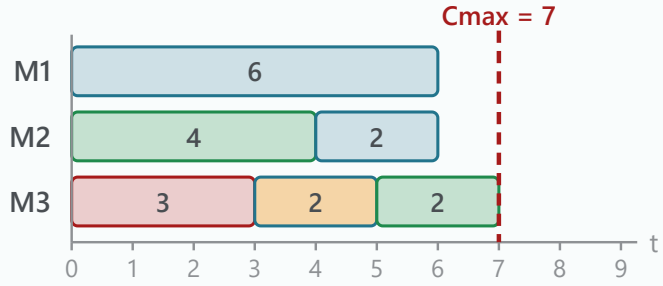
Genelde m(m − 1) adet 1 birimlik iş ve sonda tek bir m birimlik iş oranı 2 − 1/m'ye ulaştırır.

En Uzun İş Önce (LPT): 4/3-Yaklaşım

LPT (Longest Processing Time first): işler **süreye göre azalan** sıralanır, sonra liste çizelgeleme uygulanır. İşler: 2, 3, 4, 6, 2, 2 ve m = 3.



Verilen sıra: $C_{max} = 8$

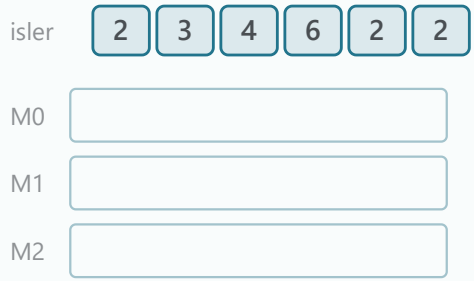


LPT sırası 6, 4, 3, 2, 2, 2: $C_{max} = 7 = OPT$ ($\geq \lceil 19/3 \rceil = 7$)

- Uzun işler başa alınınca sona kalan kısa işler dengesizliği düzeltir: $C_{max} \leq (4/3 - 1/(3m)) \cdot OPT$ (Graham, 1969). Süre O(n log n).
- Sıkı örnek (m = 3): 5, 5, 4, 4, 3, 3, 3 → LPT 11, optimum 9 ({5, 4}, {5, 4}, {3, 3, 3}).

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
5      int k = 0;              // en az yüklü
6      for (int i = 1; i < m; i++)
7        if (yuk[i] < yuk[k]) k = i;
8      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



İşler 2, 3, 4, 6, 2, 2 ve m = 3 makine. Bu sırayla doğrudan liste çizelgeleme C_{max} = 8 verir.

Adım 1 / 10

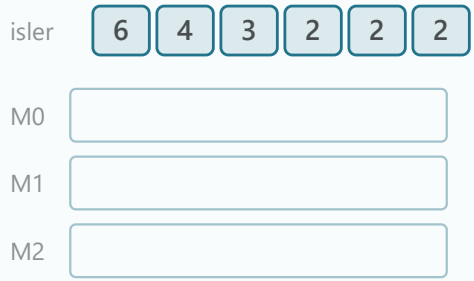
DEĞİŞKENLER

`isler = [2, 3, 4, 6, 2, 2]`

`m = 3`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
▶    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
5      int k = 0;              // en az yüklü
6      for (int i = 1; i < m; i++)
7        if (yuk[i] < yuk[k]) k = i;
8      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



LPT: işler süreye göre **azalan** sıralanır → 6, 4, 3, 2, 2, 2.

Adım 2 / 10

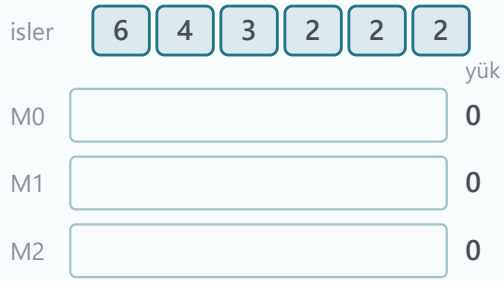
DEĞİŞKENLER

`isler = [6, 4, 3, 2, 2, 2]`

`m = 3`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
▶    int[] yuk = new int[m];
4    for (int p : isler) {
5      int k = 0;              // en az yüklü
6      for (int i = 1; i < m; i++)
7        if (yuk[i] < yuk[k]) k = i;
8      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



Üç makinenin yükü de 0. Her iş, o an **en az yüklü** makineye gidecek.

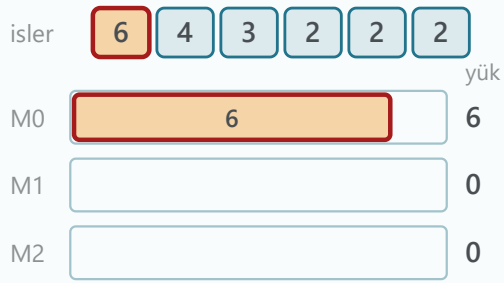
Adım 3 / 10

DEĞİŞKENLER

`yuk = [0, 0, 0]` `p = –` `k = –`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
▶      int k = 0;              // en az yüklü
▶      for (int i = 1; i < m; i++)
▶        if (yuk[i] < yuk[k]) k = i;
▶      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



p = 6: en az yüklü makine k = 0 (yük 0) (eşitlikte `<` ilk makinede kalır) → `yuk = [6, 0, 0]`.

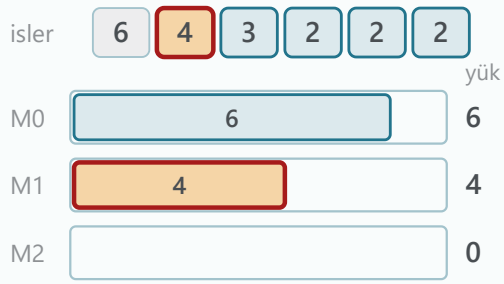
Adım 4 / 10

DEĞİŞKENLER

`yuk = [6, 0, 0]` `p = 6` `k = 0`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
▶      int k = 0;              // en az yüklü
▶      for (int i = 1; i < m; i++)
▶        if (yuk[i] < yuk[k]) k = i;
▶      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



p = 4: en az yüklü makine k = 1 (yük 0) → `yuk = [6, 4, 0]`.

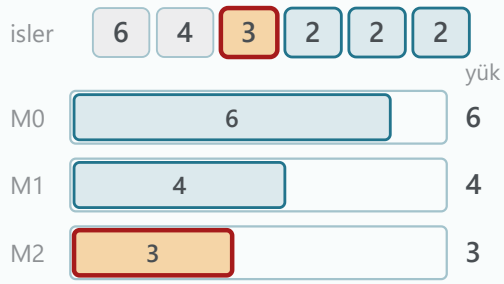
Adım 5 / 10

DEĞİŞKENLER

`yuk = [6, 4, 0]` `p = 4` `k = 1`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
▶      int k = 0;              // en az yüklü
▶      for (int i = 1; i < m; i++)
▶        if (yuk[i] < yuk[k]) k = i;
▶      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



p = 3: en az yüklü makine k = 2 (yük 0) → `yuk = [6, 4, 3]`.

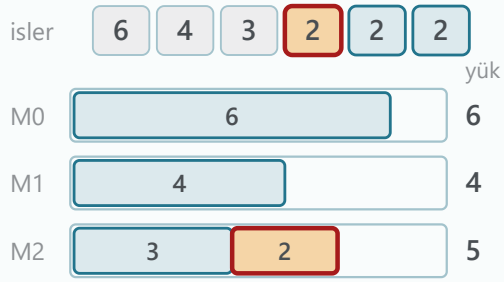
Adım 6 / 10

DEĞİŞKENLER

`yuk = [6, 4, 3]` `p = 3` `k = 2`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
▶      int k = 0;              // en az yüklü
▶      for (int i = 1; i < m; i++)
▶        if (yuk[i] < yuk[k]) k = i;
▶      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



p = 2: en az yüklü makine k = 2 (yük 3) → `yuk = [6, 4, 5]`.

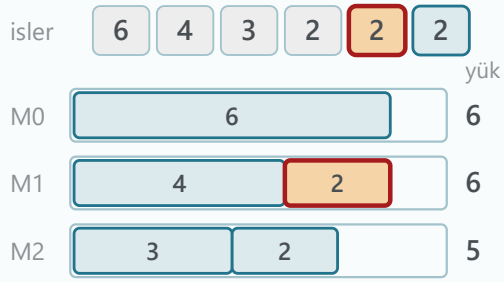
Adım 7 / 10

DEĞİŞKENLER

`yuk = [6, 4, 5]` `p = 2` `k = 2`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
▶      int k = 0;              // en az yüklü
▶      for (int i = 1; i < m; i++)
▶        if (yuk[i] < yuk[k]) k = i;
▶      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



p = 2: en az yüklü makine k = 1 (yük 4) → `yuk = [6, 6, 5]`.

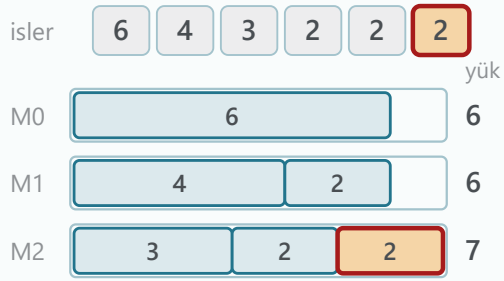
Adım 8 / 10

DEĞİŞKENLER

`yuk = [6, 6, 5]` `p = 2` `k = 1`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
▶      int k = 0;              // en az yüklü
▶      for (int i = 1; i < m; i++)
▶        if (yuk[i] < yuk[k]) k = i;
▶      yuk[k] += p;
9    }
10   return Arrays.stream(yuk).max().getAsInt();
11 }
```



p = 2: en az yüklü makine k = 2 (yük 5) → `yuk = [6, 6, 7]`.

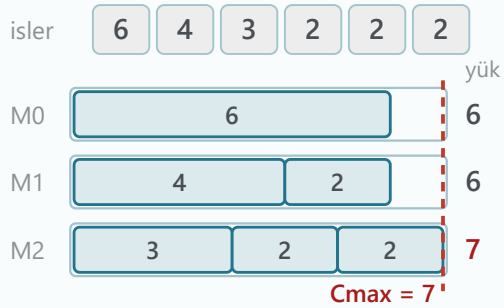
Adım 9 / 10

DEĞİŞKENLER

`yuk = [6, 6, 7]` `p = 2` `k = 2`

LPT Çizelgeleme: Kodu Adım Adım İzle

```
1  int lpt(Integer[] isler, int m) {
2    Arrays.sort(isler, Comparator.reverseOrder());
3    int[] yuk = new int[m];
4    for (int p : isler) {
5      int k = 0;              // en az yüklü
6      for (int i = 1; i < m; i++)
7        if (yuk[i] < yuk[k]) k = i;
8      yuk[k] += p;
9    }
▶    return Arrays.stream(yuk).max().getAsInt();
11 }
```



En büyük yük C_{max} = **7**. Alt sınır [19/3] = 7 olduğundan bu çizelge optimaldir.

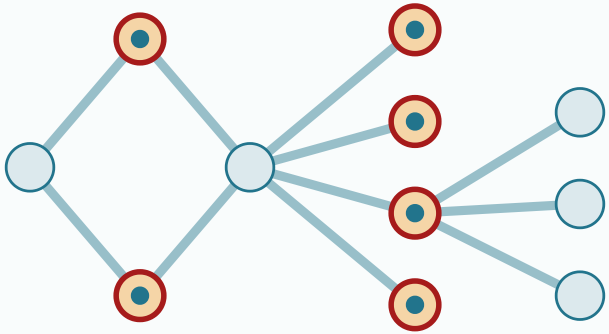
Adım 10 / 10

DEĞİŞKENLER

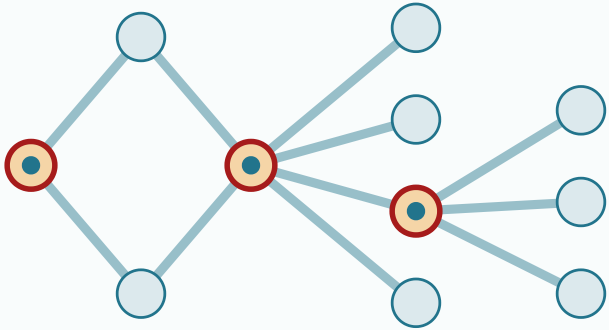
`yuk = [6, 6, 7]` `dönüş = 7`

Düğüm Kapsama (Vertex Cover)

- **Kapsama problemi:** bir çizgenin **tüm kenarlarını** kapsayacak en az sayıda düğümü seçmek. Bir kenar, uçlarından **en az biri** seçilmişse kapsanır.
- **Amaç:** tüm kenarları en az düğümle kapsayan $C \subseteq V$ kümesini bulmak. Karar sürümü ("k düğümlük kapsama var mı?") **NP-tamdır**.



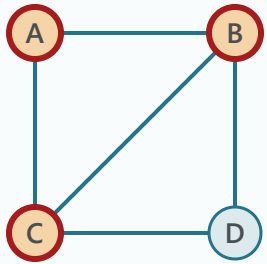
Geçerli bir kapsama, boyut 6 (her kenarı bir "bekçi" görüyor)



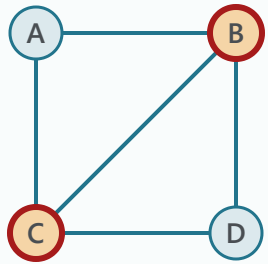
Aynı çizgede en küçük kapsama, boyut 3

Örnek: Dört Düğümlü Çizge

- **Düğümler:** A, B, C, D
- **Kenarlar:** (A, B), (A, C), (B, C), (B, D), (C, D)
- {A, B, C} bir düğüm kapsama kümesidir: her kenarın en az bir ucu kümede.
- Ama **en küçük** değildir: {B, C} de beş kenarın hepsini kapsar.
- Tek düğüm yetmez (B, (A, C)'yi kapsamaz) → optimum **2**.



{A, B, C}: geçerli, boyut 3



{B, C}: en küçük, boyut 2

Düğüm Kapsama: Çözüm Yaklaşımları

Tam çözüm

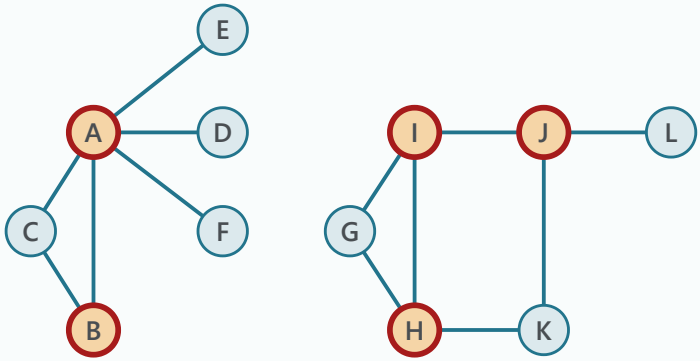
Tüm olası düğüm kümelerini kontrol etmek: 2^n alt küme × O(m) denetim.

Yaklaşık çözümler

Açgözlü algoritmalar (en yüksek dereceli düğüm), kenar seçmeli 2-yaklaşım algoritması.

Optimizasyon araçları

Doğrusal programlama (LP gevşetmesi + yuvarlama), dal sınır (branch and bound).



En küçük kapsama {A, B, H, I, J} (5 düğüm)

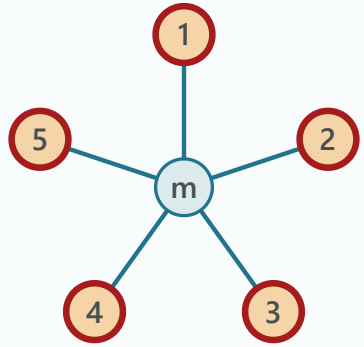
Düğüm Kapsama Algoritması

Saf fikir:

1. Başlangıçta kapsama kümesi boştur.
2. Kapsanmamış bir kenar seçilir; uçlarından biri kümeye eklenir.
3. Tüm kenarlar kapsanana kadar bu adım tekrarlanır.

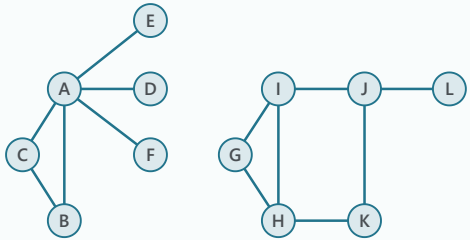
Uçlardan **yalnızca birini** eklemek garanti vermez. Doğru **2-yaklaşım**:

1. Kapsanmamış bir (u, v) kenarı seçilir; **u ve v'nin ikisi de** eklenir.
2. u ya da v'ye değen tüm kenarlar kapsanmış sayılır; kenar kalmayana kadar bu adımlar tekrarlanır.



Her kenarda yaprak ucu seçilirse 5 düğüm; optimum {m}, 1 düğüm. n yapraklı yıldızda oran n olur.

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



Kapsama kümesi boş. Kenarlar sırayla gezilecek: (A,B), (A,C), ..., (J,L).

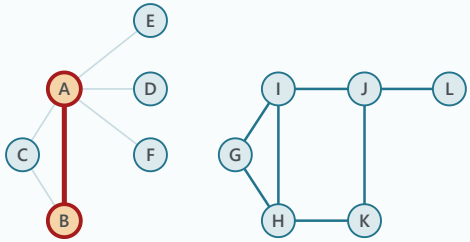
Adım 1 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
▶   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
4     char u = k[0], v = k[1];
5     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = –` `kapsama = {}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(A, B): iki uç da kapsamada değil → **A ve B** eklenir. Bu kenarlar bir **eşleştirme** (matching) oluşturur.

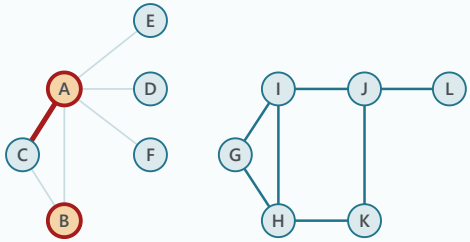
Adım 2 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
▶       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (A,B)` `kapsama = {A, B}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(A, C): A zaten kapsamada → kenar kapsanmış, atlanır.

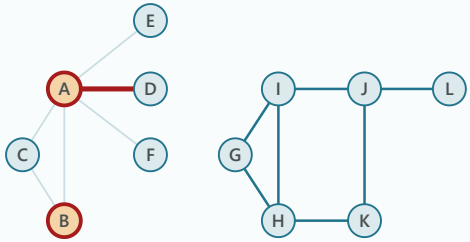
Adım 3 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (A,C)` `kapsama = {A, B}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(A, D): A zaten kapsamada → kenar kapsanmış, atlanır.

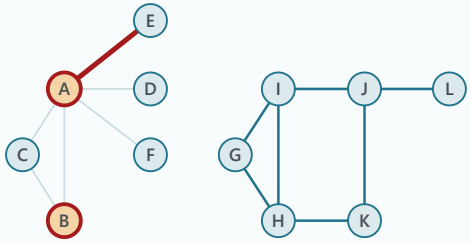
Adım 4 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (A,D)` `kapsama = {A, B}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(A, E): A zaten kapsamada → kenar kapsanmış, atlanır.

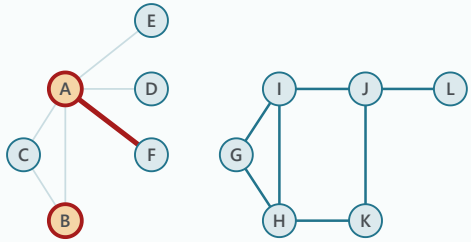
Adım 5 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (A,E)` `kapsama = {A, B}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(A, F): A zaten kapsamada → kenar kapsanmış, atlanır.

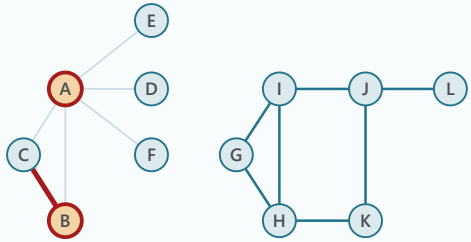
Adım 6 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = {A,F}` `kapsama = {A, B}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(B, C): B zaten kapsamada → kenar kapsanmış, atlanır.

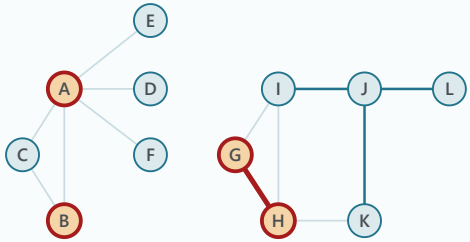
Adım 7 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (B,C)` `kapsama = {A, B}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(G, H): iki uç da kapsamada değil → **G ve H** eklenir. Bu kenarlar bir **eşleştirme** (matching) oluşturur.

Adım 8 / 15

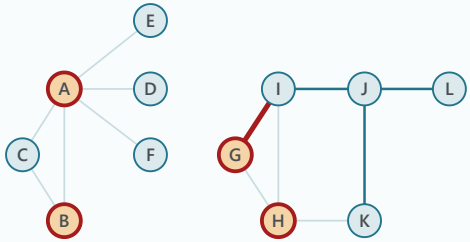
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
▶       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (G,H)`

`kapsama = {A, B, G, H}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(G, I): G zaten kapsamada → kenar kapsanmış, atlanır.

Adım 9 / 15

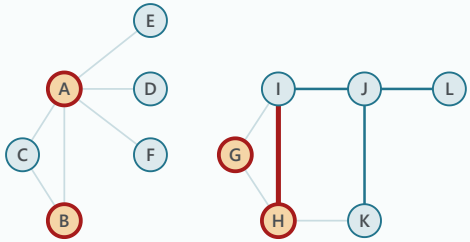
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (G,I)`

`kapsama = {A, B, G, H}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(H, I): H zaten kapsamada → kenar kapsanmış, atlanır.

Adım 10 / 15

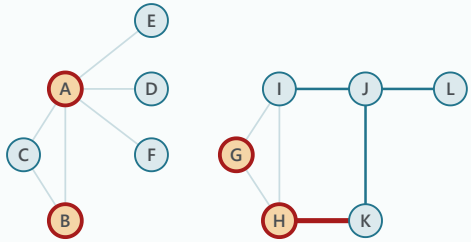
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (H,I)`

`kapsama = {A, B, G, H}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(H, K): H zaten kapsamada → kenar kapsanmış, atlanır.

Adım 11 / 15

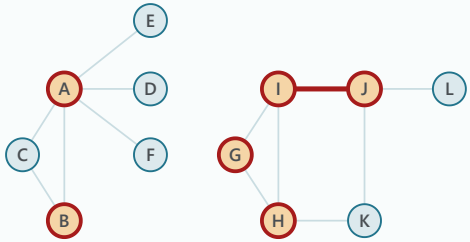
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (H,K)`

`kapsama = {A, B, G, H}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(I, J): iki uç da kapsamada değil → **I ve J** eklenir. Bu kenarlar bir **eşleştirme** (matching) oluşturur.

Adım 12 / 15

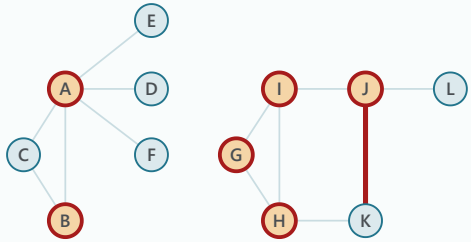
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
▶       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = (I,J)`

`kapsama = {A, B, G, H, I, J}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(J, K): J zaten kapsamada → kenar kapsanmış, atlanır.

Adım 13 / 15

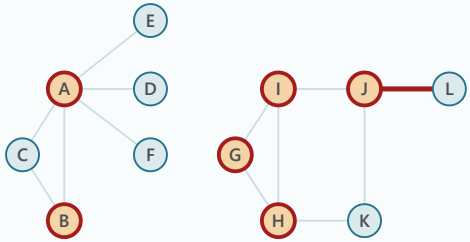
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = {J,K}`

`kapsama = {A, B, G, H, I, J}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



(J, L): J zaten kapsamada → kenar kapsanmış, atlanır.

Adım 14 / 15

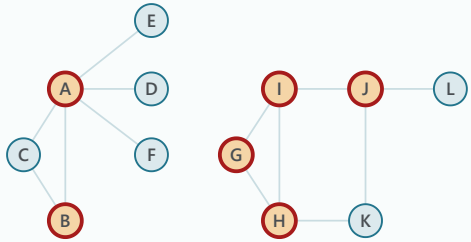
```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
▶     char u = k[0], v = k[1];
▶     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
8   return kapsama;
9 }
```

DEĞİŞKENLER

`k = {J,L}`

`kapsama = {A, B, G, H, I, J}`

Düğüm Kapsama 2-Yaklaşımı: Kodu Adım Adım İzle



Sonuç {A, B, G, H, I, J}: 6 düğüm. Optimum 5 ({A, B, H, I, J}) → oran 6/5 = 1,2 ≤ 2.

Adım 15 / 15

```
1 Set<Character> dugumKapsama(char[][] kenarlar) {
2   Set<Character> kapsama = new HashSet<>();
3   for (char[] k : kenarlar) {
4     char u = k[0], v = k[1];
5     if (!kapsama.contains(u) && !kapsama.contains(v)) {
6       kapsama.add(u); kapsama.add(v); }  // iki uç
7   }
▶   return kapsama;
9 }
```

DEĞİŞKENLER

`k = –`

`kapsama = {A, B, G, H, I, J}`

Neden En Fazla 2 × Optimum?

1. Algoritmanın seçtiği kenarlar M kümesi olsun. Seçilen her kenarın iki ucu da eklendiği için M'deki kenarların **ortak ucu yoktur** (M bir eşleştirmedir).
2. Her kapsama, M'deki her kenarı kapsamak için **farklı** bir düğüm içermek zorundadır → **OPT ≥ |M|**.
3. Algoritma tam olarak **|C| = 2 |M|** düğüm seçer → |C| ≤ 2 · OPT.

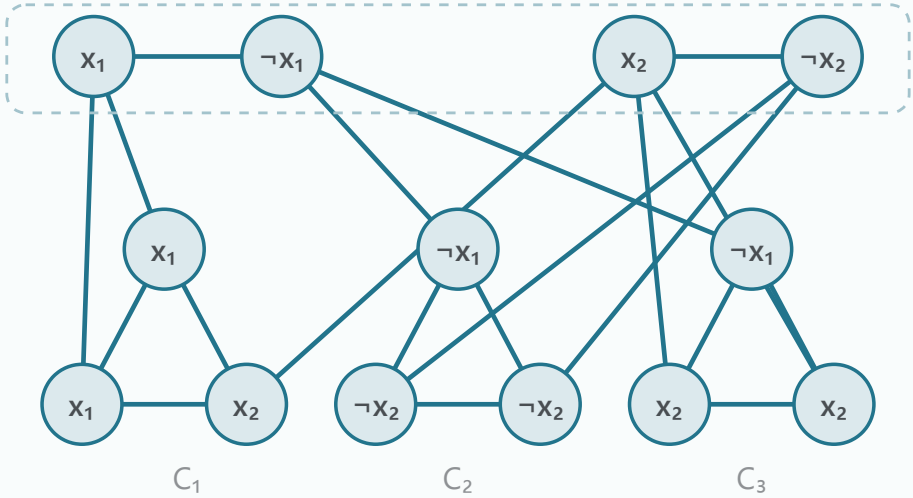
Süre
O(V + E): her kenar bir kez incelenir.

Açgözlü derece
En yüksek dereceli düğümü seçmek sezgiseldir ama oranı yalnızca O(log n)'dir.

LP yuvarlama
$x_u + x_v \geq 1$ gevşetmesi çözülür, $x_v \geq 1/2$ olanlar seçilir → yine 2-yaklaşım.

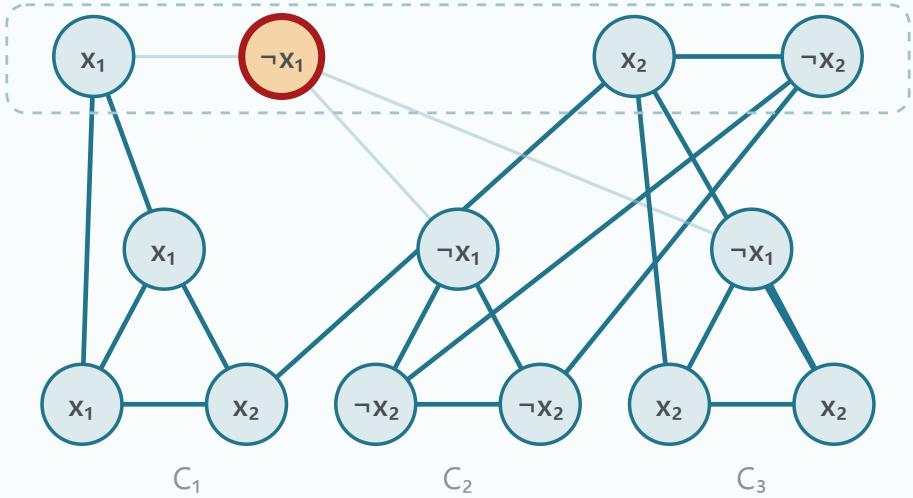
2'den belirgin biçimde iyi bir sabit oran bilinmiyor; 2 − ε oranının (Benzersiz Oyunlar Varsayımı altında) imkânsız olduğu gösterilmiştir.

NP-Tamlık: 3-SAT'tan Düğüm Kapsamaya İndirgeme



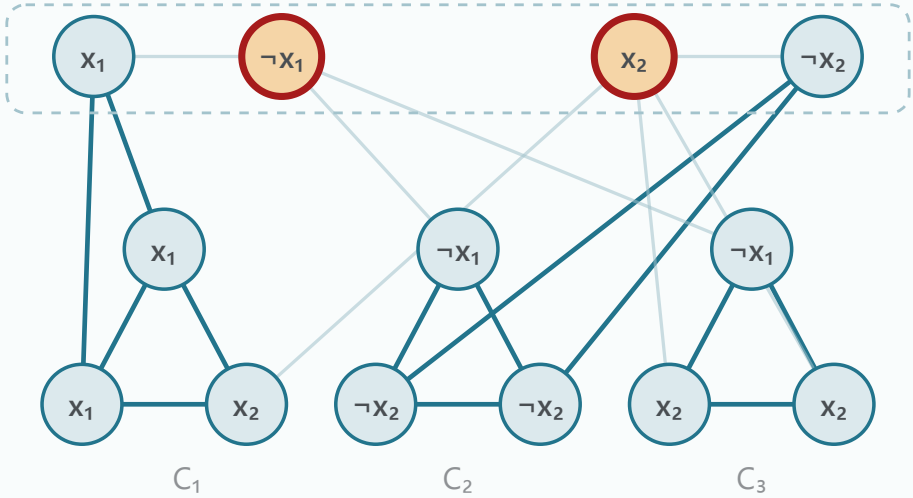
$\varphi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$. Her değişken için bir kenar (üst kutu), her tümce için bir üçgen; tümcedeki her literal aynı adlı değişken düğümüne bağlanır. $l = 2$ değişken, $m = 3$ tümce → $k = l + 2m =$ **8**.

NP-Tamlık: 3-SAT'tan Düğüm Kapsamaya İndirgeme



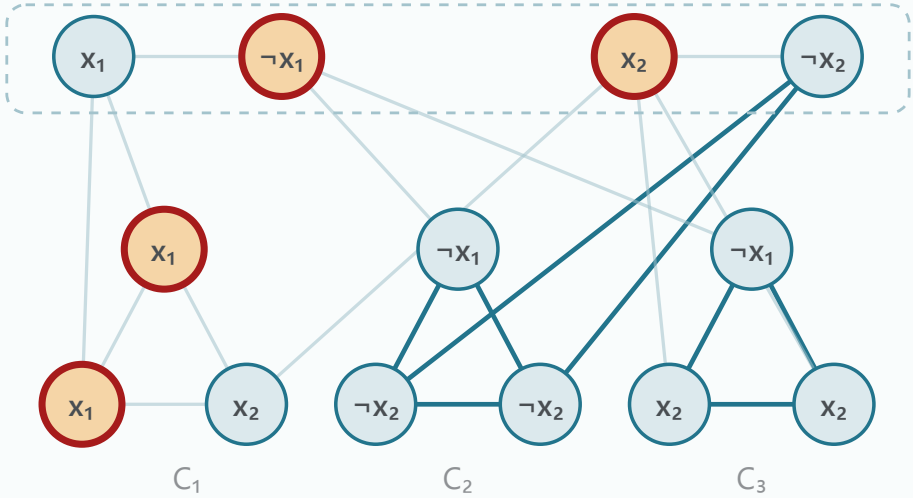
Atama x_1 = yanlış: değişken kenarından **$\neg x_1$** seçilir. Kapsanan kenarlar soluklaşır.

NP-Tamlık: 3-SAT'tan Düğüm Kapsamaya İndirgeme



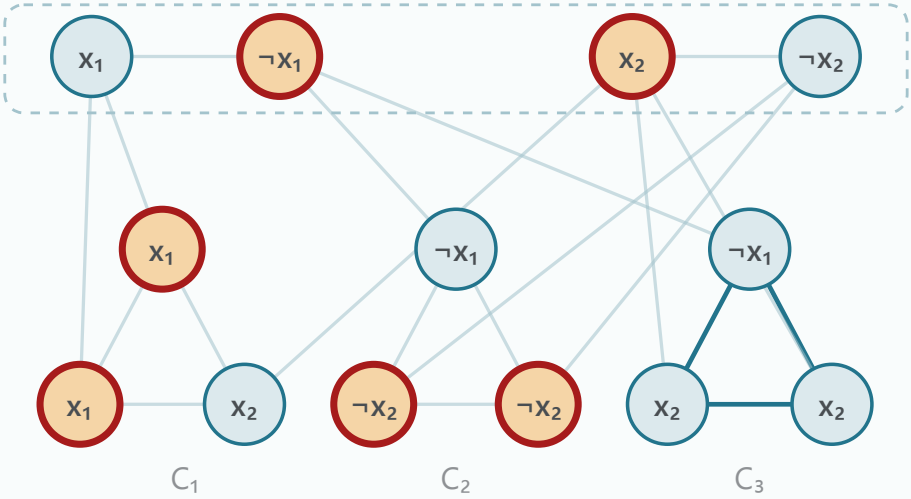
Atama x_2 = doğru: **x_2** seçilir. Her değişken kenarı tam olarak bir düğümle kapsandı (l = 2 düğüm).

NP-Tamlık: 3-SAT'tan Düğüm Kapsamaya İndirgeme



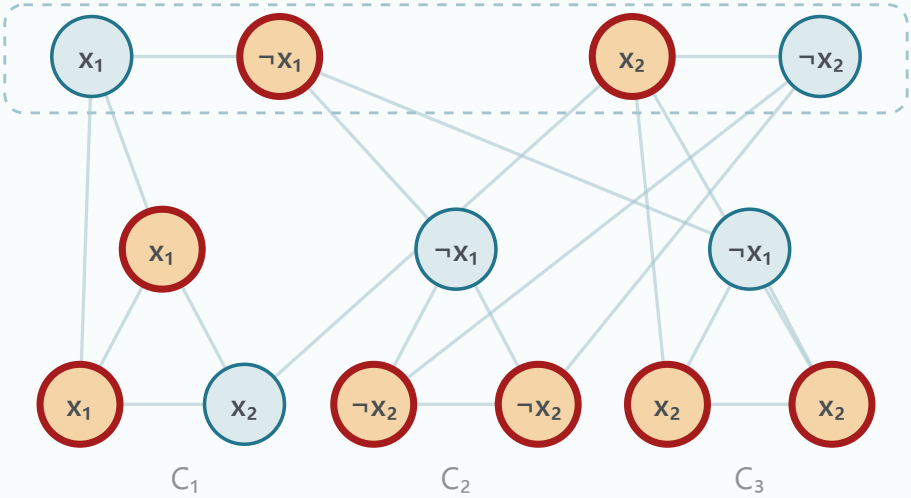
C_1'de doğru literal x_2 (bağlantı kenarı zaten kapsanmış) dışarıda bırakılır; diğer iki düğüm (x_1, x_1) eklenir.

NP-Tamlık: 3-SAT'tan Düğüm Kapsamaya İndirgeme



C_2'de doğru literal $\neg x_1$ dışarıda; $\neg x_2$ ve $\neg x_2$ düğümleri eklenir.

NP-Tamlık: 3-SAT'tan Düğüm Kapsamaya İndirgeme



C_3'te doğru literal $\neg x_1$ dışarıda; x_2 ve x_2 eklenir. **8 düğüm, 20 kenarın hepsi kapsandı**: φ sağlanabilir $\Leftrightarrow$ $k = l + 2m$ boyutlu kapsama var.

Sırt Çantası: Oran Sıralı Açgözlü Yaklaşım

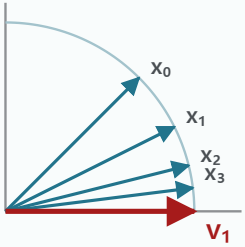
Eşya	Değer	Ağırlık	Oran	Kapasite 10
6	20	7	2,86	✓ alındı (ağırlık 7)
3	8	4	2,00	✗ sığmaz
7	15	8	1,88	✗ sığmaz
2	5	3	1,67	✓ alındı (ağırlık 10)
5	10	6	1,67	✗ sığmaz
1	3	2	1,50	✗ sığmaz
4	6	5	1,20	✗ sığmaz

Açgözlü toplam değer 25; DP ile optimum da 25 (eşya 2 + 6).

- Eşyalar **değer/ağırlık** oranına göre azalan sıralanır; sığan alınır. O(n log n).
- Kesirli sırt çantasında bu kural **optimaldir**; 0/1 sürümde ise **garanti yoktur**.
- Kötü örnek: W = 10, (değer 2, ağırlık 1) ve (değer 10, ağırlık 10). Açgözlü 2 alır, optimum 10: oran 5 = W/2; W büyüdükçe sınırsız büyür.
- **İyileştirilmiş kural:** max(açgözlü değer, sığan en değerli tek eşya) → **2-yaklaşım** (optimumun en az yarısı).
- Daha iyisi: değerleri ölçekleyip DP uygulayan **FPTAS**, $(1 - \varepsilon) \cdot OPT$, $O(n^3/\varepsilon)$.

Power Iteration (Kuvvet Yinelemesi) Algoritması

- Bir matrisin **en büyük** (mutlak değeri en büyük, baskın) **özdeğerini** λ_1 ve buna ait bir **özvektörü** v_1 yaklaşık olarak hesaplayan **yinelemeli** bir yöntemdir.
- Tüm özdeğerleri kesin olarak hesaplamak ($O(n^3)$) yerine, yalnızca baskın özdeğer ve özvektörün **yaklaşık bir tahminini** verir.
- Her adım yalnızca bir **matris–vektör çarpımıdır** ($O(n^2)$, seyrek matriste O(sıfır olmayan öğe)): **büyük boyutlu** matrislerde etkilidir.



$A = [[2, 0], [0, 1]]$, $x_0 = (1, 1)$: $x_k \propto (2^k, 1)$. Her çarpımda v_1 bileşeni 2 kat, öteki 1 kat büyür; normalleştirilmiş vektör adım adım $v_1 = (1, 0)$ yönüne döner (hata $\propto (1/2)^k$).

Burada iki kavram bir arada: sonuç kesin değer yerine bir **yaklaşımdır** (tahmin) ve yinelemeler gerçek değere **yakınsar** (converge). Google'ın PageRank'i bu yöntemin en ünlü uygulamasıdır.

Power Iteration: Adımlar ve Neden Çalışır?

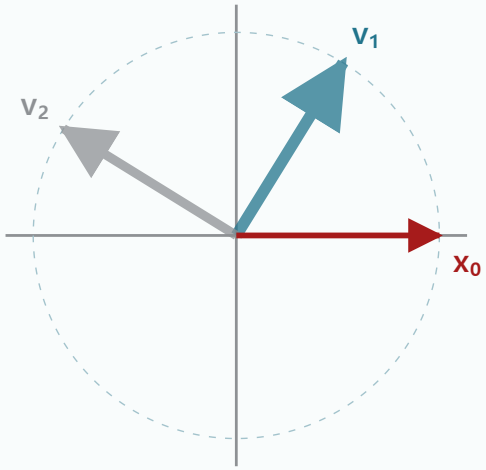
1. Bir A matrisi belirlenir ve **rastgele** (sıfır olmayan) bir başlangıç vektörü x_0 seçilir.
2. Vektör matrisle çarpılarak yeni vektör elde edilir: $y = A \cdot x_k$.
3. Yeni vektörün **normu** (uzunluğu) 1'e normalleştirilir: $x_{k+1} = y / \|y\|$.
4. Bu işlem istenen sayıda ya da değişim küçülene kadar tekrarlanır.
5. $x_k \to v_1$; özdeğer tahmini **Rayleigh bölümü** $\lambda \approx x_k^T A x_k$.

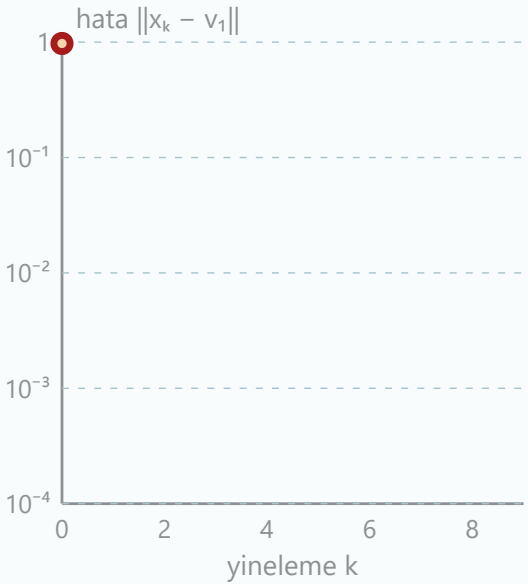
Neden? $x_0 = c_1v_1 + c_2v_2 + \dots$ ise

$$A^k x_0 = \lambda_1^k \left[c_1 v_1 + c_2 (\lambda_2/\lambda_1)^k v_2 + \dots \right]$$

- $|\lambda_2/\lambda_1| < 1$ olduğundan diğer bileşenler **geometrik** olarak söner.
- Hata her adımda $\approx |\lambda_2/\lambda_1|$ katına iner: oran 1'e yakınsa yakınsama yavaştır.
- Koşul: $c_1 \neq 0$ (x_0, v_1'e dik olmamalı) ve $|\lambda_1| > |\lambda_2|$.

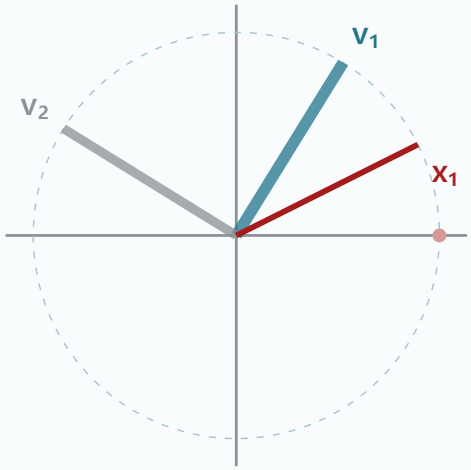
Power Iteration Adım Adım

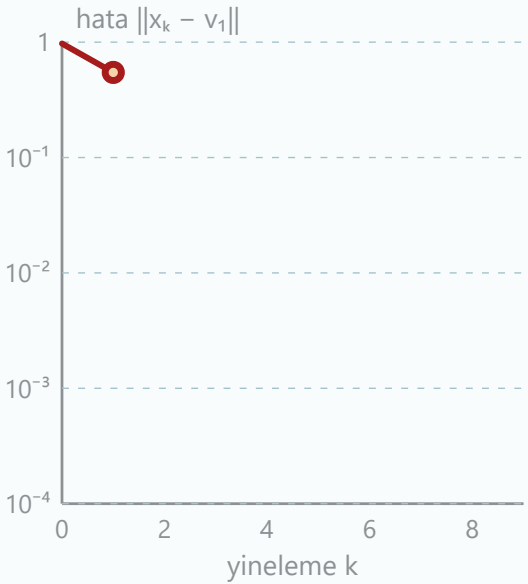




A = [[2, 1], [1, 3]], özdeğerler $\lambda_1 = (5 + \sqrt{5})/2 \approx 3{,}618$ ve $\lambda_2 \approx 1{,}382$. Başlangıç $x_0 = (1, 0)$; hata $\|x_0 - v_1\| = 0{,}974$.

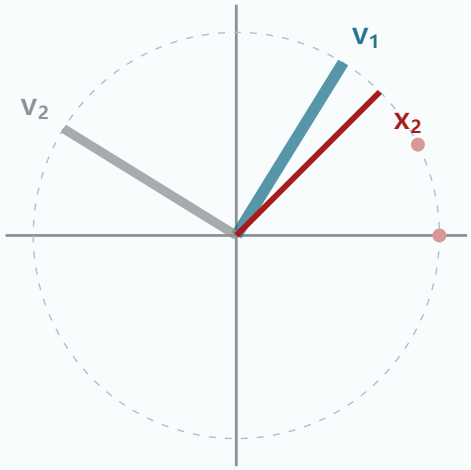
Power Iteration Adım Adım

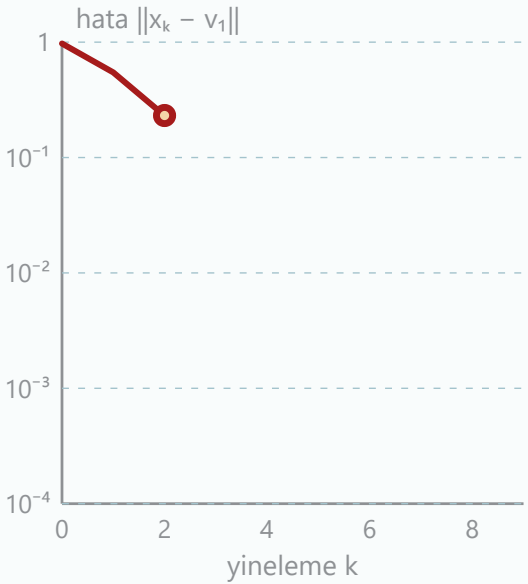




k = 1: A·x = (2,000, 1,000), normalleştirilince x = (0,894, 0,447); λ ≈ 3,000, hata 0,5465.

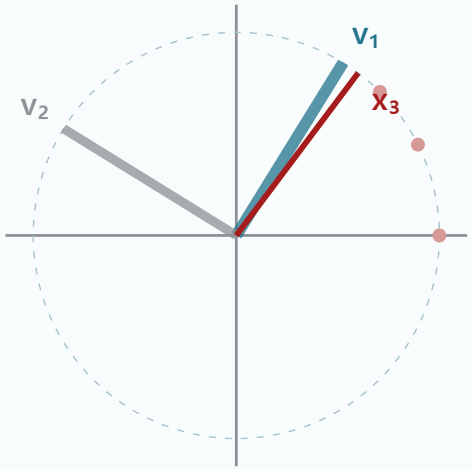
Power Iteration Adım Adım





k = 2: A·x = (2,236, 2,236), normalleştirilince x = (0,707, 0,707); λ ≈ 3,500, hata 0,2313.

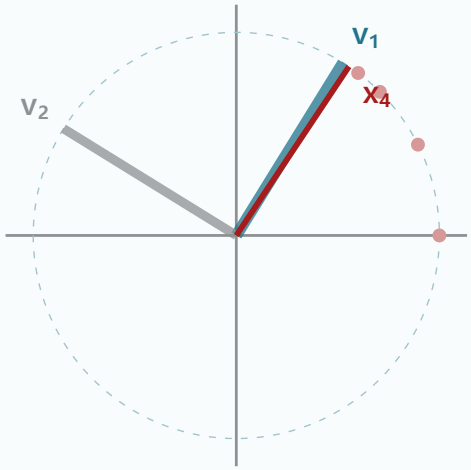
Power Iteration Adım Adım





k = 3: A·x = (2,121, 2,828), normalleştirilince x = (0,600, 0,800); λ ≈ 3,600, hata 0,0899.

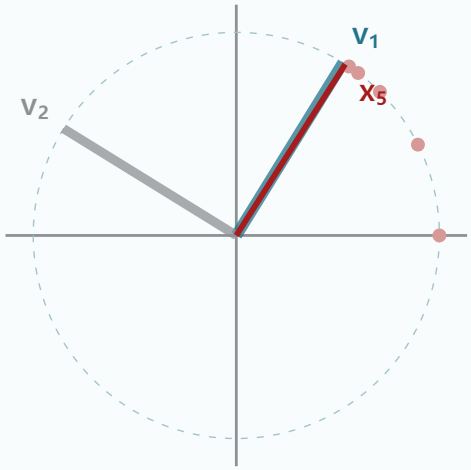
Power Iteration Adım Adım





k = 4: x = (0,5547, 0,8321); λ ≈ 3,61538; hata 3,4 · 10^{-2}. Her adımda ≈ λ_2/λ_1 ≈ 0,38 katına iner.

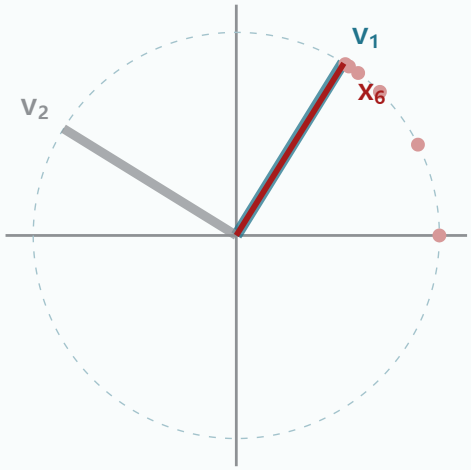
Power Iteration Adım Adım





k = 5: x = (0,5369, 0,8437); λ ≈ 3,61765; hata 1,3 · 10^{-2}. Her adımda ≈ λ_2/λ_1 ≈ 0,38 katına iner.

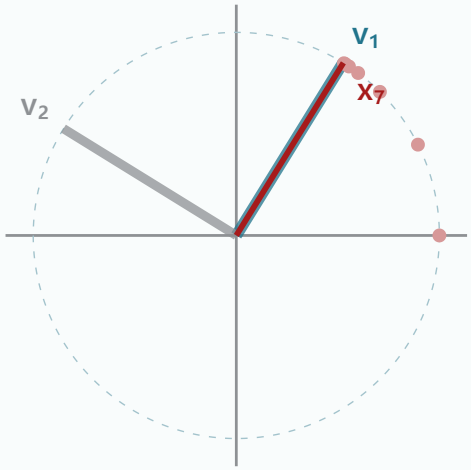
Power Iteration Adım Adım

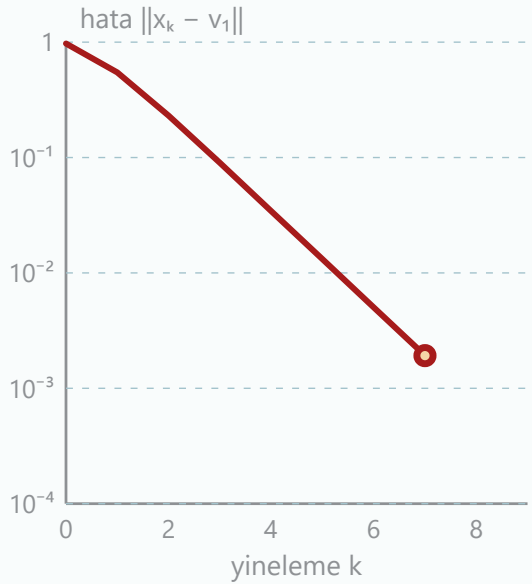




k = 6: x = (0,5300, 0,8480); λ ≈ 3,61798; hata 5,0 · 10^{-3}. Her adımda ≈ λ_2/λ_1 ≈ 0,38 katına iner.

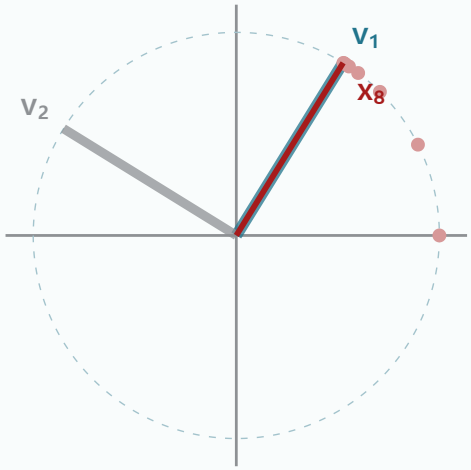
Power Iteration Adım Adım

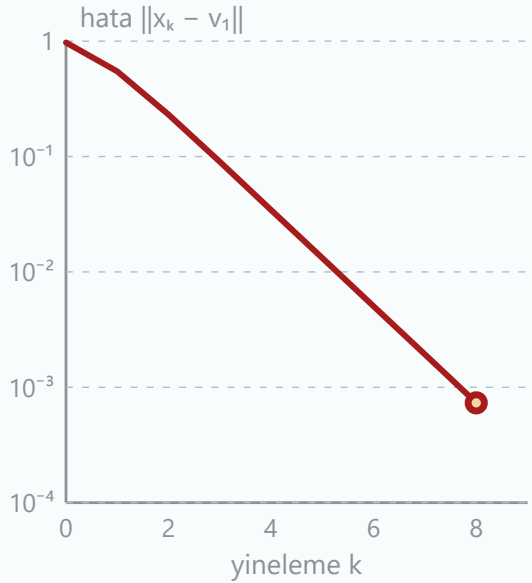




k = 7: x = (0,5274, 0,8496); λ ≈ 3,61803; hata 1,9 · 10^{-3}. Her adımda ≈ λ_2/λ_1 ≈ 0,38 katına iner.

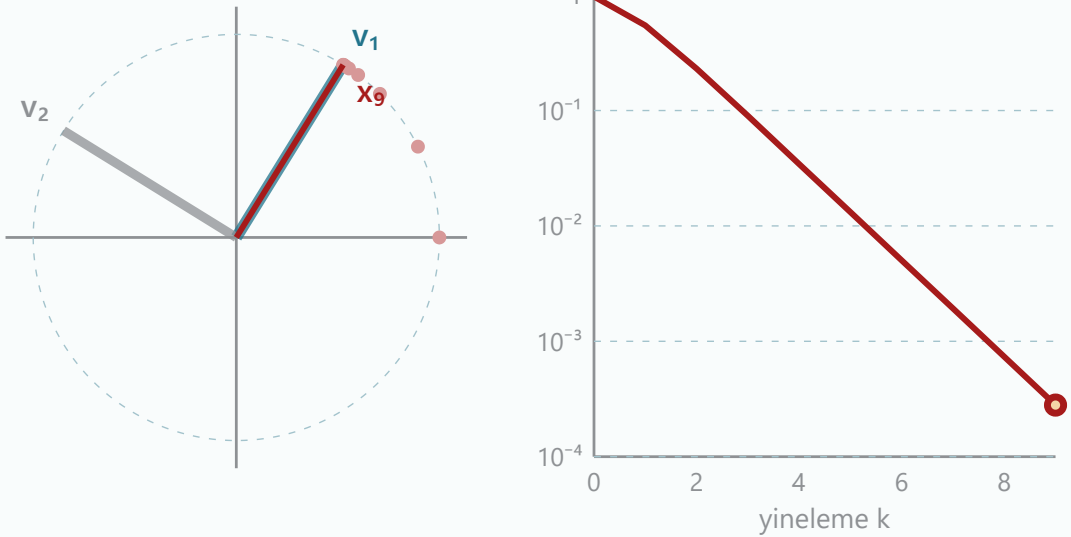
Power Iteration Adım Adım





k = 8: x = (0,5264, 0,8503); λ ≈ 3,61803; hata 7,3 · 10^{-4}. Her adımda ≈ λ_2/λ_1 ≈ 0,38 katına iner.

Power Iteration Adım Adım



k = 9: x ≈ (0,5260, 0,8505) ≈ v_1 = (0,5257; 0,8507), λ ≈ 3,618034; hata ≈ $2{,}8 \cdot 10^{-4}$. Hata grafiği log ölçekte bir doğru: **doğrusal (geometrik) yakınsama**.

Power Iteration: Java ve Bir 3 × 3 Örnek

```java
double[] kuvvet(double[][] a, int adim) {
  int n = a.length;
  double[] x = new double[n];
  Arrays.fill(x, 1.0);      // x₀ ≠ 0
  for (int k = 0; k < adim; k++) {
    double[] y = new double[n];
    for (int i = 0; i < n; i++)  // y = A·x
      for (int j = 0; j < n; j++)
        y[i] += a[i][j] * x[j];
    double norm = 0;
    for (double yi : y) norm += yi * yi;
    norm = Math.sqrt(norm);
    for (int i = 0; i < n; i++)  // ‖x‖ = 1
      x[i] = y[i] / norm;
  }
  return x;
}
```

Örnek matris (satırlar):
(0,511 0,32 0,12) · (0,211 0,62 0,12) · (0,311 0,12 0,16); x_0 = (1, 1, 1)

k	x_1	x_2	x_3	$\|A\cdot x\|$
1	0,6474	0,6474	0,4023	1,4690
2	0,6533	0,6533	0,3827	0,8974
3	0,6539	0,6539	0,3807	0,9005
4	0,6539	0,6539	0,3805	0,9008
5	0,6539	0,6539	0,3805	0,9008
6	0,6539	0,6539	0,3805	0,9008
8	0,6539	0,6539	0,3805	0,9008

$\|A\cdot x\| \to \lambda_1 = 0{,}9008$; $v_1 \propto (0{,}6539; 0{,}6539; 0{,}3805)$. $\lambda_2/\lambda_1 = 0{,}3/0{,}9008 \approx 0{,}33$ → hızlı yakınsama.

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
▶    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
7      for (int i = 0; i < n; i++)  // y = A·x
8        for (int j = 0; j < n; j++)
9          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

başlangıç A

				x		y
0,511	0,32	0,12		1,0000		
0,211	0,62	0,12	·	1,0000	=	
0,311	0,12	0,16		1,0000		

Başlangıç vektörü $x_0 = (1, 1, 1)$. v_1'e dik olmayan, sıfırdan farklı her vektör olur.

Adım 1 / 11

DEĞİŞKENLER

`k = –`

`x = [1.0000, 1.0000, 1.0000]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
       for (int i = 0; i < n; i++)  // y = A·x
         for (int j = 0; j < n; j++)
           y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 0

A				x		y
0,511	0,32	0,12		1,0000		0,9510
0,211	0,62	0,12	·	1,0000	=	
0,311	0,12	0,16		1,0000		

y[0] = A'nın 0. satırı · x = 0,511·1 + 0,32·1 + 0,12·1 = **0,9510**.

Adım 2 / 11

DEĞİŞKENLER

`k = 0` `i = 0` `y = [0.9510]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
▶      for (int i = 0; i < n; i++)  // y = A·x
▶        for (int j = 0; j < n; j++)
▶          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 0

A				x	y
0,511	0,32	0,12		1,0000	0,9510
0,211	0,62	0,12	·	1,0000	= 0,9510
0,311	0,12	0,16		1,0000	

y[1] = A'nın 1. satırı · x = 0,211·1 + 0,62·1 + 0,12·1 = **0,9510**.

Adım 3 / 11

DEĞİŞKENLER

`k = 0` `i = 1` `y = [0.9510, 0.9510]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
▶      for (int i = 0; i < n; i++)  // y = A·x
▶        for (int j = 0; j < n; j++)
▶          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 0

A				x		y
0,511	0,32	0,12		1,0000		0,9510
0,211	0,62	0,12	·	1,0000	=	0,9510
0,311	0,12	0,16		1,0000		0,5910

y[2] = A'nın 2. satırı · x = 0,311·1 + 0,12·1 + 0,16·1 = **0,5910**.

Adım 4 / 11

DEĞİŞKENLER

`k = 0` `i = 2`

`y = [0.9510, 0.9510, 0.5910]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
7      for (int i = 0; i < n; i++)  // y = A·x
8        for (int j = 0; j < n; j++)
9          y[i] += a[i][j] * x[j];
▶      double norm = 0;
▶      for (double yi : y) norm += yi * yi;
▶      norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 0

A				x		y
0,511	0,32	0,12		1,0000		0,9510
0,211	0,62	0,12	·	1,0000	=	0,9510
0,311	0,12	0,16		1,0000		0,5910

||y|| = 1,4690

$\|y\| = \sqrt{(0{,}9510^2 + 0{,}9510^2 + 0{,}5910^2)} =$ **1,4690**.

Adım 5 / 11

DEĞİŞKENLER

`k = 0` `norm = 1.4690`

`x = [1.0000, 1.0000, 1.0000]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
7      for (int i = 0; i < n; i++)  // y = A·x
8        for (int j = 0; j < n; j++)
9          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
▶      for (int i = 0; i < n; i++)  // ‖x‖ = 1
▶        x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 0

A				x		y
0,511	0,32	0,12		0,6474		0,9510
0,211	0,62	0,12	·	0,6474	=	0,9510
0,311	0,12	0,16		0,4023		0,5910

‖y‖ = 1,4690 **x ← y / ‖y‖**

x = y / ‖y‖ = (0,6474; 0,6474; 0,4023): uzunluğu 1 olan yeni yön; sayılar ne büyür ne söner.

Adım 6 / 11

DEĞİŞKENLER

`k = 0` `norm = 1.4690`

`x = [0.6474, 0.6474, 0.4023]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
▶      double[] y = new double[n];
▶      for (int i = 0; i < n; i++)  // y = A·x
▶        for (int j = 0; j < n; j++)
▶          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 1

A				x		y
0,511	0,32	0,12		0,6474		0,5862
0,211	0,62	0,12	·	0,6474	=	0,5862
0,311	0,12	0,16		0,4023		0,3434

k = 1: y = A·x = (0,5862; 0,5862; 0,3434). Bileşenlerin oranı neredeyse aynı kaldı.

Adım 7 / 11

DEĞİŞKENLER

- k = 1
- x = [0.6474, 0.6474, 0.4023]
- y = [0.5862, 0.5862, 0.3434]

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
7      for (int i = 0; i < n; i++)  // y = A·x
8        for (int j = 0; j < n; j++)
9          y[i] += a[i][j] * x[j];
▶      double norm = 0;
▶      for (double yi : y) norm += yi * yi;
▶      norm = Math.sqrt(norm);
▶      for (int i = 0; i < n; i++)  // ‖x‖ = 1
▶        x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 1

A				x		y
0,511	0,32	0,12		0,6533		0,5862
0,211	0,62	0,12	·	0,6533	=	0,5862
0,311	0,12	0,16		0,3827		0,3434

$\|y\| = 0{,}8974$ $\quad x \leftarrow y / \|y\|$

‖y‖ = 0,8974 → x = (0,6533; 0,6533; 0,3827). Yön çok az değişti.

DEĞİŞKENLER

`k = 1` `norm = 0.8974`

`x = [0.6533, 0.6533, 0.3827]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
▶      double[] y = new double[n];
▶      for (int i = 0; i < n; i++)  // y = A·x
▶        for (int j = 0; j < n; j++)
▶          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 2

A				x		y
0,511	0,32	0,12		0,6533		0,5888
0,211	0,62	0,12	·	0,6533	=	0,5888
0,311	0,12	0,16		0,3827		0,3428

k = 2: y = A·x = (0,5888; 0,5888; 0,3428). Bileşenlerin oranı neredeyse aynı kaldı.

Adım 9 / 11

DEĞİŞKENLER

- k = **2**
- x = **[0.6533, 0.6533, 0.3827]**
- y = **[0.5888, 0.5888, 0.3428]**

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
7      for (int i = 0; i < n; i++)  // y = A·x
8        for (int j = 0; j < n; j++)
9          y[i] += a[i][j] * x[j];
▶      double norm = 0;
▶      for (double yi : y) norm += yi * yi;
▶      norm = Math.sqrt(norm);
▶      for (int i = 0; i < n; i++)  // ‖x‖ = 1
▶        x[i] = y[i] / norm;
15   }
16   return x;
17 }
```

k = 2

A				x		y
0,511	0,32	0,12		0,6539		0,5888
0,211	0,62	0,12	·	0,6539	=	0,5888
0,311	0,12	0,16		0,3807		0,3428

$\|y\| = 0{,}9005$ $\quad x \leftarrow y / \|y\|$

$\|y\| = 0{,}9005 \rightarrow x = (0{,}6539;\ 0{,}6539;\ 0{,}3807)$.
$\|y\|$, $\lambda_1 = 0{,}9008$ değerine yaklaşıyor.

Adım 10 / 11

DEĞİŞKENLER

`k = 2` `norm = 0.9005`

`x = [0.6539, 0.6539, 0.3807]`

Power Iteration: Kodu Adım Adım İzle

```
1  double[] kuvvet(double[][] a, int adim) {
2    int n = a.length;
3    double[] x = new double[n];
4    Arrays.fill(x, 1.0);      // x₀ ≠ 0
5    for (int k = 0; k < adim; k++) {
6      double[] y = new double[n];
7      for (int i = 0; i < n; i++)  // y = A·x
8        for (int j = 0; j < n; j++)
9          y[i] += a[i][j] * x[j];
10     double norm = 0;
11     for (double yi : y) norm += yi * yi;
12     norm = Math.sqrt(norm);
13     for (int i = 0; i < n; i++)  // ‖x‖ = 1
14       x[i] = y[i] / norm;
15   }
▶  return x;
17 }
```

sonuç = A · x

0,511	0,32	0,12	0,6539	
0,211	0,62	0,12	0,6539	=
0,311	0,12	0,16	0,3807	

`adim = 3` yineleme bitti: x = (0,6539; 0,6539; 0,3807) ≈ v_1. Bir sonraki yinelemede son bileşen 0,3805 olur; sonra dört basamak artık değişmez.

Adım 11 / 11

DEĞİŞKENLER

`x = [0.6539, 0.6539, 0.3807]`

Power Iteration: Sınırlar ve Uygulamalar

Güçlü yanları

+ Çok basit; yalnızca matris–vektör çarpımı.
+ Seyrek ve **dev** matrislerde (milyarlarca satır) uygulanabilir.
+ Bellek O(n): matris değiştirilmez.

Sınırları

– Yalnızca **baskın** özdeğeri bulur.

– $|\lambda_2| \approx |\lambda_1|$ ise çok yavaş; $|\lambda_1| = |\lambda_2|$ ise yakınsamayabilir.

– x_0, v_1'e dik seçilirse ($c_1 = 0$) kuramsal olarak başarısız olur.

- **PageRank:** web bağlantı matrisinin baskın özvektörü sayfaların önemini verir.
- **Türevler:** ters yineleme (en küçük özdeğer), kaydırmalı yineleme, Lanczos/Arnoldi; PCA'da ilk temel bileşen.

Özet

Problem	Algoritma	Oran	Süre
TSP (genel)	en yakın komşu (sezgisel)	garanti yok	$O(n^2)$
Metrik TSP	MST + önce-kök + kısayol / Christofides	2 / 1,5	$O(n^2)$ / $O(n^3)$
TSP	2-opt yerel arama	garanti yok	$O(n^2)$ / tarama
Makine çizelgeleme	liste çizelgeleme / LPT	$2 - 1/m$ / $4/3 - 1/(3m)$	$O(n \log m)$ / $O(n \log n)$
Düğüm kapsama	kenar seç, iki ucu ekle	2	$O(V + E)$
0/1 sırt çantası	max(oran sıralı, en iyi tek eşya) / FPTAS	2 / $1 + \varepsilon$	$O(n \log n)$ / $O(n^3/\varepsilon)$
Baskın özdeğer	Power Iteration	hata $\propto \lvert\lambda_2/\lambda_1\rvert^k$	$O(n^2)$ / adım

ÖZET

Etkileşimli Simülatörler

Gezgin Satıcı Problemi

Rastgele şehirlerde hızlı en yakın komşu sezgiseli ile kaba kuvvetle bulunan optimal turun karşılaştırması; faktöriyel büyüme.

ÖZET

Sık Yapılan Hatalar

Sezgisel = yaklaşım algoritması sanmak

En yakın komşu ve 2-opt iyi sonuç verir ama **oran garantisi** yoktur; yaklaşım algoritması kanıtlı bir ρ ister.

Oranı ters yazmak

En küçüklemede $C \le \rho \cdot C^*$, en büyüklemede $C \ge C^*/\rho$. ρ her zaman ≥ 1'dir.

Kapsamada tek uç eklemek

Kenarın yalnızca bir ucunu eklemek yıldız çizgede n kat kötü olabilir; 2-yaklaşım **iki ucu da** ekler.

Üçgen eşitsizliğini unutmak

MST tabanlı 2-yaklaşımın kısayol adımı yalnızca **metrik** TSP'de yolu uzatmaz.

Metasezgiseli kesin sanmak

Benzetimli tavlama ve genetik algoritma optimuma yakınsamayı **garanti etmez**; farklı tohum ve parametrelerle birkaç kez çalıştırılıp en iyi sonuç alınır.

Normalleştirmeyi atlamak

Power Iteration'da x her adımda normalleştirilmezse sayılar taşar ($|\lambda_1| > 1$) ya da sıfıra iner.

ÖZET

Örnek Problemler

1. Bir en küçükleme probleminde algoritma 30, optimum 20 buluyor. Bu girdi için oran nedir?
2. 5 şehirlik matriste (TSP örneği) en yakın komşuyu **2** numaralı şehirden başlatın: tur ve uzunluk?
3. 1–2–3–4–5–6 yolunda kenarlar (1,2), (2,3), …, (5,6) sırasıyla işlenirse 2-yaklaşım kaç düğüm seçer? Optimum?
4. m = 2, işler 3, 3, 2, 2, 2: liste çizelgeleme ve LPT C_{max}'ı ve optimum nedir?
5. Benzetimli tavlamada $\Delta = 5$ kadar kötüleştiren bir hamle $T = 10$ ve $T = 1$ iken hangi olasılıkla kabul edilir?
6. W = 10, eşyalar (2, 1) ve (10, 10) (değer, ağırlık): oran sıralı açgözlü ve iyileştirilmiş kural ne bulur?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. a(0,0), b(4,3), c(4,0), d(0,3) ve turda (a,b), (c,d) kenarları var. 2-opt kazancı Δ nedir?
2. $m = 3$ için liste çizelgelemenin 5/3 oranına ulaştığı bir iş listesi kurun.
3. K_5 tam çizgesinde 2-yaklaşım kaç düğüm seçer? Optimum kaçtır?
4. $A = [[2, 0], [0, 1]]$, $x_0 = (1, 1)$. x_k'nın ikinci bileşeni birincisinin 10^{-3}'ünden kaç adımda küçük olur?
5. Benzetimli tavlamada $T_0 = 100$ ve her adımda $T \leftarrow 0{,}997 \cdot T$ ise T kaç adımda 1'in altına iner?
6. 2-yaklaşımı, seçilen eşleştirmeyi de döndürecek biçimde Java ile yazın.

▶ **Cevaplar / ipuçları**

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 35: Approximation Algorithms).
- V. V. Vazirani, *Approximation Algorithms*, Springer, 2001.
- D. P. Williamson ve D. B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.
- R. L. Graham, "Bounds on Multiprocessing Timing Anomalies", *SIAM Journal on Applied Mathematics* 17(2), 1969.
- N. Christofides, "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem", Rapor 388, Carnegie Mellon Üniversitesi, 1976.
- M. Sipser, *Introduction to the Theory of Computation*, 3. baskı, 2012 (3-SAT → düğüm kapsama indirgemesi).
- G. H. Golub ve C. F. Van Loan, *Matrix Computations*, 4. baskı, 2013 (Power Iteration).

Son

Bölüm 6: Yaklaşım Algoritmaları

Sonraki bölüm: Bölüm 7: Rastgele Algoritmalar · Terim sözlüğü