

# Yapay Zeka

## Ders 3 – Bölüm 2

Doç. Dr. Mehmet Dinçer Erbaş  
Bolu Abant İzzet Baysal Üniversitesi  
Mühendislik Fakültesi  
Bilgisayar Mühendisliği Bölümü

# Problem çözen etmen

- Basit refleks etmeni direk olarak durumları aksiyonlara eşler
- Bu sebeple bu eşleştirmenin saklanamayacak kadar büyük yer kaplaması ve öğrenilmesinin çok uzun süre alacağı çevrelerde basit refleks etmenler başarılı olamaz.
- Hedef tabanlı etmenler ise gelecekteki aksiyonları ve bu aksiyonların getirilerini hesaplayarak bu tür çevrelerde başarılı olabilirler.
- Problem çözen etmenler kendilerini istenilen durumlara götürecek aksiyon sıralarını bulan hedef tabanlı etmenlerdir.
- Akıllı etmenler beklenen performans ölçülerini maksimize etmeye çalışırlar.
  - ~ Bu durumu basitleştirmek için etmenin hedefini tanımlayabilir ve bu hedefe ulaşmayı amaçlayabiliriz.
- Hedefler, etmenin başarmaya çalıştığı amaçları sınırlandırarak etmenin davranışını tamamlamıza yardımcı olur.
- Problem çözümündeki ilk adım, şu anki duruma ve etmenin performans ölçüsüne göre oluşturulan **hedef formülasyonu** yapmaktır.

# Problem çözen etmen

- Hedef bir durumlar kümesidir
  - ~ Etmenin görevi, hedef duruma ulaşmasını sağlayacak aksiyon sırasını (şimdi ve gelecekte gerçekleştirmesi gereken aksiyonları) bulmaktır.
  - ~ Bunu yapabilmek için etmen öncelikle, hangi tür durum ve aksiyonları değerlendirmesi gerektiğine karar vermelidir.
  - ~ Verilen bir hedef için değerlendirilmesi gereken aksiyon ve durumları belirleme aktivitesine **problem formülasyonu** denir.
  - ~ Bir etmenin sonuçlarını bilmediği birçok seçeneği olduğunda, bilinen sonuçları olan farklı aksiyon sıralarını deneyebilir ve sonunda en iyi sırayı seçebilir.
    - Bu şekilde aksiyon sırası aramaya **arama** diyoruz.
  - ~ Bir arama algoritması bir problemi girdi olarak alır ve bu problemi çözen bir aksiyon sırasına dönüş yapar.
  - ~ Çözüm bulunduktan sonra çözümü oluşturan aksiyonlar gerçekleştirilebilir – çalıştırma aşaması

# Problem çözen etmen

- Formüle et, arama yap ve uygula

**fonksiyon** BASIT-PROBLEM-COZEN-ETMEN (*algı*) **dönüş** bir aksiyon

**girdiler:** *sira*, bir aksiyon sirasi, başta boş

*durum*, dünyanın şu anki durumunun tanımı

*hedef*, bir hedef, başta tanımsız

*problem*, problem formülasyonu

*durum* <== DURUMU-YENILE(*durum*, *algı*)

**if** *sira* boş **then**

*hedef* <== HEDEF-FORMÜLE-ET(*durum*)

*problem* <== PROBLEM-FORMÜLE-ET(*durum*, *hedef*)

*sira* <== ARAMA(*problem*)

**if** *sira* = başarısız **then return** null aksiyon

*aksiyon* <== ILK(*sira*)

*sira* <== GERISI(*sira*)

**return** *aksiyon*

# Problem çözen etmen

- Etmen hedefini formüle ettikten sonra bir arama prosedürü kullanarak problemi çözer.
- Daha sonra bu çözümü kullanarak yapacağı aksiyonlara sırasıyla karar verir.
  - ~ İlk aksiyonu uygular, daha sonra bir sonrakini, daha sonra bir sonrakini...
- Çözüm tamamlandığında etmen yeni bir hedef formüle eder.

# Problem çözen etmen

- Örnek: Romanya tatil

~ Etmenin Romanya'nın Arad şehrinde tatil yaptığını farzedelim.

~ Etmenin anlık olarak birçok farklı performans ölçüsü olabilir

- Örneğin: güneşlenme, Romencesini geliştirme, etrafı gezme, gece gezmesi, yorgunluktan kaçınma vb.

~ Etmenin bir sonraki gün Bükreş'ten dönüş bileti olsun.

- Bu durumda etmen bir gün içerisinde bulunduğu şehirden Bükreş'e gitmelidir.
- Öyleyse etmenin hedef olarak Bükreş'e gitmeyi seçmesi mantıklıdır.

~ Etmeni Bükreş'e zamanında ulaştırmayan aksiyonlar fazla düşünmeden reddedilebilir ve bu sayede etmenin karar verme problemi basitleştirilebilir.

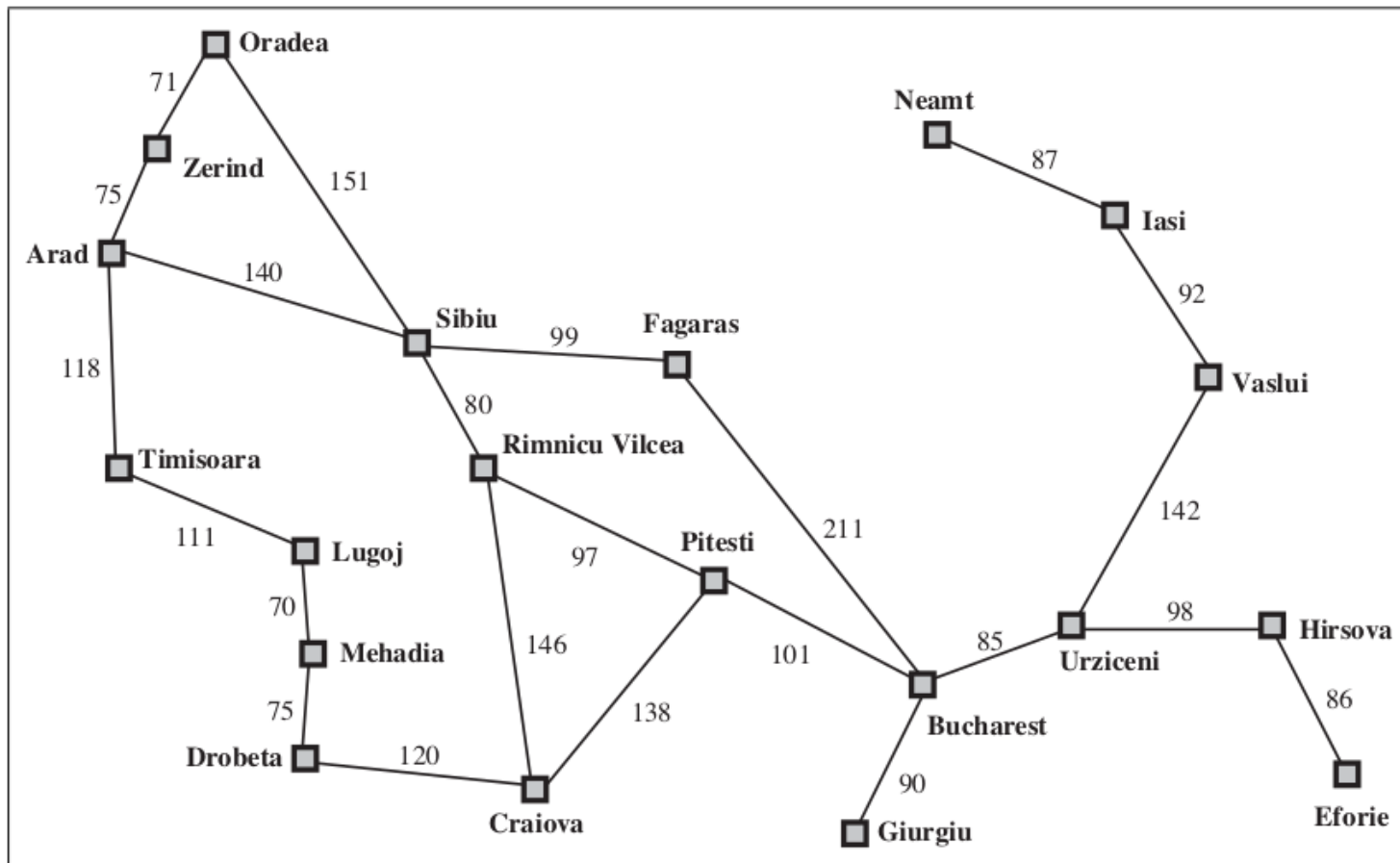
# Problem çözen etmen

- Örnek: Romanya tatili

~ Etmenin buradaki görevi, şimdi veya gelecekte kendisini hedef durumuna ulaştıracak hamleleri seçmektir.

- Eğer etmen, “sol ayağını bir adım ileri at” veya “direksiyonu 1 derece çevir” gibi alt seviye aksiyonları değerlendirseydi bulunduğu yerden pek de uzağa gidemezdi.
- Öyleyse etmen, kendini hedefe ulaştırabilecek aksiyonları değerlendirmelidir.
- Bu durumda etmenin hedefi Bükreş’e araba ile gitmek olsun.
- Problem formülasyonu, Bükreş’e gitmek için değerlendirilecek aksiyonlar ve durumları içerecektir.
- Etmen hedef olarak Bükreş’e araba ile gitmeyi belirledi. Şimdi bunun için Azad şehrinden nereye gitmesi gerektiğini düşünür
  - ~ Azad’dan Bükreş’e gidebilmek için üç farklı yol vardır
  - ~ Siblu, Timisoara ve Zerind şehirleri üzerinden
- Etmenin bu yolları kullanabilmesi için Romanya’daki yollar hakkında bilgisi olmalıdır.

# Problem çözen etmen





# Problem çözen etmen

- Etmenin Romanya yollarını gösteren bir haritası olsun
  - ~ Bu durumda harita üzerindeki noktalar, etmenin ulaşabileceği durumlar ve alabileceği aksiyonlar hakkında bilgi verir.
  - ~ Bu bilgiyi kullanarak etmen, farklı üç şehirden geçen yolları aldığı anda yolculuğunun ne şekilde geçeceğini değerlendirebilir.
- Genel olarak bir etmenin anlık olarak birden fazla opsiyonu olduğunda gelecekteki aksiyonları doğrultusunda neler olacağını değerlendirebilir.
- Öyleyse, etmenin çevresinin özelliklerini inceleyelim.
  - ~ Çevre gözlemlenebilirdir, etmen bulunduğu durumu bilir.
  - ~ Çevre kesiklidir, etmen bulunduğu durumda seçebileceğini sınırlı sayıda aksiyondan birini seçebilir.
  - ~ Çevre bilinendir, etmen yaptığı aksiyonlar sonrası hangi durumlara ulaşacağını bilir.
  - ~ Çevre deterministiktir, her aksiyonun tam olarak bir sonucu vardır.

# Problem çözen etmen

- İyi-tanımlanmış (ing: well-defined) problemler ve çözümleri
  - ~ Bir problem beş bileşen ile resmi olarak tanımlanabilir:
    - **Başlangıç durumu**, etmenin probleme başladığı durum.
      - ~ Örnekte İçinde(Arad)
    - Gerçekleştirilebilecek **aksiyonların** tanımı
      - ~ Belli bir durum  $s$  düşünüldüğünde,  $s$  durumunda gerçekleştirilebilecek aksiyonlar AKSİYONLAR( $s$ ) tarafından verilir.
        - Örnekte Arad şehrinde uygulanabilecek aksiyonlar  $\{Git(Sibiu), Git(Timisoara), Git(Zerind)\}$ .
    - Her aksiyonun ne yaptığının tanımı. Bu tanıma **geçiş modeli** (ing: transition model) denir.
      - ~ SONUÇ( $s, a$ ),  $s$  durumunda  $a$  aksiyonu yapılırsa ulaşılan durumu belirtir.
      - ~ Bir durumdan tek aksiyon ile ulaşılabilen durumlara, o durumun ardılı (ing: successor) denir.
        - Örnekte  $SONUÇ(İçinde(Arad), GIT(Zerind)) = İçinde(Zerind)$
      - ~ Başlangıç durumu, aksiyonlar ve geçiş modeli içsel olarak durum uzayını (geçilebilecek durumlar) tanımlar.

# Problem çözen etmen

- İyi-tanımlı problemler ve çözümleri

~ Bir problem beş bileşen ile resmi olarak tanımlanabilir:

- **Hedef testi**, verilen bir durumun hedef durumu olup olmadığını belirler.

~ Bazı durumlarda hedef olarak belirlenen durumlar kümesi vardır ve hedef testi verilen durumun bu kümede olup olmadığını söyler.

- Örnekte hedef durum kümesi {İçinde(Bükreş)}
- **Yol maliyeti** fonksiyonu, mümkün olan her yola sayısal bir maliyet değeri verir.

~ Durum uzayında bir **yol**, bir dizi aksiyon ile birbirine bağlanan bir dizi durumdur.

~ Adım maliyeti, belli bir durumda alınacak aksiyona maliyet değeri verir.

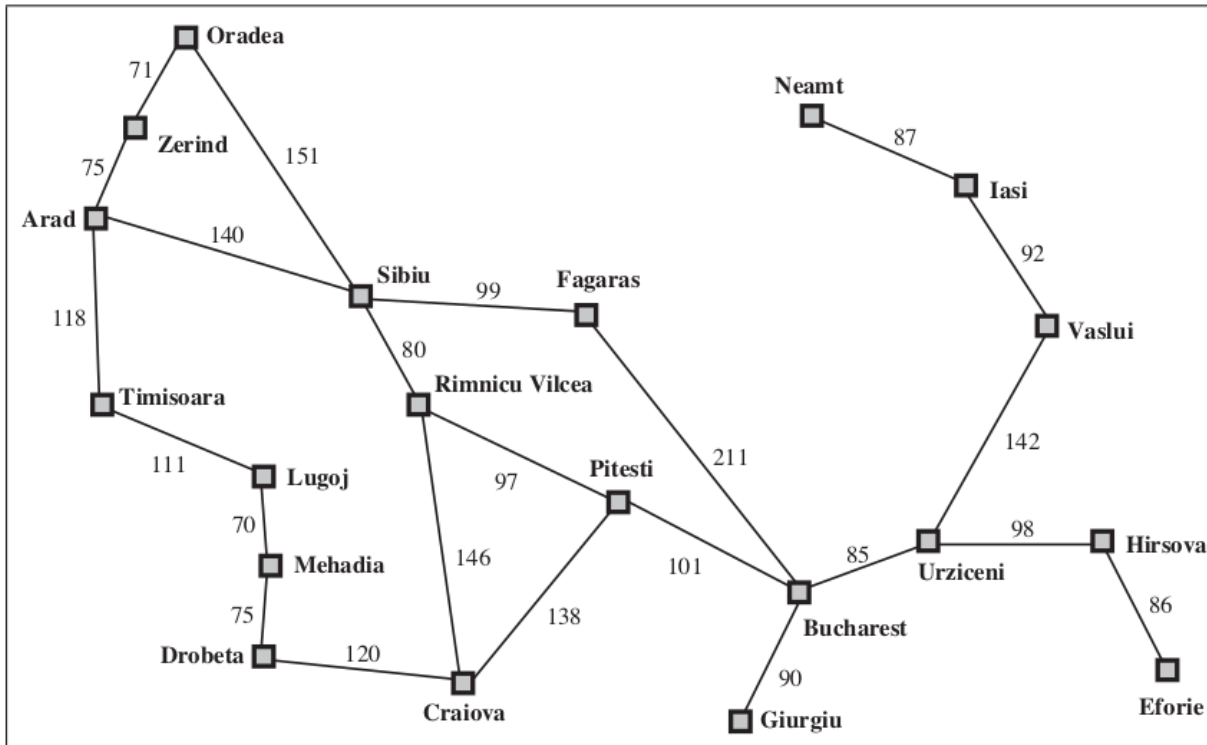
~  $s$  durumunda  $a$  aksiyonu ile  $s'$  durumuna ulaşmanın adım maliyeti  $c(a,s,s')$  ile ifade edilir.

# Problem çözen etmen

- İyi tanımlanmış problemler ve çözümleri

~ Başlangıç durumu, aksiyonlar, ve geçiş modeli içsel olarak problemin durum uzayını tanımlar.

~ Durum uzayı, düğümlerin durumları, aradaki bağlantıların ise aksiyonları temsil ettiği bir yönlü graf veya ağ tanımlar.



# Problem çözen etmenler

- Problem formülasyonu

~ Bir problemin çözümü, etmenin başlangıç durumundan hedef durumuna kadar gitmesini sağlayan aksiyon sırasıdır.

~ Bir çözümün kalitesi yol maliyet fonksiyonu tarafından ölçülür.

- Optimal (en uygun) çözüm en düşük yol maliyetine sahiptir.

~ Gerçek dünyadaki problemler fazlasıyla karmaşık olabilir.

- Örneğin, Arad'dan Bükreş'e giderken birçok farklı faktörü göz önüne alabiliriz. Mesela, yolculuk arkadaşları, yolculuk sırasında çalan müzik, yoldaki polis kontrolü var mı, bir sonraki mola yeri ne kadar uzunlukta...

~ Bu faktörleri çözümü araştırırken görmezden geliriz, çünkü asıl problemimiz olan Bükreş'e giden en uygun yol bulma ile alakaları yoktur.

- Problemi tanımlarken gereksiz detayları dışarda bırakma işlemine **soyutlama** (ing: abstraction) adı verilmiştir.

# Problem çözen etmenler

- Problem formülasyonu

~ Durumları soyutlamaya ek olarak aksiyonlar da soyutlanmalıdır.

- Araba sürme konusunda bir çok detay vardır. Örneğin geçen süre, tüketilen yakıt, oluşturduğu kirlilik vb.
- Ancak bu detayları problemimizde görmezden geliyoruz.

~ Çözümü oluştururken de soyutlama yapılabilir.

- Örneğin Arad – Sibiu – Rimnicu – Vilcea – Pitesti – Bükreş yolunda birçok detay vardır. Ancak plan yaparken bunları görmezden geliyoruz.

~ Soyutlama sayesinde daha sade ve anlaşılır çözümlere ulaşabiliriz.

~ Ancak doğru bir soyutlama yapabilmek için konuya hakim olmalı ve gerektiği kadar detayı, problemin çözümünü bozmayacak şekilde, görmezden gelmeliyiz.

# Problem çözen etmenler

- Örnek problemler:

~ Elektrik süpürgesi dünyası

- **Durumlar**

~ Etmen iki lokasyondan birinde olabilir, iki lokasyon temiz veya kirli olabilir.

~ Bu durumda  $2 \times 2^2 = 8$  farklı durum mümkün.

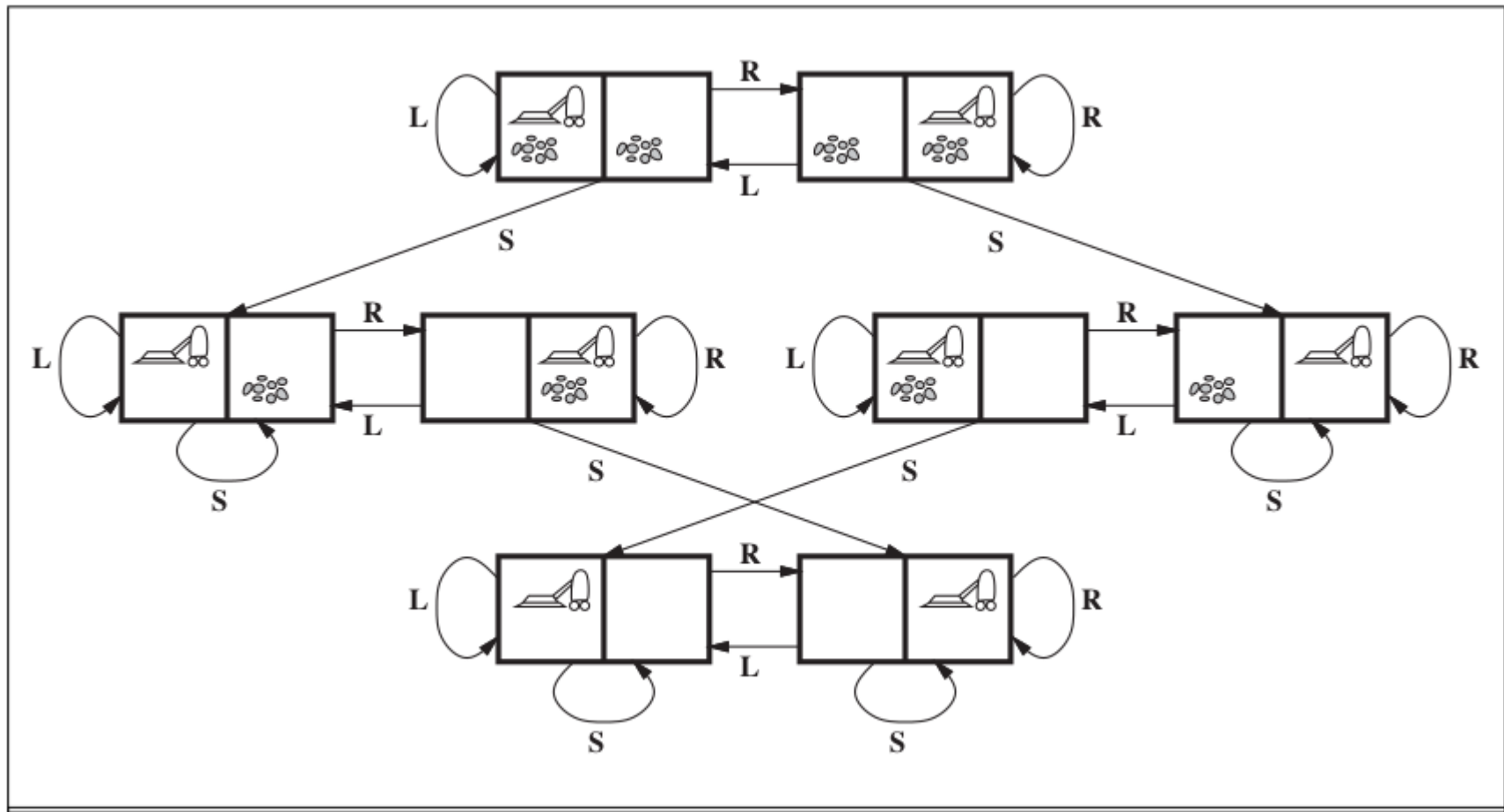
~ 2 değil n tane lokasyon olsaydı?

- **Başlangıç durumu:** Herhangi bir durum başlangıç olabilir.
- **Aksiyonlar:** sola git, sağa git, temizle
- **Geçiş modeli:** Bir sonraki slaytta geçiş modelinin tamamı görülmektedir.
- **Hedef testi:** Her lokasyonun temiz olup olmadığını kontrol eder.
- **Yol maliyeti:** Her aksiyon 1 maliyet olduğunu farzederseniz, yol maliyeti yol içerisindeki toplam aksiyon sayısıdır.

# Problem çözen etmenler

- Örnek problemler

Elektrik süpürgesi dünyası





# Problem çözen etmenler

- Örnek problemler:

~ 8'li bulmaca

7	2	4
5		6
8	3	1

	1	2
3	4	5
6	7	8

~  
Başlangıç durumu

Hedef durumu

# Problem çözen etmenler

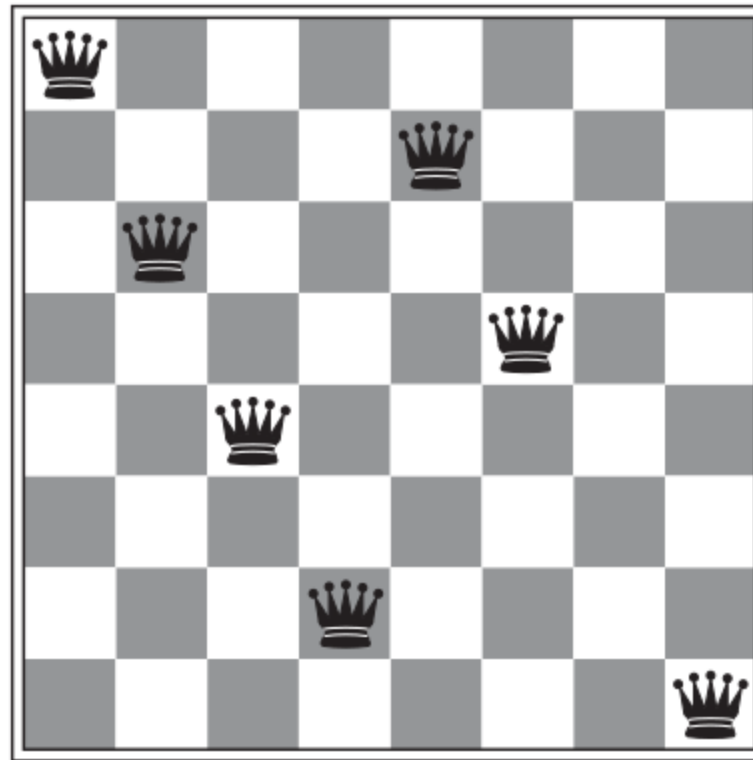
- Örnek problemler:

~ 8'li bulmaca

- **Durumlar:** 8 döşemenin bulmacadaki yerleri ve boş olan döşeme
- **Başlangıç durumu:** Herhangi bir durum başlangıç olabilir.  
~ Belirtilen herhangi bir hedef durumuna ancak olası başlangıç durumlarının yarısından ulaşılabilir.
- **Aksiyonlar:** Bir döşemeyi sağa, sola, yukarı veya aşağı hareket ettirebiliriz.
- **Geçiş modeli:** Verilen bir durum ve yapılan hamle ile bir sonraki duruma ulaşırız.
- **Hedef testi:** Bu test şu anki durumun bir önceki slaytta görülen hedef durumu olup olmadığını test eder.
- **Yol maliyeti:** Her aksiyon 1 maliyet olduğunu farzederseniz, yol maliyeti yol içerisindeki toplam aksiyon sayısıdır.

# Problem çözen etmenler

- Örnek problemler:
  - ~ 8 vezir problemi



# Problem çözen etmenler

- Örnek problemler:

~ 8 vezir problemi

- **Durumlar:** 0 ile 8 arası vezirin satranç tahtasına yerleştirilmiş hali
- **Başlangıç durumu:** Tahta üzerinde vezir yok.
- **Aksiyonlar:** Tahtaya boş olan bir pozisyona vezir yerleştirme.
- **Geçiş modeli:** Yeni eklenen vezir ile tahtanın yeni hali oluşur.
- **Hedef testi:** 8 vezir yerleştirmiş ve hiçbirinin birbirini tehdit etmeme durumu

~ Bir başka alternatif

- **Durumlar:** Soldan başlayarak her sütunda bir vezir, hiçbir vezir birbirini tahdit etmez şekilde oluşacak durumlar
- **Aksiyonlar:** En soldan başlayarak boş olan sütuna diğer vezirleri tehdit etmeyecek şekilde yeni bir vezir yerleştirme

# Problem çözen etmenler

- Gerçek dünyadan örnekler

~ Havayolu yol bulma problemi

- **Durumlar:** Bir havaalanı ve şu anki zaman
- **Başlangıç durumu:** Problem tarafından tanımlanır.
- **Aksiyonlar:** Bir havaalanından belli zamanda kalkan bir uçakla diğer havaalanına ulaşmak
- **Geçiş modeli:** Bir havaalanından diğerine uçuş gerçekleştirilmesi sonucu sonraki durum oluşur.
- **Hedef testi:** Problem tarafından tanımlanmış hedefe vardık mı?
- **Yol maliyeti:** ödenen ücret + bekleme zamanı + uçuş zamanı + gümrük işlemleri + koltuk kalitesi + varış zamanı + uçak tipi + kazanılan bonus ...

# Problem çözen etmenler

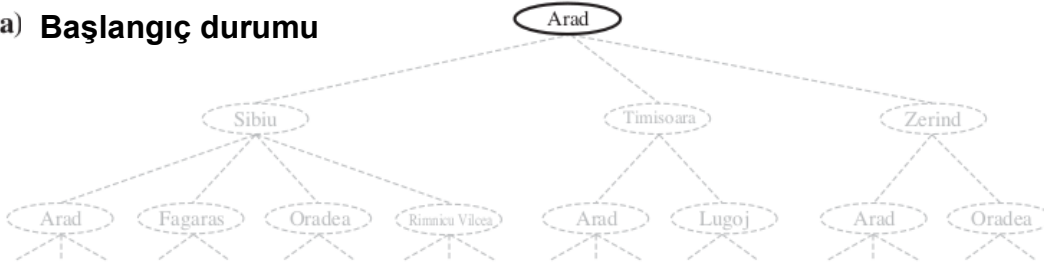
- Gerçek dünyadan örnekler
  - ~ Tur ayarlama problemi
  - ~ Gezgin satıcı problemi (İng: TSP – Travelling salesman problem)
  - ~ VLSI düzeni
  - ~ Robot navigasyonu
  - ~ Otomatik montaj sıralaması
  - ~ Protein dizaynı
  - ~ ...

# Çözüm arama

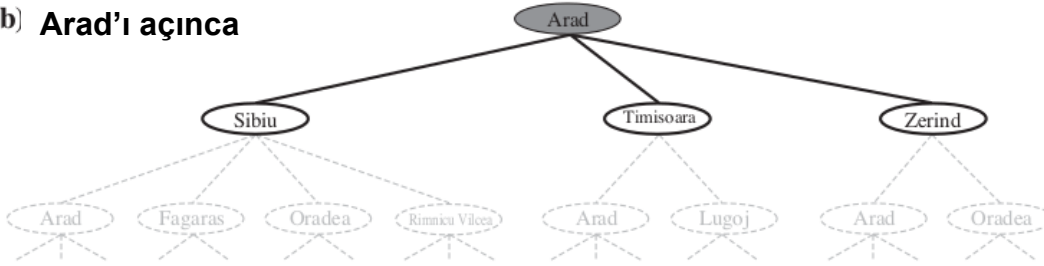
- Şu ana kadar problemleri formüle etmeyi öğrendik.
  - Sırada çözümleri oluşturmak var.
  - Bir çözüm, aksiyon dizisidir. Çözüm algoritmaları farklı aksiyon dizilerini değerlendirir.
  - Başlangıç durumu kök olacak şekilde düşünersek, aksiyon dizileri bir **arama ağacı** oluşturur.
- ~ Ağacın dalları **aksiyonları**, düğümleri ise problemin **durum uzayındaki durumları** temsil eder.

# Çözüm arama

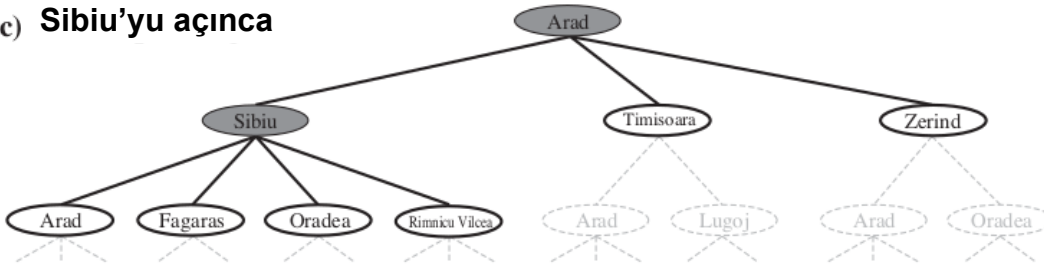
(a) Başlangıç durumu



(b) Arad'ı açınca



(c) Sibiu'yu açınca

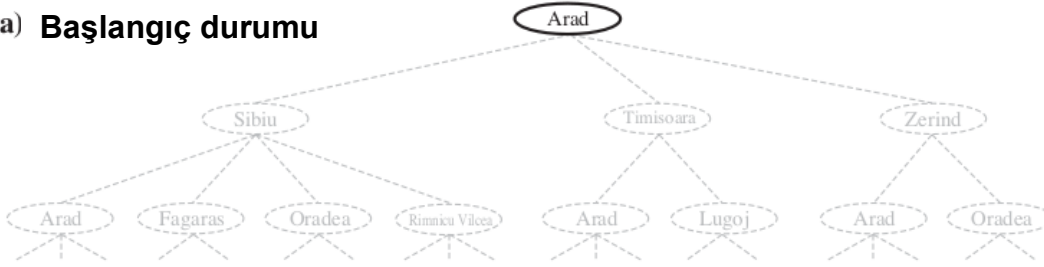


- Ağacın kökü içinde(Arad) durumuna karşılık gelir
- İlk adım olarak hedef durumu olup olmadığını kontrol ederiz
  - ~ Görüldüğü üzere değil
- Öyleyse bir aksiyon almayı değerlendiririz
  - ~ Bunu şu anki durumu açarak yaparız ve yeni durumlar oluşur
- Yeni üç durum açılır
- Şimdi değerlendirmemiz gereken üç yeni durum vardır
  - ~ Bunlardan birini seçmeliyiz.
- Arama bu şekilde yapılır - bir opsiyon seçilir ve diğerleri kenarda bırakılır.

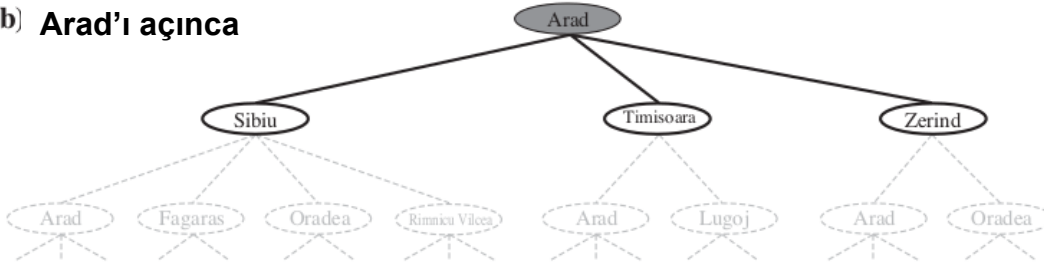


# Çözüm arama

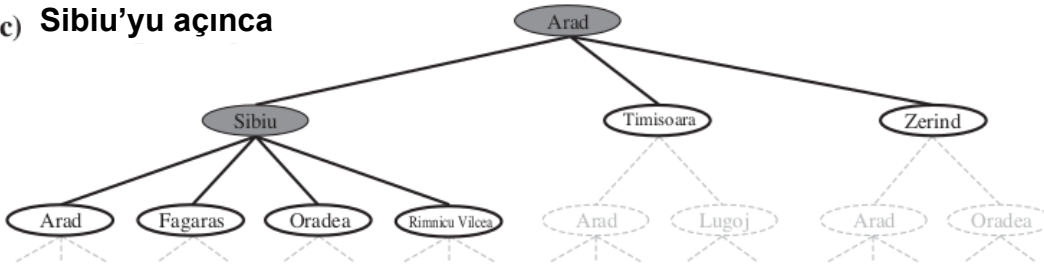
(a) Başlangıç durumu



(b) Arad'ı açınca



(c) Sibiu'yu açınca



- Önce Sibiu'yu seçtiğimizi farzedelim
- Hedef durumu ulaşıldı mı? Hayır!
- O zaman bu düğümü açarız
- Bu durumda ya yeni açılan dört seçenekten biri seçilmeli yada geri dönüp Timisoara veya Zerind seçilmelidir
- Belirtilen altı düğüm yaprak düğümlerdir.
- Yaprak düğümleri çocukları olmayan düğümlerdir.
- Herhangi bir zamanda elimizde bulunan yaprak düğümler kümesi **sınır** olarak adlandırılır.
- Sınırdaki düğümleri açma işlemi çözüme ulaşıncaya veya açacak düğüm kalmayıncaya kadar devam eder.

# Çözüm arama

- Düğümleri ne şekilde açtığımız bizim **arama stratejimizi** belirler.
  - ~ Farklı stratejilerle oluşturulmuş algoritmalara göreceğiz.
- Bazı arkadaşlarımız ilginç bir durumu farketmiş olabilir.
  - ~ içinde(Arad) durumu tekrar ediyor.
  - ~ Döngü içeren yol
- Arama ağacında döngüler varsa tam ağacın büyüklüğü sonsuz olur.
  - ~ Halbuki ağaçta sadece 20 durum var.
- Arama ağaçları değerlendirme yaptıklarında döngü içeren yolları seçmeyecektir.
  - ~ Çünkü döngüsüz daha kısa bir yol bulmak mümkündür.
- Döngü içeren yollar, bir durumdan başka bir duruma birden fazla yoldan ulaşıldığı problemlerde görülen gereksiz yol (ing: redundant path) kavramının özel bir örneğidir.
- Döngülerin engelenebilmesi için keşfedilmiş düğümler kümesi saklanmalıdır.

# Çözüm arama

**fonksiyon** AĞAÇ-ARAMA (*problem*) **dönüş** bir çözüm veya çözümsüz  
sınır düğümleri problemin başlangıç durumuna göre başlat

**loop do**

**if** sınır boş ise **then return** çözümsüz

Bir yaprak düğümü seç ve sınırdan bu düğümü sil

**if** düğüm hedef durumu ise **then return** bulunan çözüm  
seçili düğümü aç, yeni açılan düğümleri sınıra ekle

**fonksiyon** GRAF-ARAMA (*problem*) **dönüş** bir çözüm veya çözümsüz  
sınır düğümleri problemin başlangıç durumuna göre başlat

**keşfedilmiş kümeyi başta boş olarak ayarla**

**loop do**

**if** sınır boş ise **then return** çözümsüz

bir yaprak düğümü seç ve sınırdan bu düğümü sil

**if** düğüm hedef durumu ise **then return** bulunan çözüm

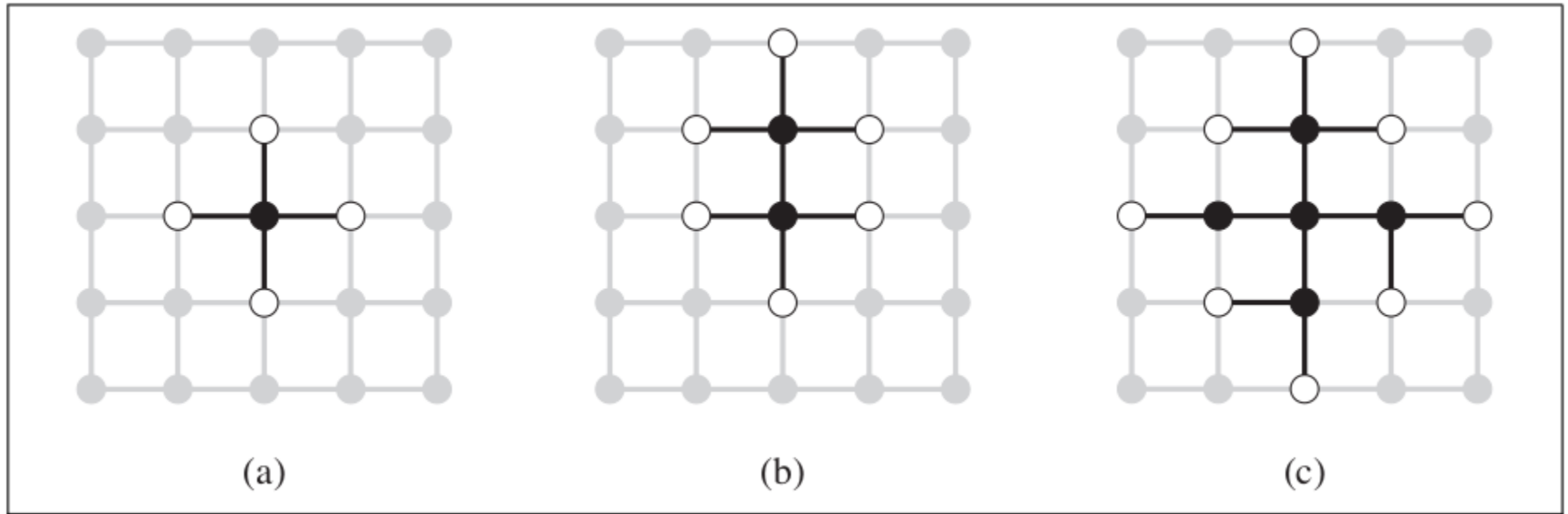
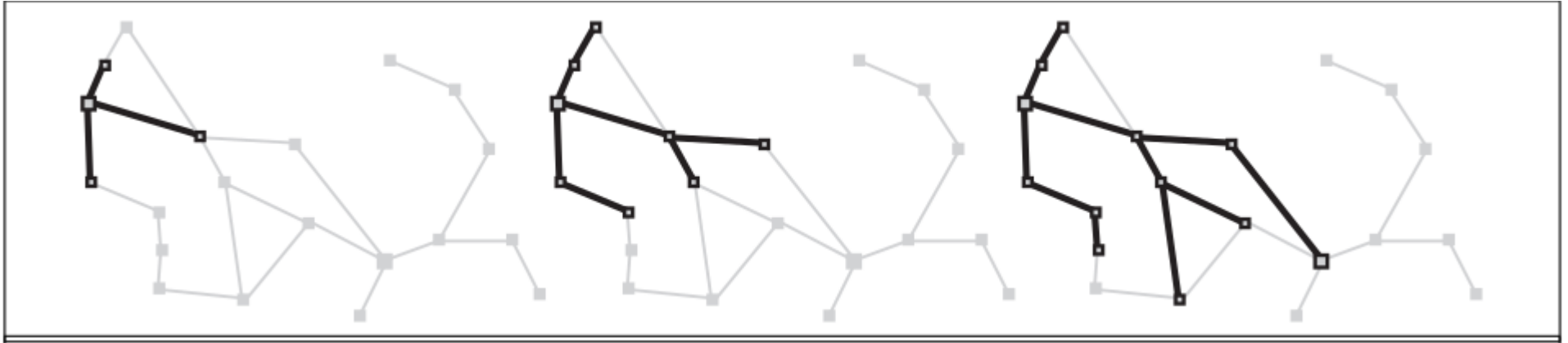
düğümü keşfedilmiş kümeye ekle

seçili düğümü aç, yeni açılan düğümleri, **sınır içerisinde veya  
keşfedilmiş kümede değilse** sınıra ekle.

# Çözüm arama

- GRAF-ARAMA algoritması ile oluşturulan arama ağacı her düğümden en fazla bir tane içerebilir.
- Sınır, durum uzayının keşfedilmiş ve keşfedilmemiş kısımlarını ayırır.
  - ~ Başlangıç durumundan başlayarak keşfedilmemiş bir düğüme giden her yol bir sınır düğümünden geçmelidir.

# Çözüm arama



# Çözüm arama

- Arama algoritmalarının altyapısı

~ Arama algoritmalarının çalışabilmesi için bazı veri yapılarına ihtiyaç vardır. Bir ağaçtaki düğüm  $n$  için aşağı belirtilen 4 yapı bulunur:

- $n.DURUM$ : düğümün karşılık geldiği durum
- $n.ÜST$ : arama ağacında düğümü oluşturan düğüm
- $n.AKSIYON$ : düğümü oluşturmak için üst düğüme uygulanan aksiyon
- $n.YOL-MALİYETİ$ : maliyet, genellikle  $g(n)$  şeklinde gösterilir, başlangıç durumundan düğüme kadar gelen yolun maliyetidir.

# Çözüm arama

- Üst düğüm için 4 bileşen belirtildiğinde alt düğüm şu şekilde oluşturulur.

**fonksiyon** ALT-DÜĞÜM (*problem, üst, aksiyon*) **dönüş** bir düğüm  
**return** aşağıda belirtilen düğüm  
DURUM = *problem.SONUÇ(üst.durum, aksiyon)*  
ÜST = *üst*, AKSİYON = *aksiyon*,  
YOL-MALİYETİ = *üst.YOL-MALİYETİ* + *problem.ADIM-MALİYETİ(üst.DURUM, aksiyon)*

