BLM-431 YAPAY ZEKA

Ders-6 Yerel Arama Algoritmaları ve Optimizasyon Problemleri

Dr. Ümit ATİLA

umitatila@karabuk.edu.tr

http://web.karabuk.edu.tr/umitatilla/

Hesaplamalı Zeka (Computational Intelligence)

- Hesaplamalı zeka, yapay zekanın bir dalıdır.
  - Evrimsel hesaplama (Evolutionary computation),
  - Sürü zekası (Swarm intelligence)
  - Sinir ağları (Neural networks)
  - Bulanık sistemler (Fuzzy systems)
- Hesaplamalı zeka yöntemleri, herhangi bir hesaplamalı algoritma ile çözümü olmayan ve formülleştirilemeyen problemleri çözmeye çalışan yöntemlerdir.
- Genelde biyolojik ilhamlardan beslenerek oluşturulan yöntemler içerir

Hesaplamalı Zeka (Computational Intelligence)

- Hesaplamalı zekada problem sınıfları:
  - Kontrol problemleri
  - Optimizasyon problemleri
  - Sınıflandırma problemleri
  - Regresyon problemleri
  - NP Tam problemler (NP: Non-deterministic polinomial time)

Optimizasyon

- Optimizasyon problemi
  - f: A→R olarak tanımlı bir fonksiyon.
- A'da yer alan her x elemanı için f(x0)<= f(x) sağlayan bir x0 bulmak (minimizasyon)
- A'da yer alan her x elemanı için f(x0)>= f(x) sağlayan bir x0 bulmak (maksimizasyon)
- Gerçek hayat problemleri optimizasyon problemi olarak modellenebilir.
- Burada A, kısıtlar, eşitlikler veya eşitsizlikler ile tanımlanan bir alt Öklit uzayı.
  - A'da yer alan her bir üye bu kısıtları, eşitlikleri veya eşitsizlikleri sağlamalı
- A, arama uzayıdır ve A'nın her bir üyesi ise aday çözümlerdir.
- f fonksiyonu **amaç fonksiyonudur** (objective function)
- Amaç fonksiyonu değerini minimize (veya maksimize) eden uygun bir çözüme optimum çözüm denir.

Optimizasyon

- Amaç fonksiyonunun konveks olmadığı durumlarda yerel minimum veya yerel maksimum noktalar olabilir.
- **Yerel minimum** x* öyle bir noktadır ki bunun etrafındaki bölgede amaç fonksiyon değerleri bu f(x*) değerinden büyük veya ona eşittir.
- **Global optimizasyon:** Uygulamalı matematiğin ve nümerik analizin bir dalı olarak konveks olmayan problemlerin çözümünü sonlu zamanda bulmayı garanti eden deterministik algoritmalar geliştirmeyi hedefler.

NP-Tam Problemler (NP-Complete)

- Hesaplamalı zekada en yaygın temel karmaşık varsayımlarındandır.
- NP-Tam Problem: Bir P polinom zamanında çözülemeyen problemler
- Örn: 5,4,3,2,1 sayılarını bilgisayarda "buble sort" algoritması ile sıralasak algoritma en fazla n^2 karşılaştırma yaparak çözümü bulur.
- Çözüme giden adım sayısı n'in bir kuvveti gibi olan problemler "polinom" zamanında çözülebilir. Bilgisayarlar kolayca çözebilir.

NP-Tam Problemler (NP-Complete)

- Örn: Gezgin satıcı probleminde belli sayıda şehir var. Bu şehirlerin her biri bir defa ziyaret edilecek, tüm şehirlerden geçilecek. Hedef en az maliyetli yolu bulmak.
- Yaklaşık 100 şehir varsa, elimizde saniyede 10^{18} işlem yapabilecek bir bilgisayar bulunuyorsa problemin yukarıda bahsettiğimiz yaklaşımla çözülmesi 400.000 yıl alacaktır.
- Bu problem NP-Tam dır ve bunu çözebilecek verimli bir algoritma henüz yoktur.
- Bu sınıfa giren problemler için çözümleme zamanı arttıkça artan (super increasing) yapıya sahip olmaktadır.
- Problem yapı olarak artan zamanda çözüldüğü için de bu problem tiplerinin çokterimli zamanda (polynomial time) çözülmesi mümkün değildir

Yerel Arama

- Sistematik olarak arama uzayını keşfeden algoritmalar
  - Breadth First Search (BFS), Depth First Search (DFS), A* vs.
  - Sistematiklik:
    - Yolu hafızada tutma
    - Her noktada hangi düğümün açıldığını, hangilerinin açılmadığını kaydetme
    - Hedef düğüm bulunduğunda çözüm olarak yolu sunar.
- Çoğu problemde ise yolu bulmak gereksizdir.
  - Örn: 8-vezir probleminde önemli olan 8 vezirin son dizilimlerinin ne olduğu. Her adımda vezirlerin hangi konumlarda olduğu önemli değil.
  - Örn: Entegre devre tasarımı, araç rotalama, job shop scheduling, portföy yönetimi vb.

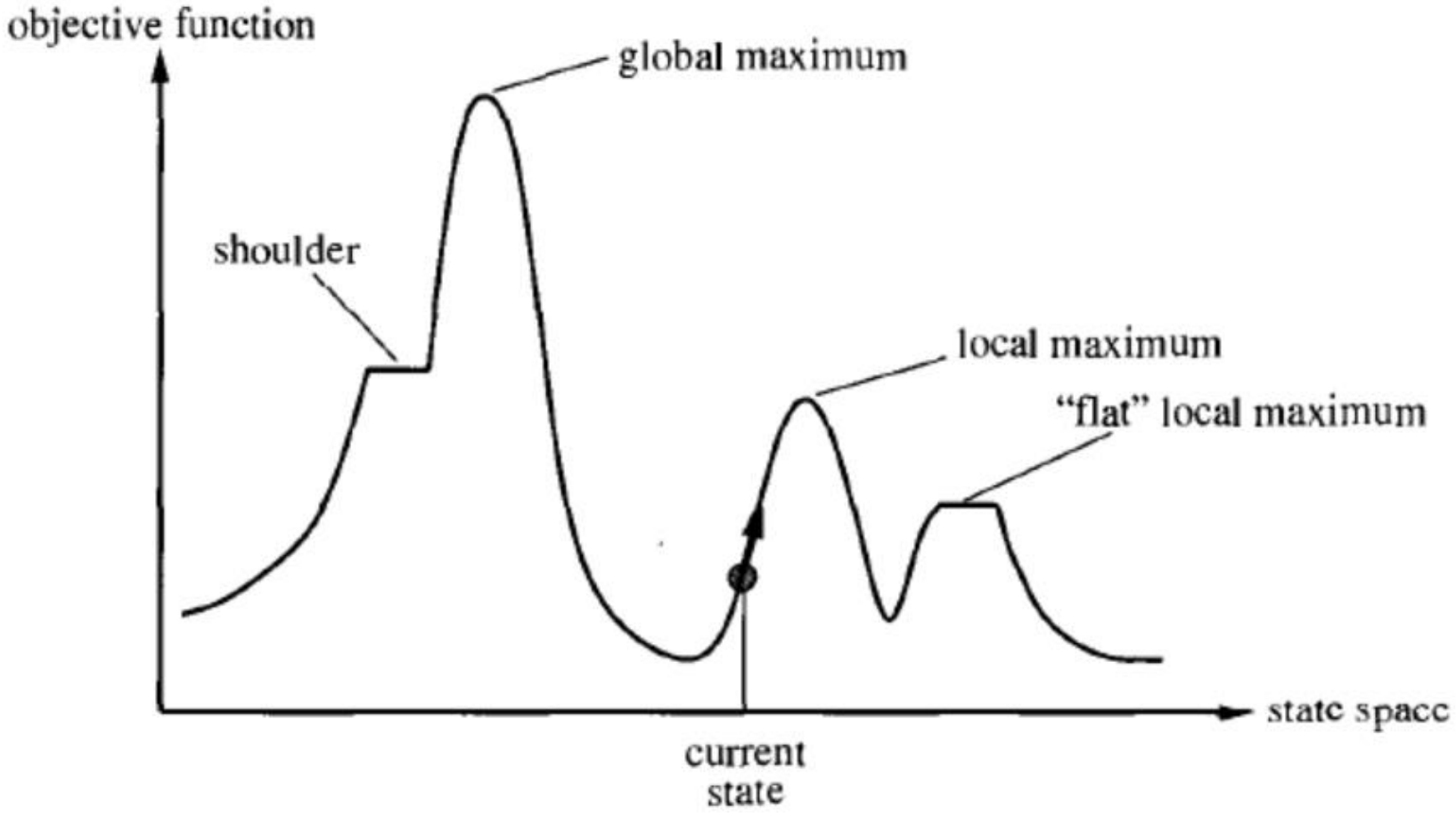
Yerel Arama

- Eğer bize çözüme giden yol değil de çözüm lazımsa sistematik olarak arama uzayını keşfeden algoritmalardan farklı yöntemleri düşünebiliriz.
- Yerel arama algoritmaları:
  - Yola değil çözüme odaklıdır.
  - Birçoğunda mevcut bulunulan durumdan genelde komşu bir duruma geçiş yapılır.
  - Takip ettiği yolu tutmaz.
- Sistematik olarak arama uzayını keşfeden algoritmalara göre hafıza tüketimi çok düşüktür.
- Sistematik algoritmaların yetersiz kaldığı geniş veya sürekli arama uzaylarında kabul edilebilir çözümler bulabilir.

Yerel Arama

- Yerel arama algoritmaları sadece hedef durumu bulmada değil aynı zamanda belli bir amaç fonksiyona göre en iyi durumun arandığı optimizasyon problemlerinin çözümü için de uygundur.
  - Birçok optimizasyon problemi daha önce bahsedilen sistematik arama algoritmaları ile çözülmeye uygun değildir.
- 1 boyutlu durum uzayında konum durumu, yükseklik ise amaç fonksiyonu veya maliyet fonksiyonunu gösterir.
- Yükseklik amaç fonksiyonu ise amaç en yüksek noktayı bulmak (global maximum), yükseklik maliyet fonksiyonu ise amaç en düşük noktayı bulmak (global minimum)
- Complete yerel arama algoritmaları eğer çözüm varsa onu bulur.
- Optimal algoritmalar ise global maksimum/minimum noktayı bulur.

Yerel Arama


objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

Hill Climbing Algoritması

- Bu algoritma mevcut durumun komşuları arasından sürekli en yüksek değerli olanı seçer.
- Başlangıç noktasından itibaren yukarı yönlü hareket gerçekleştirir.
- Bir tepe noktasına vardığında yani komşulardan hiç biri mevcut durum değerinden daha yüksek değilse algoritma durur.
- Algoritma bir ağaç yapısı kullanmaz. Sadece mevcut durumu ve amaç fonksiyonunu tutar.
- Amaç fonksiyonu yerine sezgisel bir maliyet fonksiyonu h kullanılsaydı bu durumda en düşük h değerine sahip komşuya gidilmesi gerekirdi.

Hill Climbing Algoritması

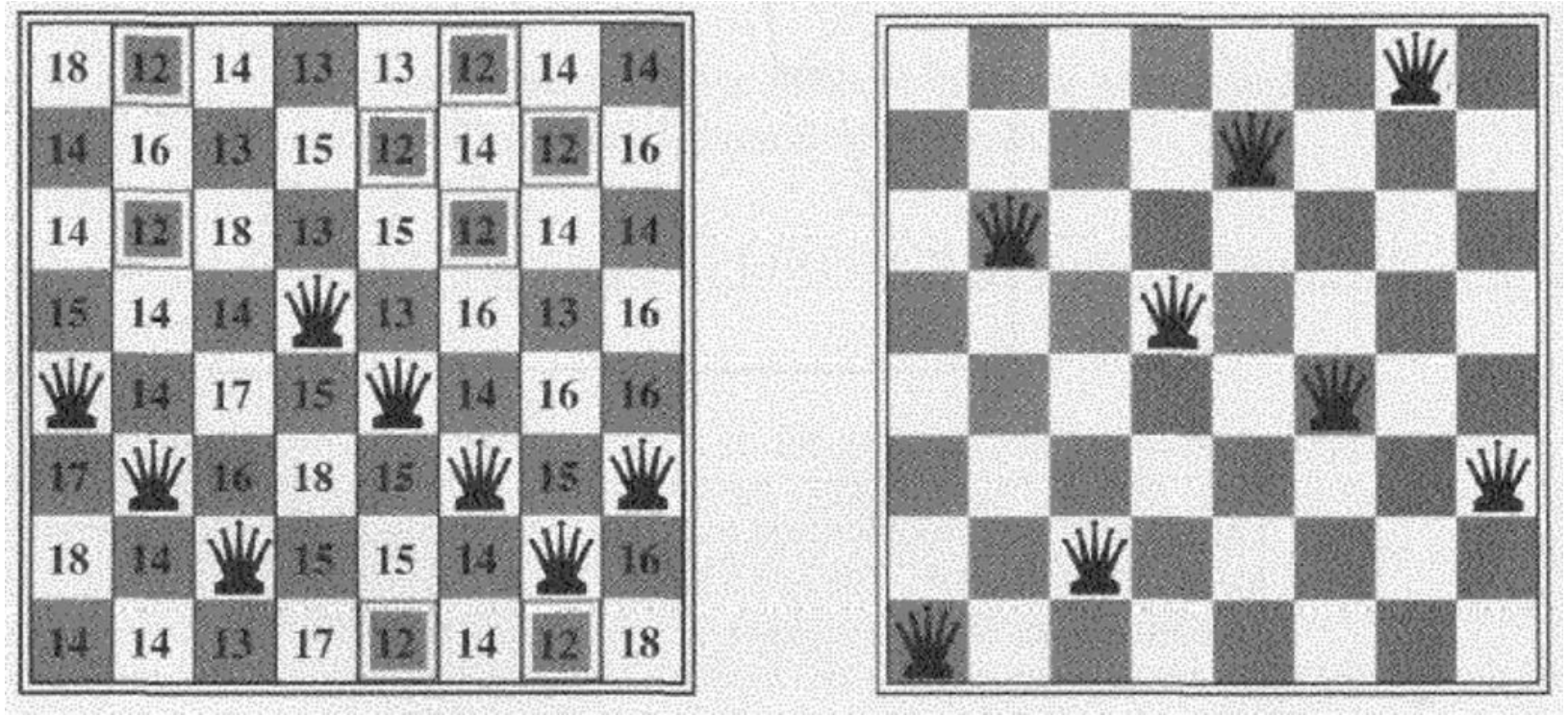
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Hill Climbing Algoritması

- Yerel arama algoritmaları tam durum formülasyonu (complete state formulation) kullanırlar.
- Örn: 8-puzzle problemi için yerel arama algoritmaları genelde 8 vezirinde tahta üzerinde bir sütunda olduğu durumları kullanır.
- Successor fonksiyonu bir sütundaki vezirin aynı sütunda başka bir kareye hareket ettirilmesi sonucu oluşan tüm olası durumları döndürür.
- Bu sayde her durumun 8x7=56 successor u olur.

Hill Climbing Algoritması



- h=17 olan durum solda görülüyor.
- Bir vezirin kendi bulunduğu sütunda başka bir kareye hareket ettirilmesi sonucu oluşacak olan maliyet fonksiyonu h değerleri tahta üzerinde gösteriliyor. En iyi hamleler işaretlenmiştir.
- 8-vezir durum uzayında bir yerel nokta sağda görülüyor. Burada h=1 ve tüm successorlar için h değeri 1 den büyük.

Hill Climbing Algoritması

- Burada kullanılan sezgisel maliyet fonksiyon *h*, doğrudan veya dolaylı olarak birbirine hamle yapabilen vezir çifti sayısıdır.
- Bu maliyet fonksiyonunun global minimum değeri h=0 dır ve sadece en iyi çözümde elde edilir.
- Burada en iyi successor lar h=12 olarak işaretli.
- Hill climbing algoritması eğer birden fazla en iyi successor varsa bunlar arasından birisini rasgele seçer.
- Hill climbing algoritması çözüme doğru hızlıca hareket eder çünkü kötü durumun iyileştirilmesi genelde çok kolaydır.
- Örneğin solda verilen 8 vezir durumundan (h=17), sağda verilen duruma (h=1) sadece 5 adımda gelinir.

Hill Climbing Algoritması

- Hill Climbing algoritmasının dezavantajlı yönlerine bakacak olursak.
- 1) Optimal çözüm etrafında bir çok yerel maksimum veya yerel minimum noktaları varsa burada takılıp kalınabilir ve böylece optimal çözüme ulaşmak ya mümkün olamayacaktır ya da çok zaman alacaktır.
  - 8 vezir örneğinde h=1 aslında yerel maksimumdur (h sezgisel maliyet fonksiyonuna göre yerel minimum). Çünkü bu durumda vezirlerden her hangi birinin hareketi daha kötü bir durumla sonuçlanır.
- 2) Bir platoya rastlanabilir ve çözümde herhangi bir iyileşme meydana getirilemez. Bu durumda çözüm rasgele değişir ve büyük olasılıkla optimal çözüm bulunamaz.
- 3) Çok hafif eğimlere rastlanabilir ve algoritmanın bakacağı her yön ona çıkış gibi görünebilir ve algoritma optimal çözümü bulduğu kanısına varabilir.

Hill Climbing Algoritması

- Rasgele vezir konumları ile 100 defa başlandığında Hill Climbing algoritması 86'sında yerel maksimum veya plato da takılma problemi yaşayacaktır. 14 ünde ise başarılı olacaktır.
- Algoritma takıldığı durumlarda 3 adım, başarılı olduğu durumlarda ise 4 adımda sonuca gidecektir.
- 8^8 = 17 milyon duruma sahip durum uzayı için fena sayılmaz.

Hill Climbing Algoritması

- Algoritma platoya rastladığında başarısız olur çünkü en iyi successor lar mevcut durumla aynı değere sahiptir.
- Bir omuz bölgesinde olma ümidiyle yatayda hareket etmeye devam edilebilir ancak dikkatli olunmalıdır.
- Eğer bulunulan nokta omuz bölgesi değil de düz bir yerel maksimum noktası ise bu durumda sonsuz döngüye girilebilir.
- Bu durumda 100 ardışıl deneme ile omuz bölgesi arama işlemi sınırlandırılabilir.
- Rasgele vezir konumları ile 100 defa başlandığında Hill Climbing algoritması başarılı olma sayısını 14'ten 94'e çıkaracaktır.
- Ancak bunun bir maliyeti vardır. Bu da başarılı olunan durumlarda algoritmanın 21 adım gerçekleştirmesi, başarısız olunan durumlarda ise 64 adım gerçekleştirmesidir.

Hill Climbing

- Algoritma complete değildir çünkü yerel minimuma takılma ihtimali vardır.
- Bu sebeple yerel aramada global minimumu bulacağımızı garanti edemeyiz çünkü problem uzayında bir çok yerel minimum noktalar olabilir.
- Hill Climbing yerel optimum çözümler bulmak için uygundur ancak en iyi çözümü (global optimum) bulmak için uygun değildir.
- O zaman Hill Climbing ile bu problemin üstesinden gelemiyorsak nasıl üstesinden geleceğiz?

Hill Climbing

- Yerel arama algoritmaları :
  - Keşif yapanlar (Random walk gibi)
  - Mevcut çözümü kullananlar (Hill Climbing gibi)
- Bazen bu iki yöntemi bir algoritmada birleştirerek daha güçlü algoritmalar elde edebiliriz.
- Yani hem mevcut çözümü geliştirmek için yerel arama yapmak hem de durum uzayında bir yerlerde saklı kalmış daha iyi bir çözüm var mı diye bakmak.
- Bunu gerçekleştiren yöntemler genelde Evrimsel yöntemler olarak bilinir.
- Biyolojik yöntemlerden esinlenen (Genetik Algoritma) veya fizikten esinlenen yöntemler (Simulated Annealing) gibi.

Simulated Annealing Algoritması

- Simulated Annealing, aslen metallerin soğurken mükemmel atom dizilişlerini örnek alan bir algoritmadır.
- Anneal: Metalin ısıtıp sonra yavaşça soğutularak tava getirilmesidir. Metalin ısıtılması ve sonra soğutulması suretiyle fiziksel özelliklerinin değiştirilmesi işlemidir.
- Metal soğudukça yapısı sabitleşir. Simulated annealing algoritmasında bu ısıtma işlemini takip edecek bir sıcaklık değeri tutulur.

Simulated Annealing Algoritması

- Başlangıçta yüksek bir sıcaklık değeri ayarlanır ve rasgele bir çözüm seçilir.
- Bu noktadan sonra en iyi hareketi seçmek yerine mevcut çözümden rasgele küçük bir değişiklik yaparak bir komşu çözüm bulunur.
- Bu komşu çözüme geçiş yapıp yapılmayacağına enerjiye göre karar verilir.
- Eğer yeni çözümün enerjisi mevcut olandan daha büyük ise bu yeni çözüme geçilir. Yeni çözümün enerjisi mevcuttan daha düşük ise doğrudan geçilmez. Belli bir olasılıkla yeni duruma geçiş yapılır.
- Yeni durum ne kadar fazla eskiye göre kötü olursa olasılık üssel olarak o kadar azalır.

$$P = e^{\frac{E_{yeni} - E_{eski}}{T}}$$

Simulated Annealing Algoritması

- Algotima çalıştıkça sıcaklık yavaş yavaş düşürülür. Bu sıcaklık yüksek olduğu sürece algoritma mevcut durumdan daha kötü olan durumları daha yüksek olasılıkla kabul eder.
- Bu da algoritmanın yerel optimumlarda takılıp kalmasını engeller. Sıcaklık düştükçe kötü durumları seçme sıklığı şansı da azalacak ve böylece algoritmanın global optimuma daha yakın bir alana odaklanması sağlanacaktır.
- Bu soğutma işlemi simulated annealing'in geniş çaplı ve çok sayıda yerel optimumlar barındıran problemlerde optimum çözüme yakın çözümler bulmasını sağlar.

Simulated Annealing Algoritması

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[*problem*])
  for $t \leftarrow 1$ **to** ∞ **do**
    $T \leftarrow schedule[t]$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{\Delta E/T}$