Yapay Zekâ El Kitabı Super VIP

Afshine Amidi ve Shervine Amidi

September 14, 2019

Contents

1 Refleks-temelli modeller

Yavuz Kömeçoğlu ve Ayyüce Kızrak tarafından çevrilmiştir

1.1 Doğrusal öngörücüler

Bu bölümde, girdi-çıktı çiftleri olan örneklerden geçerek, deneyim ile gelişebilecek refleks-temelli modelleri göreceğiz.

❒ **Öznitelik vektörü** – x girişinin öznitelik vektörü $\phi(x)$ olarak not edilir ve şöyledir:

$$\boxed{\phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_d(x) \end{bmatrix} \in \mathbb{R}^d}$$

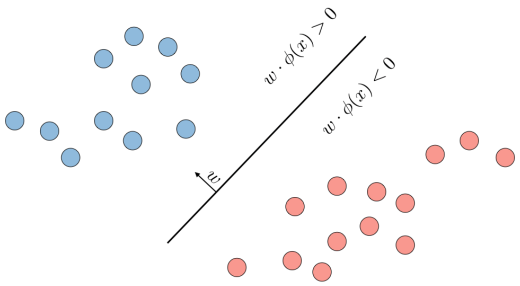
❒ **Puan** – Bir örneğin $s(x,w)$ si ni $(\phi(x),y) \in \mathbb{R}^d \times \mathbb{R}$, $w \in \mathbb{R}^d$ doğrusal ağırlık modeline bağlı olarak:

$$\boxed{s(x,w) = w \cdot \phi(x)}$$

1.1.1 Sınıflandırma

❒ **Doğrusal sınıflandırıcı** – Bir ağırlık vektörü $w \in \mathbb{R}^d$ ve bir öznitelik vektörü $\phi(x) \in \mathbb{R}^d$ verildiğinde, ikili doğrusal sınıflandırıcı f_w şöyle verilir:

$$\boxed{f_w(x) = \text{sign}(s(x,w)) = \begin{cases} +1 & \text{ise } w \cdot \phi(x) > 0 \\ -1 & \text{ise } w \cdot \phi(x) < 0 \\ ? & \text{ise } w \cdot \phi(x) = 0 \end{cases}}$$



❒ **Marj** – $(\phi(x),y) \in \mathbb{R}^d \times \{-1,+1\}$ örneğinin $m(x,y,w) \in \mathbb{R}$ marjları $w \in \mathbb{R}^d$ doğrusal ağırlık modeliyle ilişkili olarak, tahminin güvenirliği ölçülür: daha büyük değerler daha iyidir. Şöyle ifade edilir:

$$\boxed{m(x,y,w) = s(x,w) \times y}$$

1.1.2 Bağlanım

❒ **Doğrusal bağlanım** – $w \in \mathbb{R}^d$ bir ağırlık vektörü ve bir öznitelik vektörü $\phi(x) \in \mathbb{R}^d$ verildiğinde, f_w olarak belirtilen ağırlıkların doğrusal bir bağlanım (linear regression) çıktısı şöyle verilir:

$$\boxed{f_w(x) = s(x,w)}$$

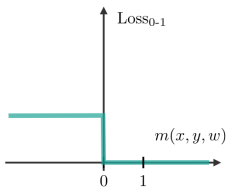
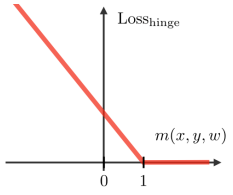
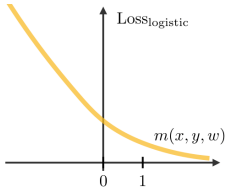
❒ **Artık** – Artık (residual) $\text{res}(x,y,w) \in \mathbb{R}$, $f_w(x)$ tahmininin y hedefini aştığı miktar olarak tanımlanır:

$$\boxed{\text{res}(x,y,w) = f_w(x) - y}$$

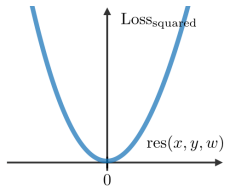
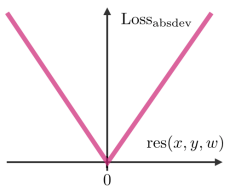
1.2 Kayıp minimizasyonu

❒ **Kayıp fonksiyonu** – Kayıp fonksiyonu $\text{Loss}(x,y,w)$, x girişinden y çıktısının öngörme görevindeki model ağırlıkları ile ne kadar mutsuz olduğumuzu belirler. Bu değer eğitim sürecinde en aza indirmek istediğimiz bir miktar.

❒ **Sınıflandırma durumu** – Doğru etiket $y \in \{-1,+1\}$ değerinin x örneğinin doğrusal ağırlık w modeliyle sınıflandırılması $f_w(x) \triangleq \text{sign}(s(x,w))$ belirleyicisi ile yapılabilir. Bu durumda, sınıflandırma kalitesini ölçen bir fayda ölçütü $m(x,y,w)$ marjı ile verilir ve aşağıdaki kayıp fonksiyonlarıyla birlikte kullanılabilir:

Ad	Sıfır-bir kayıp	Menteşe kaybı	Lojistik kaybı
$\text{Loss}(x,y,w)$	$1_{\{m(x,y,w) \leqslant 0\}}$	$\max(1 - m(x,y,w), 0)$	$\log(1 + e^{-m(x,y,w)})$
Örnekleme			

❒ **Regresyon durumu** – Doğru etiket $y \in \mathbb{R}$ değerinin x örneğinin bir doğrusal ağırlık modeli w ile öngörülmesi $f_w(x) \triangleq s(x,w)$ öngörüsü ile yapılabilir. Bu durumda, regresyonun kalitesini ölçen bir fayda ölçütü $\text{res}(x,y,w)$ marjı ile verilir ve aşağıdaki kayıp fonksiyonlarıyla birlikte kullanılabilir:

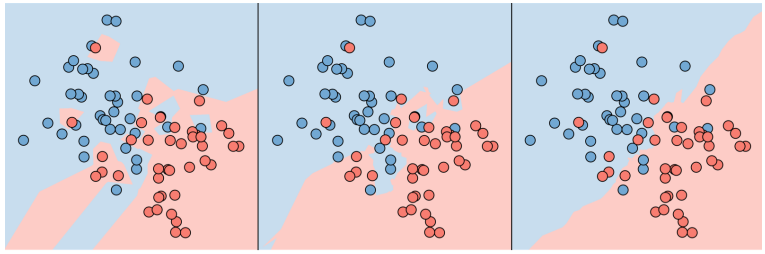
Ad	Kareler kaybı	Mutlak sapma kaybı
$\text{Loss}(x,y,w)$	$(\text{res}(x,y,w))^2$	$\lvert\text{res}(x,y,w)\rvert$
Görselleştirme		

❒ **Kayıp minimize etme çerçevesi** – Bir modeli eğitmek için, eğitim kaybını en aza indirmek istiyoruz:

$$\boxed{\text{TrainLoss}(w) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x,y,w)}$$

1.3 Doğrusal olmayan öngörücüler

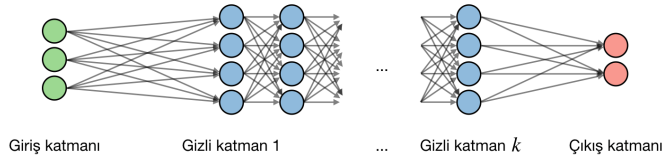
❒ **k-en yakın komşu** – Yaygın olarak k-NN (k-nearest neighbors) olarak bilinen k-en yakın komşu algoritması, bir veri noktasının tepkisinin eğitim kümesinden k komşularının yapısı tarafından belirlendiği parametrik olmayan bir yaklaşımdır. Hem sınıflandırma hem de regresyon ayarlarında kullanılabilir.



$k = 1$ $k = 3$ $k = 11$

Not: k parametresi ne kadar yüksekse, önyargı (bias) o kadar yüksek ve k parametresi ne kadar düşükse, varyans o kadar yüksek olur.

❒ **Yapay sinir ağları** – Neural networks are a class of models that are built with layers. Commonly used types of neural networks include convolutional and recurrent neural networks. The vocabulary around neural networks architectures is described in the figure below:



i ağın i. katmanı ve j, katmanın j. gizli birimi olacak şekilde aşağıdaki gibi ifade edilir:

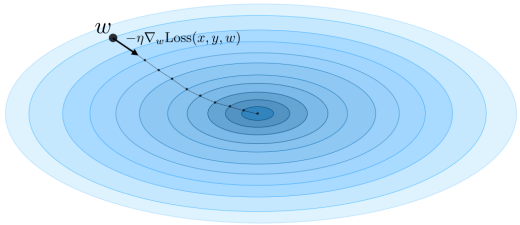
$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

w, b, x, z değerlerinin sırasıyla nöronun ağırlık, önyargı (bias), girdi ve aktive edilmemiş çıkışını olarak ifade eder.

1.4 Stokastik gradyan inişi

❒ **Gradyan inişi** – $\eta \in \mathbb{R}$ öğrenme oranını (aynı zamanda adım boyutu olarak da bilinir) dikkate alınarak, gradyan (bayır) inişine ilişkin güncelleme kuralı, öğrenme oranı ve Loss(x,y,w) kayıp fonksiyonu ile aşağıdaki şekilde ifade edilir:

$$\boxed{w \longleftarrow w - \eta\nabla_w \text{Loss}(x,y,w)}$$



❒ **Stokastik güncellemeler** – Stokastik gradyan inişi (SGİ/SGD, stochastic gradient descent), bir seferde bir eğitim örneğinin $(\phi(x),y) \in \mathcal{D}_{\text{train}}$ parametrelerini günceller. Bu yöntem bazen gürültülü, ancak hızlı güncellemeler yol açar.

❒ **Yığın güncellemeler** – Yığın gradyan inişi (YGİ/BGD, batch gradient descent), bir seferde bir grup örnek (örneğin, tüm eğitim kümesi) parametrelerini günceller. Bu yöntem daha yüksek bir hesaplama maliyetiyle kararlı güncelleme talimatlarını hesaplar.

1.5 İnce ayar modelleri

❒ **Hipotez sınıfı** – Bir hipotez sınıfı $\mathcal{F}$, sabit bir $\phi(x)$ ve değişken w ile olası öngörücü kümesidir:

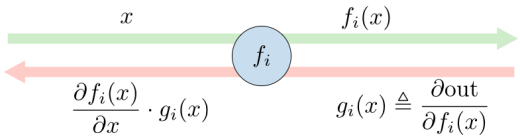
$$\boxed{\mathcal{F} = \left\{f_w : w \in \mathbb{R}^d\right\}}$$

❒ **Lojistik fonksiyon** – Ayrıca sigmoid fonksiyon olarak da adlandırılan lojistik fonksiyon (sigmoid function) σ, şöyle tanımlanır:

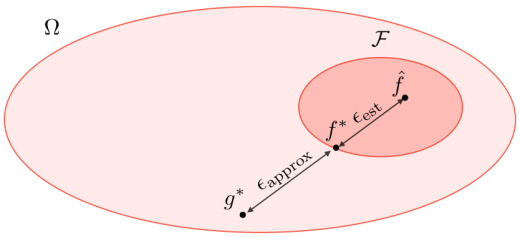
$$\boxed{\forall z \in]-\infty,+\infty[, \quad \sigma(z) = \frac{1}{1+e^{-z}}}$$

Not: $\sigma'(z) = \sigma(z)(1-\sigma(z))$ şeklinde ifade edilir.

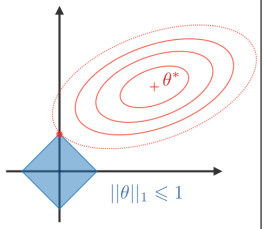
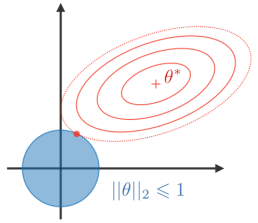
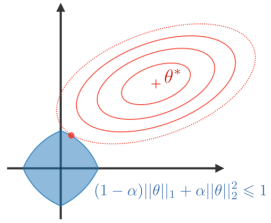
❒ **Geri yayılım** – İleriye geçiş, i'de yer alan alt ifadenin değeri olan fi ile yapılırken, geriye doğru geçiş $g_i = \frac{\partial \text{out}}{\partial f_i}$ aracılığıyla yapılır ve f_i'nin çıkışı nasıl etkilediğini gösterir.



❒ **Yaklaşım ve kestirim hatası** – Yaklaşım hatası ϵ_{approx}, $\mathcal{F}$ tüm hipotez sınıfının hedef öngörücü g^* ne kadar uzak olduğunu gösterirken, kestirim hatası ϵ_{est} öngörücüsü $\hat{f}$, $\mathcal{F}$ hipotez sınıfının en iyi yordayıcısı f^*'ya göre ne kadar iyi olduğunu gösterir.



❒ **Düzenlileştirme** – Düzenlileştirme (regularization) prosedürü, modelin verilerin aşırı öğrenmesinden kaçınmayı amaçlar ve böylece yüksek değişkenlik sorunlarıyla ilgilenir. Aşağıdaki tablo, yaygın olarak kullanılan düzenlileştirme tekniklerinin farklı türlerini özetlemektedir:

LASSO	Ridge	Elastic Net
- Katsayıları 0'a düşürür - Değişken seçimi için iyi	Katsayıları daha küçük yapar	Değişken seçimi ile küçük katsayılar arasında ödünleşim
		
$... + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$... + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$... + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \quad \alpha \in [0,1]$

❒ **Hiperparametreler** – Hiperparametreler öğrenme algoritmasının özellikleridir ve öznitelikler dahildir, λ normalizasyon parametresi, yineleme sayısı T, adım büyüklüğü η, vb.

❒ **Kümeler** – Bir model seçerken, veriyi aşağıdaki gibi 3 farklı parçaya ayırırız:

Eğitim kümesi	Doğrulama kümesi	Test kümesi
- Model eğitilir - Veri kümesinin genellikle %80'i	- Model değerlendirilir - Veri kümesinin genellikle %20'si - Ayrıca tutma veya geliştirme kümesi olarak da adlandırılır	- Model tahminlerini verir - Görünmeyen veriler

Model seçildikten sonra, tüm veri kümesi üzerinde eğitilir ve görünmeyen test kümesinde test edilir. Bunlar aşağıdaki şekilde gösterilmektedir:

1.6 Gözetimsiz Öğrenme

Gözetimsiz öğrenme yöntemlerinin sınıfı, zengin gizli yapılara sahip olabilecek verilerin yapısını keşfetmeyi amaçlamaktadır.

1.6.1 k-ortalama

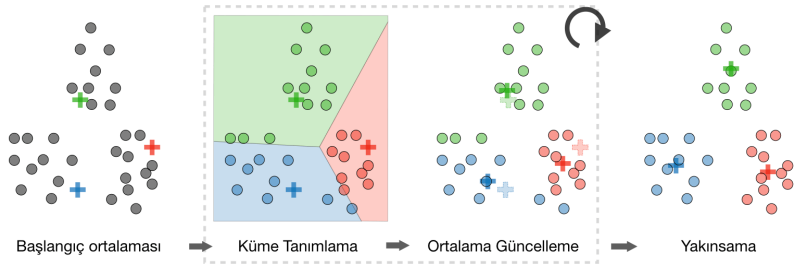
❒ **Kümeleme** – $\mathcal{D}_{\text{train}}$ giriş noktalarından oluşan bir eğitim kümesi göz önüne alındığında, kümeleme (clustering) algoritmasının amacı, her bir $\phi(x_i)$ noktasını $z_i \in \{1,...,k\}$ kümesine atamaktır.

❒ **Amaç fonksiyonu** – Ana kümeleme algoritmalarından biri olan k-ortalama için kayıp fonksiyonu şöyle ifade edilir:

$$\boxed{\text{Loss}_{\text{k-means}}(x,\mu) = \sum_{i=1}^{n} ||\phi(x_i) - \mu_{z_i}||^2}$$

❒ **Algoritma** – Küme merkezlerini $\mu_1,\mu_2,...,\mu_k \in \mathbb{R}^n$ kümesini rasgele başlattıktan sonra, k-ortalama algoritması yakınsayana kadar aşağıdaki adımı tekrarlar:

$$\boxed{z_i = \arg\min_j ||\phi(x_i) - \mu_j||^2} \quad \text{and} \quad \boxed{\mu_j = \frac{\sum_{i=1}^{m} 1_{\{z_i=j\}}\phi(x_i)}{\sum_{i=1}^{m} 1_{\{z_i=j\}}}}$$



1.6.2 Temel Bileşenler Analizi

❒ **Özdeğer, özvektör** – Bir $A \in \mathbb{R}^{n\times n}$ matrisi verildiğinde, $z \in \mathbb{R}^n\backslash\{0\}$ olacak şekilde bir vektör varsa λ, A'nın bir öz değeri olduğu söylenir, aşağıdaki gibi ifade edilir:

$$\boxed{Az = \lambda z}$$

❒ **Spektral teoremi** – $A \in \mathbb{R}^{n\times n}$ olsun. A simetrik ise, o zaman A gerçek ortogonal matris $U \in \mathbb{R}^{n\times n}$ olacak şekilde köşegenleştirilebilir. $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$ formülü dikkate alınarak aşağıdaki gibi ifade edilir:

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

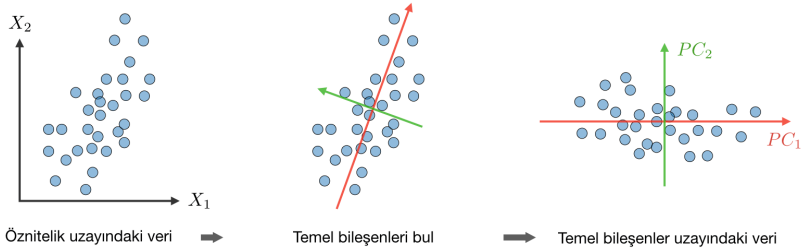
Not: en büyük özdeğerle ilişkilendirilen özvektör, A matrisinin temel özvektörüdür.

❒ **Algoritma** – Temel Bileşenler Analizi (PCA, principal component analysis) prosedürü, verilerin varyansını en üst düzeye çıkararak k boyutlarına indirgeyen bir boyut küçültme tekniğidir:

- Adım 1: Verileri ortalama 0 ve 1 standart sapma olacak şekilde normalize edin.

$$\boxed{x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}} \quad \text{where} \quad \boxed{\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}} \quad \text{and} \quad \boxed{\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2}$$

- Adım 2: Hesaplama $\Sigma = \frac{1}{m}\sum_{i=1}^{m}\phi(x_i)\phi(x_i)^T \in \mathbb{R}^{n\times n}$, ki bu, gerçek özdeğerlerle simetriktir.
- Adım 3: Hesaplama $u_1,...,u_k \in \mathbb{R}^n$ k'nin ortogonal ana özvektörleri, yani k en büyük özdeğerlerin ortogonal özvektörleri.
- Adım 4: $\text{span}_{\mathbb{R}}(u_1,...,u_k)$'daki verilerin izdüşümünü al. Bu prosedür, tüm k boyutlu uzaylar arasındaki farkı en üst düzeye çıkarır.



2 Durum-temelli modeller

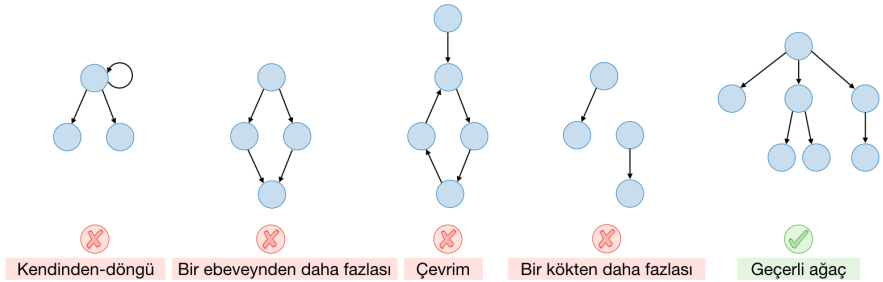
Cemal Gurpinar ve Başak Buluz tarafından çevrilmiştir

2.1 Arama optimizasyonu

Bu bölümde, s durumunda a eylemini gerçekleştirdiğimizde, Succ(s,a) durumuna varacağımızı varsayıyoruz. Burada amaç, başlangıç durumundan başlayıp bitiş durumuna götüren bir eylem dizisi (a_1,a_2,a_3,a_4,...) belirlenmesidir. Bu tür bir problemi çözmek için, amacımız durum-temelli modelleri kullanarak asgari maliyet yolunu bulmak olacaktır.

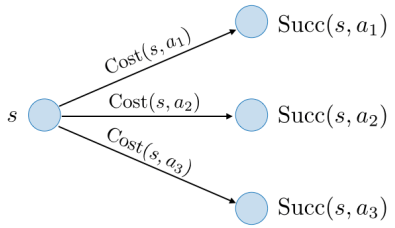
2.1.1 Ağaç arama

Bu durum-temelli algoritmalar, olası bütün durum ve eylemleri araştırırlar. Oldukça bellek verimli ve büyük durum uzayları için uygundurlar ancak çalışma zamanı en kötü durumlarda üstel olabilir.



❒ **Arama problemi** – Bir arama problemi aşağıdaki şekilde tanımlanmaktadır:

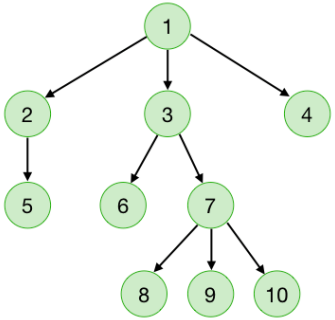
- bir başlangıç durumu s_{start}
- s durumunda gerçekleşebilecek olası eylemler Actions(s)
- s durumunda gerçekleşen a eyleminin eylem maliyeti Cost(s,a)
- a eyleminden sonraki varılacak durum Succ(s,a)
- son duruma ulaşılıp ulaşılamadığı IsEnd(s)



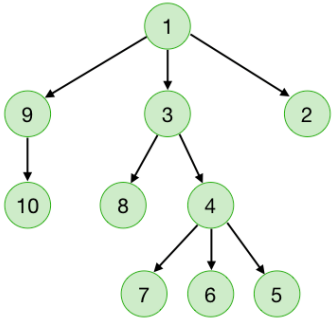
Amaç, maliyeti en aza indiren bir yol bulmaktır.

❒ **Geri izleme araması** – Geri izleme araması (backtracking search), asgari maliyet yolunu bulmak için tüm olasılıkları deneyen saf (naive) bir özyinelemeli algoritmadır. Burada, eylem maliyetleri pozitif ya da negatif olabilir.

❒ **Genişlik öncelikli arama (BFS)** – Genişlik öncelikli arama (BFS, breadth-first search), seviye seviye arama yapan bir çizge arama algoritmasıdır. Gelecekte her adımda ziyaret edilecek düğümleri tutan bir kuyruk yardımıyla yinelemeli olarak gerçekleyebiliriz. Bu algoritma için, eylem maliyetlerinin belirli bir sabite $c \geqslant 0$ eşit olduğunu kabul edebiliriz.



❒ **Derinlik öncelikli arama (DFS)** – Derinlik öncelikli arama (DFS, depth-first search), her bir yolu olabildiğince derin bir şekilde takip ederek çizgeyi dolaşan bir arama algoritmasıdır. Bu algoritmayı, ziyaret edilecek gelecek düğümleri her adımda bir yığın yardımıyla saklayarak, yinelemeli (recursively) ya da tekrarlı (iteratively) olarak uygulayabiliriz. Bu algoritma için eylem maliyetlerinin 0 olduğu varsayılmaktadır.



❒ **Tekrarlı derinleşme** – Tekrarlı derinleşme (iterative deepening) hilesi, derinlik-ilk arama algoritmasının değiştirilmiş bir halidir, böylece belirli bir derinliğe ulaştıktan sonra durur, bu da tüm işlem maliyetleri eşit olduğunda en iyiliği (optimal) garanti eder. Burada, işlem maliyetlerinin $c \geqslant 0$ gibi sabit bir değere eşit olduğunu varsayıyoruz.

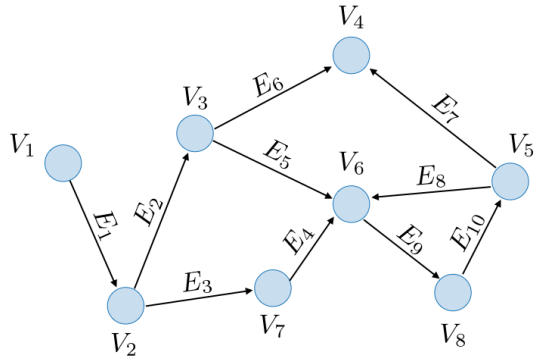
❒ **Ağaç arama algoritmaları özeti** – b durum başına eylem sayısını, d çözüm derinliğini ve D en yüksek (maksimum) derinliği ifade ederse, o zaman:

Algoritma	Eylem maliyetleri	Arama uzayı	Zaman
Geri izleme araması	herhangi bir şey	$\mathcal{O}(D)$	$\mathcal{O}(b^D)$
Genişlik öncelikli arama	$c \geqslant 0$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^d)$
Derinlik öncelikli arama	0	$\mathcal{O}(D)$	$\mathcal{O}(b^D)$
DFS-tekrarlı derinleşme	$c \geqslant 0$	$\mathcal{O}(d)$	$\mathcal{O}(b^d)$

2.1.2 Çizge arama

Bu durum-temelli algoritmalar kategorisi, üssel tasarruf sağlayan en iyi (optimal) yolları oluşturmayı amaçlar. Bu bölümde, dinamik programlama ve tek tip maliyet araştırması üzerinde duracağız.

❒ **Çizge** – Bir çizge, V köşeler (düğüm olarak da adlandırılır) kümesi ile E kenarlar (bağlantı olarak da adlandırılır) kümesinden oluşur.

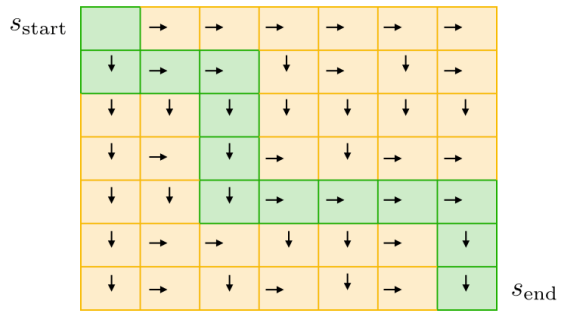


Not: çevrim olmadığında, bir çizgenin asiklik (çevrimsiz) olduğu söylenir.

❒ **Durum** – Bir durum gelecekteki eylemleri en iyi (optimal) şekilde seçmek için, yeterli tüm geçmiş eylemlerin özetidir.

❒ **Dinamik programlama** – Dinamik programlama (DP, dynamic programming), amacı s durumundan bitiş durumu olan s_{end}'e kadar asgari maliyet yolunu bulmak olan hatırlamalı (memoization) (başka bir deyişle kısmi sonuçlar kaydedilir) bir geri izleme (backtracking) arama algoritmasıdır. Geleneksel çizge arama algoritmalarına kıyasla üstel olarak tasarruf sağlayabilir ve yalnızca asiklik (çevrimsiz) çizgeler ile çalışma özelliğine sahiptir. Herhangi bir durum için gelecekteki maliyet aşağıdaki gibi hesaplanır:

$$\text{FutureCost}(s) = \begin{cases} 0 & \text{eğer IsEnd}(s) \\ \min\limits_{a \in \text{Actions}(s)} \left[\text{Cost}(s,a) + \text{FutureCost}(\text{Succ}(s,a))\right] & \text{aksi taktirde} \end{cases}$$

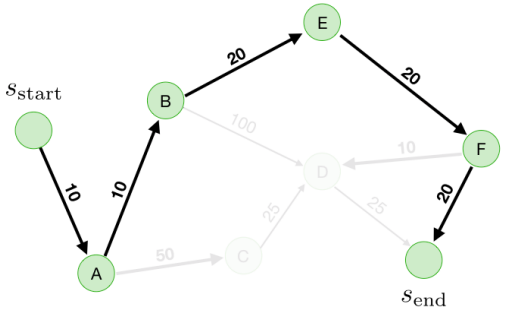


Not: yukarıdaki şekil, aşağıdan yukarıya bir yaklaşımı sergilerken, formül ise yukarıdan aşağıya bir önsezi ile problem çözümü sağlar.

❒ **Durum türleri** – Tek tip maliyet araştırması bağlamındaki durumlara ilişkin terminoloji aşağıdaki tabloda sunulmaktadır:

Durum	Açıklama
Keşfedilmiş $\mathcal{E}$	En iyi (optimal) yolun daha önce bulunduğu durumlar
Sırada $\mathcal{F}$	Görülen ancak hala en ucuza nasıl gidileceği hesaplanmaya çalışılan durumlar
Keşfedilmemiş $\mathcal{U}$	Daha önce görülmeyen durumlar

❒ **Tek tip maliyet araması** – Tek tip maliyet araması (UCS, uniform cost search) bir başlangıç durumu olan s_{start}, ile bir bitiş durumu olan s_{end} arasındaki en kısa yolu bulmayı amaçlayan bir arama algoritmasıdır. Bu algoritma s durumlarını artan geçmiş maliyetleri olan PastCost(s)'a göre araştırır ve eylem maliyetlerinin negatif olmayacağı kuralına dayanır.



Not 1: UCS algoritması mantıksal olarak Dijkstra algoritması ile aynıdır.
Not 2: algoritma, negatif eylem maliyetleriyle ilgili bir problem için çalışmaz ve negatif olmayan bir hale getirmek için pozitif bir sabit eklemek problemi çözmez, çünkü problem farklı bir problem haline gelmiş olur.

❒ **Doğruluk teoremi** – s durumu sıradaki (frontier) $\mathcal{F}$'den çıkarılır ve daha önceden keşfedilmiş olan $\mathcal{E}$ kümesine taşınırsa, önceliği başlangıç durumu olan s_{start}'dan, s durumuna kadar asgari maliyet yolu olan PastCost(s)'e eşittir.

❒ **Çizge arama algoritmaları özeti** – N toplam durumların sayısı, n-bitiş durumu s_{end}'ndan önce keşfedilen durum sayısı ise:

Algoritma	Asiklik	Maliyetler	Zaman/arama uzayı
Dinamik programlama	evet	herhangi bir şey	$\mathcal{O}(N)$
Tek tip maliyet araması	değil	$c \geqslant 0$	$\mathcal{O}(n\log(n))$

Not: karmaşıklık geri sayımı, her durum için olası eylemlerin sayısını sabit olarak kabul eder.

2.1.3 Öğrenme maliyetleri

Diyelim ki, Cost(s,a) değerleri verilmedi ve biz bu değerleri maliyet yolu eylem dizisini, $(a_1, a_2, ..., a_k)$, en aza indiren bir eğitim kümesinden tahmin etmek istiyoruz.

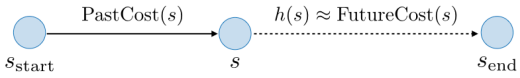
❒ **Yapılandırılmış algılayıcı** – Yapılandırılmış algılayıcı, her bir durum-eylem çiftinin maliyetini tekrarlı (iteratively) olarak öğrenmeyi amaçlayan bir algoritmadır. Her bir adımda, algılayıcı:

- eğitim verilerinden elde edilen gerçek asgari y yolunun her bir durum-eylem çiftinin tahmini (estimated) maliyetini azaltır,
- öğrenilen ağırlıklardan elde edilen şimdiki tahmini(predicted) y' yolununun durum-eylem çiftlerinin tahmini maliyetini artırır.

Not: algoritmanın birkaç sürümü vardır, bunlardan biri problemi sadece her bir a eyleminin maliyetini öğrenmeye indirger, bir diğeri ise öğrenilebilir ağırlık öznitelik vektörünü, Cost(s,a)'nın parametresi haline getirir.

2.1.4 A* arama

❒ **Sezgisel işlev** – Sezgisel (heuristic), s durumu üzerinde işlem yapan bir h fonksiyonudur, burada her bir $h(s)$, s ile s_{end} arasındaki yol maliyeti olan FutureCost(s)'yi tahmin etmeyi amaçlar.



❒ **Algoritma** – A^* s durumu ile s_{end} bitiş durumu arasındaki en kısa yolu bulmayı amaçlayan bir arama algoritmasıdır. Bahse konu algoritma PastCost(s) + $h(s)$'yi artan sıra ile araştırır. Aşağıda verilenler ışığında kenar maliyetlerini de içeren tek tip maliyet aramasına eşittir:

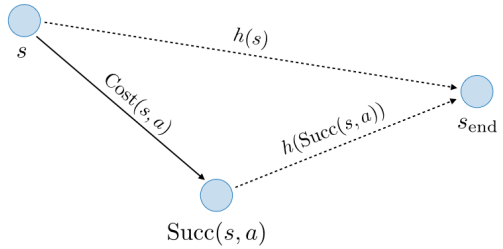
$$\boxed{\text{Cost}'(s,a) = \text{Cost}(s,a) + h(\text{Succ}(s,a)) - h(s)}$$

Not: bu algoritma, son duruma yakın olduğu tahmin edilen durumları araştıran tek tip maliyet aramasının taraflı bir sürümü olarak görülebilir.

❒ **Tutarlılık** – Bir sezgisel h, aşağıdaki iki özelliği sağlaması durumunda tutarlıdır denilebilir:

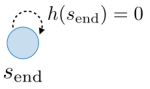
- Bütün s durumları ve a eylemleri için,

$$\boxed{h(s) \leqslant \text{Cost}(s,a) + h(\text{Succ}(s,a))}$$



- Bitiş durumu aşağıdakileri doğrular:

$$\boxed{h(s_{\text{end}}) = 0}$$



❒ **Doğruluk** – Eğer h tutarlı ise o zaman A^* algoritması asgari maliyet yolunu döndürür.

❒ **Kabul edilebilirlik** – Bir sezgisel h kabul edilebilirdir eğer:

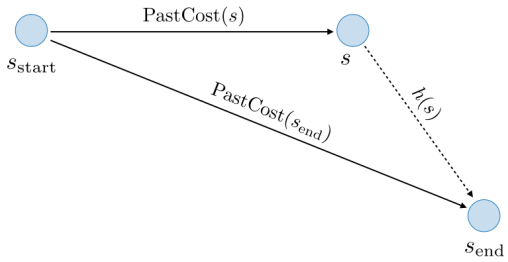
$$\boxed{h(s) \leqslant \text{FutureCost}(s)}$$

❒ **Teorem** – $h(s)$ sezgisel olsun ve:

$$\boxed{h(s) \text{ tutarlı} \implies h(s) \text{ kabul edilebilir}}$$

❒ **Verimlilik** – A^* algoritması aşağıdaki eşitliği sağlayan bütün s durumlarını araştırır:

$$\boxed{\text{PastCost}(s) \leqslant \text{PastCost}(s_{\text{end}}) - h(s)}$$



Not: $h(s)$'nin yüksek değerleri, bu eşitliğin araştırılacak olan s durum kümesini kısıtlayacak olması nedeniyle daha iyidir.

2.1.5 Rahatlama

Bu tutarlı sezgisel için bir altyapıdır. Buradaki fikir, kısıtlamaları kaldırarak kapalı şekilli (closed-form) düşük maliyetler bulmak ve bunları sezgisel olarak kullanmaktır.

❒ **Rahat arama problemi** – Cost maliyetli bir arama probleminin rahatlaması, Cost_{rel} maliyetli P_{rel} ile ifade edilir ve kimliği karşılar:

$$\boxed{\text{Cost}_{\text{rel}}(s,a) \leqslant \text{Cost}(s,a)}$$

❒ **Rahat sezgisel** – Bir P_{rel} rahat arama problemi verildiğinde, $h(s) = \text{FutureCost}_{\text{rel}}(s)$ rahat sezgisel (relaxed heuristic) eşitliğini $\text{Cost}_{\text{rel}}(s,a)$ maliyet çizgesindeki s durumu ile bir bitiş durumu arasındaki asgari maliyet yolu olarak tanımlarız.

❒ **Rahat sezgisel tutarlılığı** – P_{rel} bir rahat problem olarak verilmiş olsun. Teoreme göre:

$$\boxed{h(s) = \text{FutureCost}_{\text{rel}}(s) \implies h(s) \text{ tutarlı}}$$

❒ **Sezgisel seçiminde ödünleşim** – Sezgisel seçiminde iki yönü dengelemeliyiz:

- Hesaplamalı verimlilik: $h(s) = \text{FutureCost}_{\text{rel}}(s)$ eşitliği kolay hesaplanabilir olmalıdır. Kapalı bir şekil, daha kolay arama ve bağımsız alt problemler üretmesi gerekir.
- Yeterince iyi yaklaşım: sezgisel $h(s)$, $\text{FutureCost}(s)$ işlevine yakın olmalı ve bu nedenle çok fazla kısıtlamayı ortadan kaldırmamalıyız.

❒ **En yüksek sezgisel** – $h_1(s)$ ve $h_2(s)$ aşağıdaki özelliklere sahip iki adet sezgisel olsun:

$$\boxed{h_1(s),\, h_2(s) \text{ tutarlı} \implies h(s) = \max\{h_1(s),\, h_2(s)\} \text{ tutarlı}}$$

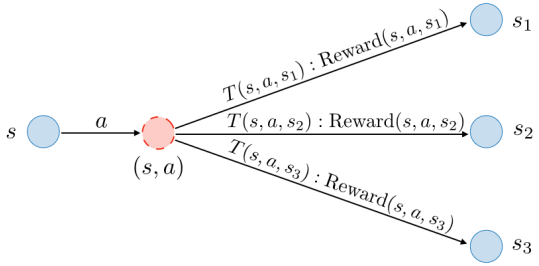
2.2 Markov karar süreçleri

Bu bölümde, s durumunda a eyleminin gerçekleştirilmesinin olasılıksal olarak birden fazla durum s'_1,s'_2,..., ile sonuçlanacağını kabul ediyoruz. Başlangıç durumu ile bitiş durumu arasındaki yolu bulmak için amacımız, rastgelelilik ve belirsizlik ile başa çıkabilmek için yardımcı olan Markov karar süreçlerini kullanarak en yüksek değer politikasını bulmak olacaktır.

2.2.1 Gösterimler

❒ **Tanım** – Markov karar sürecinin (MDP, Markov decision process) amacı ödülleri en yüksek seviyeye çıkarmaktır. Markov karar süreci aşağıdaki bileşenlerden oluşmaktadır:

- başlangıç durumu s_{start}
- s durumunda gerçekleştirilebilecek olası eylemler $\text{Actions}(s)$
- s durumunda a eyleminin gerçekleştirilmesi ile s' durumuna geçiş olasılıkları $T(s,a,s')$
- s durumunda a eyleminin gerçekleştirilmesi ile elde edilen ödüller $\text{Reward}(s,a,s')$
- bitiş durumuna ulaşılıp ulaşılamadığı $\text{IsEnd}(s)$
- indirim faktörü $0 \leqslant \gamma \leqslant 1$



❒ **Geçiş olasılıkları** – Geçiş olasılığı $T(s,a,s')$ s durumundayken gerçekleştirilen a eylemi neticesinde s' durumuna gitme olasılığını belirtir. Her bir $s' \mapsto T(s,a,s')$ aşağıda belirtildiği gibi bir olasılık dağılımıdır:

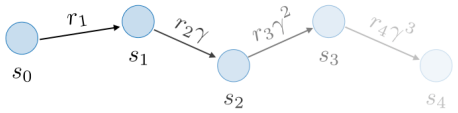
$$\forall s,a, \quad \boxed{\sum_{s' \in \text{ States}} T(s,a,s') = 1}$$

❒ **Politika** – Bir π politikası her s durumunu bir a eylemi ile ilişkilendiren bir işlevdir:

$$\boxed{\pi : s \mapsto a}$$

❒ **Fayda** – Bir $(s_0, ..., s_k)$ yolunun faydası, o yol üzerindeki ödüllerin indirimli toplamıdır. Diğer bir deyişle,

$$\boxed{u(s_0,...,s_k) = \sum_{i=1}^{k} r_i \gamma^{i-1}}$$



Yukarıdaki şekil $k = 4$ durumunun bir gösterimidir.

❒ **Q-değeri** – s durumunda gerçekleştirilen bir a eylemi için π politikasının Q-değeri (Q-value), $Q_\pi(s,a)$ olarak da gösterilir, a eylemini gerçekleştirip ve sonrasında π politikasını takiben s durumundan beklenen faydadır. Q-değeri aşağıdaki şekilde tanımlanmaktadır:

$$\boxed{Q_\pi(s,a) = \sum_{s' \in \text{ States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_\pi(s')\right]}$$

❒ **Bir politikanın değeri** – s durumundaki π politikasının değeri, $V_\pi(s)$ olarak da gösterilir, rastgele yollar üzerinde s durumundaki π politikasını izleyerek elde edilen beklenen faydadır. s durumundaki π politikasının değeri aşağıdaki gibi tanımlanır:

$$\boxed{V_\pi(s) = Q_\pi(s,\pi(s))}$$

Not: eğer s bitiş durumu ise $V_\pi(s)$ sıfıra eşittir.

2.2.2 Uygulamalar

❒ **Politika değerlendirme** – Bir π politikası verildiğinde, politika değerlendirmesini (policy evaluation), V_π, tahmin etmeyi amaçlayan bir tekrarlı (iterative) algoritmadır. Politika değerlendirme aşağıdaki gibi yapılmaktadır:

- <u>İlklendirme</u>: bütün s durumları için

$$\boxed{V_\pi^{(0)}(s) \longleftarrow 0}$$

- <u>Tekrar</u>: 1'den T_{PE}'ye kadar her t için, ile

$$\forall s, \quad \boxed{V_\pi^{(t)}(s) \longleftarrow Q_\pi^{(t-1)}(s,\pi(s))}$$

ile

$$\boxed{Q_\pi^{(t-1)}(s,\pi(s)) = \sum_{s' \in \text{ States}} T(s,\pi(s),s') \left[\text{Reward}(s,\pi(s),s') + \gamma V_\pi^{(t-1)}(s')\right]}$$

Not: S durum sayısını, A her bir durum için eylem sayısını, S' ardılların (successors) sayısını ve T yineleme sayısını gösterdiğinde, zaman karmaşıklığı $\mathcal{O}(T_{PE}SS')$ olur.

❒ **En iyi Q-değeri** – s durumunda a eylemi gerçekleştirildiğinde bu durumun en iyi Q-değeri, $Q_{\text{opt}}(s,a)$, herhangi bir politika başlangıcında elde edilen en yüksek Q-değeri olarak tanımlanmaktadır. En iyi Q-değeri aşağıdaki gibi hesaplanmaktadır:

$$\boxed{Q_{\text{opt}}(s,a) = \sum_{s' \in \text{ States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}(s')\right]}$$

❒ **En iyi değer** – s durumunun en iyi değeri olan $V_{\text{opt}}(s)$, herhangi bir politika ile elde edilen en yüksek değer olarak tanımlanmaktadır. En iyi değer aşağıdaki gibi hesaplanmaktadır:

$$\boxed{V_{\text{opt}}(s) = \max_{a \in \text{ Actions}(s)} Q_{\text{opt}}(s,a)}$$

❒ **En iyi politika** – En iyi politika olan π_{opt}, en iyi değerlere götüren politika olarak tanımlanmaktadır. En iyi politika aşağıdaki gibi tanımlanmaktadır:

$$\forall s, \quad \boxed{\pi_{\text{opt}}(s) = \operatorname*{argmax}_{a \in \text{ Actions}(s)} Q_{\text{opt}}(s,a)}$$

❒ **Değer tekrarı** – Değer tekrarı (value iteration) en iyi politika olan π_{opt}, yanında en iyi değeri V_{opt}'ı, bulan bir algoritmadır. Değer tekrarı aşağıdaki gibi yapılmaktadır:

- <u>İlklendirme</u>: bütün s durumları için

$$\boxed{V_{\text{opt}}^{(0)}(s) \longleftarrow 0}$$

- <u>Tekrar</u>: 1'den T_{VI}'ya kadar her bir t için:

$$\forall s, \quad \boxed{V_{\text{opt}}^{(t)}(s) \longleftarrow \max_{a \in \text{ Actions}(s)} Q_{\text{opt}}^{(t-1)}(s,a)}$$

with

$$\boxed{Q_{\text{opt}}^{(t-1)}(s,a) = \sum_{s' \in \text{ States}} T(s,a,s') \left[\text{Reward}(s,a,s') + \gamma V_{\text{opt}}^{(t-1)}(s')\right]}$$

Not: eğer $\gamma < 1$ ya da Markov karar süreci asiklik olursa, o zaman değer tekrarı algoritmasının doğru cevaba yakınsayacağı garanti edilir.

2.2.3 Bilinmeyen geçişler ve ödüller

Şimdi, geçiş olasılıklarının ve ödüllerin bilinmediğini varsayalım.

❒ **Model-temelli Monte Carlo** – Model-temelli Monte Carlo yöntemi, $T(s,a,s')$ ve $\text{Reward}(s,a,s')$ işlevlerini Monte Carlo benzetimi kullanarak aşağıdaki formüllere uygun bir şekilde tahmin etmeyi amaçlar:

$$\boxed{\widehat{T}(s,a,s') = \frac{\#\text{ kere } (s,a,s') \text{ gerçekleşme sayısı}}{\#\text{ kere } (s,a) \text{ gerçekleşme sayısı}}}$$

ve

$$\boxed{\widehat{\text{Reward}}(s,a,s') = r \text{ içinde } (s,a,r,s')}$$

Bu tahminler daha sonra Q_π ve Q_{opt}'yi içeren Q-değerleri çıkarımı için kullanılacaktır.

Not: model-tabanlı Monte Carlo'nun politika dışı olduğu söyleniyor, çünkü tahmin kesin politikaya bağlı değildir.

❒ **Model içermeyen Monte Carlo** – Model içermeyen Monte Carlo yöntemi aşağıdaki şekilde doğrudan Q_π'yi tahmin etmeyi amaçlar:

$$\boxed{\widehat{Q}_\pi(s,a) = \text{ortalama } u_t,\ s_{t-1} = s \text{ ve } a_t = a \text{ olduğunda}}$$

u_t belirli bir bölümün t anında başlayan faydayı ifade etmektedir.

Not: model içermeyen Monte Carlo'nun politikaya dahil olduğu söyleniyor, çünkü tahmini değer veriyi üretmek için kullanılan π politikasına bağlıdır.

❒ **Eşdeğer formülasyon** – Sabit tanımı $\eta = \frac{1}{1+(\#\text{güncelleme sayısı } (s,a))}$ ve eğitim kümesinin her bir (s,a,u) üçlemesi için, model içermeyen Monte Carlo'nun güncelleme kuralı dışbükey bir kombinasyon formülasyonuna sahiptir:

$$\boxed{\widehat{Q}_\pi(s,a) \leftarrow (1-\eta)\widehat{Q}_\pi(s,a) + \eta u}$$

olasılıksal bayır formülasyonu yanında:

$$\boxed{\widehat{Q}_\pi(s,a) \leftarrow \widehat{Q}_\pi(s,a) - \eta(\widehat{Q}_\pi(s,a) - u)}$$

❒ **SARSA** – Durum-eylem-ödül-durum-eylem (SARSA, State-Action-Reward-State-Action), hem ham verileri hem de güncelleme kuralının bir parçası olarak tahminleri kullanarak Q_π'yi tahmin eden bir destekleme yöntemidir. Her bir (s,a,r,s',a') için:

$$\boxed{\widehat{Q}_\pi(s,a) \longleftarrow (1-\eta)\widehat{Q}_\pi(s,a) + \eta\left[r + \gamma\widehat{Q}_\pi(s',a')\right]}$$

Not: the SARSA tahmini, tahminin yalnızca bölüm sonunda güncellenebildiği model içermeyen Monte Carlo yönteminin aksine anında güncellenir.

❒ **Q-öğrenme** – Q-öğrenme (Q-learning), Q_{opt} için tahmin üreten politikaya dahil olmayan bir algoritmadır. Her bir (s,a,r,s',a') için:

$$\boxed{\widehat{Q}_{\text{opt}}(s,a) \leftarrow (1-\eta)\widehat{Q}_{\text{opt}}(s,a) + \eta\left[r + \gamma \max_{a' \in \text{ Actions}(s')} \widehat{Q}_{\text{opt}}(s',a')\right]}$$

❒ **Epsilon-açgözlü** – Epsilon-açgözlü politika, ϵ olasılıkla araştırmayı ve $1-\epsilon$ olasılıkla sömürüyü dengeleyen bir algoritmadır. Her bir s durumu için, π_{act} politikası aşağıdaki şekilde hesaplanır:

$$\boxed{\pi_{\text{act}}(s) = \begin{cases} \underset{a \in \text{ Actions}}{\text{argmax}}\ \widehat{Q}_{\text{opt}}(s,a) & \text{olasılıkla } 1-\epsilon \\ \text{Actions}(s) \text{ eylem kümesi içinden rastgele} & \text{olasılıkla } \epsilon \end{cases}}$$

2.3 Oyun oynama

Oyunlarda (örneğin satranç, tavla, Go), başka oyuncular vardır ve politikamızı oluştururken göz önünde bulundurulması gerekir.

❒ **Oyun ağacı** – Oyun ağacı, bir oyunun olasılıklarını tarif eden bir ağaçtır. Özellikle, her bir düğüm, oyuncu için bir karar noktasıdır ve her bir kökten (root) yaprağa (leaf) giden yol oyunun olası bir sonucudur.

❒ **İki oyunculu sıfır toplamlı oyun** – Her durumun tamamen gözlendiği ve oyuncuların sırayla oynadığı bir oyundur. Aşağıdaki gibi tanımlanır:

- bir başlangıç durumu s_{start}
- s durumunda gerçekleştirilebilecek olası eylemler $\text{Actions}(s)$
- durumunda a eylemi gerçekleştirildiğindeki ardıllar $\text{Succ}(s,a)$
- bir bitiş durumuna ulaşılıp ulaşılmadığı $\text{IsEnd}(s)$
- s bitiş durumunda etmenin elde ettiği fayda $\text{Utility}(s)$
- s durumunu kontrol eden oyuncu $\text{Player}(s)$

Not: oyuncu faydasının işaretinin, rakibinin faydasının tersi olacağını varsayacağız.

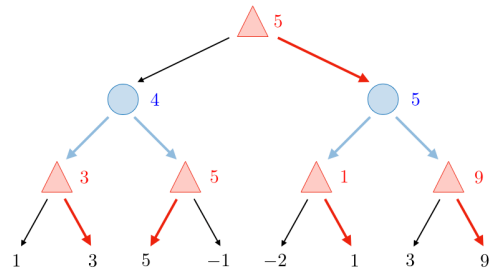
❒ **Politika türleri** – İki tane politika türü vardır:

- $\pi_p(s)$ olarak gösterilen belirlenimci politikalar, p oyuncusunun s durumunda gerçekleştirdiği eylemler.
- $\pi_p(s,a) \in [0,1]$ olarak gösterilen olasılıksal politikalar, p oyuncusunun s durumunda a eylemini gerçekleştirme olasılıkları.

❒ **Expectimax** – Belirli bir s durumu için, en yüksek beklenen değer olan $V_{\text{exptmax}}(s)$, sabit ve bilinen bir rakip politikası olan π_{opp}'a göre oynarken, bir oyuncu politikasının en yüksek beklenen faydasıdır. En yüksek beklenen değer (expectimax) aşağıdaki gibi hesaplanmaktadır:

$$\boxed{V_{\text{exptmax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max\limits_{a \in \text{Actions}(s)} V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \sum\limits_{a \in \text{Actions}(s)} \pi_{\text{opp}}(s,a) V_{\text{exptmax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}}$$

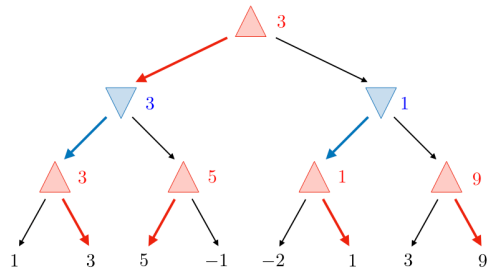
Not: en yüksek beklenen değer, MDP'ler için değer yinelemenin analog halidir.



❒ **Minimax** – En küçük-enbüyük (minimax) politikaların amacı en kötü durumu kabul ederek, diğer bir deyişle; rakip, oyuncunun faydasını en aza indirmek için her şeyi yaparken, rakibe karşı en iyi politikayı bulmaktır. En küçük-en büyük aşağıdaki şekilde yapılır:

$$\boxed{V_{\text{minimax}}(s) = \begin{cases} \text{Utility}(s) & \text{IsEnd}(s) \\ \max\limits_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{agent} \\ \min\limits_{a \in \text{Actions}(s)} V_{\text{minimax}}(\text{Succ}(s,a)) & \text{Player}(s) = \text{opp} \end{cases}}$$

Not: π_{max} ve π_{min} değerleri, en küçük-en büyük olan $V_{minimax}$'dan elde edilebilir.



❒ **Minimax özellikleri** – V değer fonksiyonunu ifade ederse, en küçük-en büyük ile ilgili aklımızda bulundurmamız gereken 3 özellik vardır:

- *Özellik 1*: oyuncu politikasını herhangi bir π_{agent} ile değiştirecek olsaydı, o zaman oyuncu daha iyi olmazdı.

$$\boxed{\forall \pi_{\text{agent}}, \quad V(\pi_{\max},\pi_{\min}) \geqslant V(\pi_{\text{agent}},\pi_{\min})}$$

- *Özellik 2*: eğer rakip oyuncu politikasını $\pi_{\min}$'den π_{opp}'a değiştirecek olsaydı, o zaman rakip oyuncu daha iyi olamazdı.

$$\boxed{\forall \pi_{\text{opp}}, \quad V(\pi_{\max},\pi_{\min}) \leqslant V(\pi_{\max},\pi_{\text{opp}})}$$

- *Özellik 3*: eğer rakip oyuncunun muhalif (adversarial) politikayı oynamadığı biliniyorsa, o zaman en küçük-en büyük politika oyuncu için ey iyi (optimal) olmayabilir.

$$\boxed{\forall \pi, \quad V(\pi_{\max},\pi) \leqslant V(\pi_{\text{exptmax}},\pi)}$$

Sonunda, aşağıda belirtildiği gibi bir ilişkiye sahip oluruz:

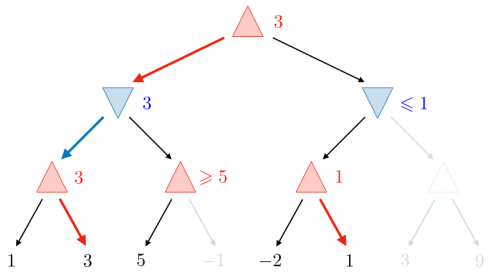
$$\boxed{V(\pi_{\text{exptmax}},\pi_{\min}) \leqslant V(\pi_{\max},\pi_{\min}) \leqslant V(\pi_{\max},\pi) \leqslant V(\pi_{\text{exptmax}},\pi)}$$

2.3.1 Minimax hızlandırma

❒ **Değerlendirme işlevi** – Değerlendirme işlevi, alana özgü ve $V_{\text{minimax}}(s)$ değerinin yaklaşık bir tahminidir. Eval(s) olarak ifade edilmektedir.

Not: FutureCost(s) arama problemleri için bir benzetmedir.

❒ **Alpha-beta budama** – Alpha-beta budama (alpha-beta pruning), oyun ağacının parçalarının gereksiz yere keşfedilmesini önleyerek en küçük-en büyük algoritmasını en iyileyen (optimize eden) alana-özgü olmayan genel bir yöntemdir. Bunu yapmak için, her oyuncu ümit edebileceği en iyi değeri takip eder (maksimize eden oyuncu için α'da ve minimize eden oyuncu için β'de saklanır). Belirli bir adımda, $\beta < \alpha$ koşulu, önceki oyuncunun emrinde daha iyi bir seçeneğe sahip olması nedeniyle en iyi (optimal) yolun mevcut dalda olamayacağı anlamına gelir.



❒ **TD öğrenme** – Geçici fark (TD, temporal difference) öğrenmesi, geçiş/ödülleri bilmediğimiz zaman kullanılır. Değer, keşif politikasına dayanır. Bunu kullanabilmek için, oyununun kurallarını, Succ(s,a), bilmemiz gerekir. Her bir (s,a,r,s') için, güncelleme aşağıdaki şekilde yapılır:

$$\boxed{w \longleftarrow w - \eta\Big[V(s,w) - (r + \gamma V(s',w))\Big]\nabla_w V(s,w)}$$

2.3.2 Eşzamanlı oyunlar

Bu, oyuncunun hamlelerinin sıralı olmadığı sıra temelli oyunların tam tersidir.

❒ **Tek-hamleli eşzamanlı oyun** – Olası hareketlere sahip A ve B iki oyuncu olsun. $V(a,b)$, A'nın a eylemini ve B'nin de b eylemini seçtiği A'nın faydasını ifade eder. V, getiri dizeyi (payoff matrix) olarak adlandırılır.

❒ **Stratejiler** – İki tane ana strateji türü vardır:

- Saf strateji, tek bir eylemdir:

$$\boxed{a \in \text{Actions}}$$

- Karışık strateji, eylemler üzerindeki bir olasılık dağılımıdır:

$$\forall a \in \text{Actions}, \quad \boxed{0 \leqslant \pi(a) \leqslant 1}$$

❒ **Oyun değerlendirme** – Oyuncu A π_A'yı ve oyuncu B de π_B'yi izlediğinde, oyun değeri $V(\pi_A,\pi_B)$:

$$\boxed{V(\pi_A,\pi_B) = \sum_{a,b} \pi_A(a)\pi_B(b)V(a,b)}$$

❒ **Minimax teoremi** – π_A, π_B'nin karma stratejilere göre değiştiğini belirterek, sonlu sayıda eylem ile eşzamanlı her iki oyunculu sıfır toplamlı oyun için:

$$\boxed{\max_{\pi_A}\min_{\pi_B} V(\pi_A,\pi_B) = \min_{\pi_B}\max_{\pi_A} V(\pi_A,\pi_B)}$$

2.3.3 Sıfır toplamı olmayan oyunlar

❒ **Getiri matrisi** – $V_p(\pi_A,\pi_B)$'yi oyuncu p'nin faydası olarak tanımlıyoruz.

❒ **Nash dengesi** – Nash dengesi (π_A^*,π_B^*) öyle birşey ki hiçbir oyuncuyu, stratejisini değiştirmeye teşvik etmiyor:

$$\boxed{\forall \pi_A, V_A(\pi_A^*,\pi_B^*) \geqslant V_A(\pi_A,\pi_B^*)} \quad \text{and} \quad \boxed{\forall \pi_B, V_B(\pi_A^*,\pi_B^*) \geqslant V_B(\pi_A^*,\pi_B)}$$

Not: sonlu sayıda eylem olan herhangi bir sonlu oyunculu oyunda, en azından bir tane Nash denegesi mevcuttur.

3 Değişken-temelli modeller

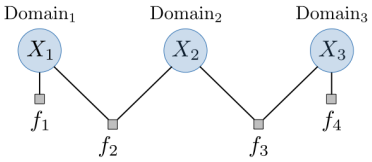
Başak Buluz ve Ayyüce Kızrak tarafından çevrilmiştir

3.1 Kısıt memnuniyet problemleri

Bu bölümde hedefimiz değişken-temelli modellerin maksimum ağırlık seçimlerini bulmaktır. Durum temelli modellerle kıyaslandığında, bu algoritmaların probleme özgü kısıtları kodlamak için daha uygun olmaları bir avantajdır.

3.1.1 Faktör grafikleri

❒ **Tanımlama** – Markov rasgele alanı olarak da adlandırılan faktör grafiği, $X_i \in \text{Domain}_i$ ve herbir $f_j(X) \geqslant 0$ olan $f_1,...,f_m$ m faktör olmak üzere $X = (X_1,...,X_n)$ değişkenler kümesidir.



❒ **Kapsam ve ilişki derecesi** – f_j faktörünün kapsamı, dayandığı değişken kümesidir. Bu kümenin boyutuna ilişki derecesi (arity) denir.

Not: faktörlerin ilişki derecesi 1 ve 2 olanlarına sırasıyla tek ve ikili denir.

❒ **Atama ağırlığı** – Her atama $x = (x_1,...,x_n)$, o atamaya uygulanan tüm faktörlerin çarpımı olarak tanımlanan bir Weight(x) ağırlığı verir. Şöyle ifade edilir:

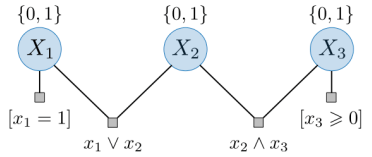
$$\boxed{\text{Weight}(x) = \prod_{j=1}^{m} f_j(x)}$$

❒ **Kısıt memnuniyet problemi** – Kısıtlama memnuniyet problemi (CSP, constraint satisfaction problem), tüm faktörlerin ikili olduğu bir faktör grafiğidir; bunları kısıt olarak adlandırıyoruz:

$$\boxed{\forall j \in [\![1,m]\!], \quad f_j(x) \in \{0,1\}}$$

Burada, j kısıtlı x ataması ancak ve ancak $f_j(x) = 1$ olduğunda uygundur denir.

❒ **Tutarlı atama** – Bir CSP'nin bir x atamasının, yalnızca Weight$(x) = 1$ olduğunda, yani tüm kısıtların yerine getirilmesi durumunda tutarlı olduğu söylenir.



3.1.2 Dinamik düzenleşim

❒ **Bağımlı faktörler** – X_i değişkeninin kısmi atamaya sahip bağımlı x değişken faktörlerinin kümesi $D(x,X_i)$ ile gösterilir ve X_i'yi önceden atanmış değişkenlere bağlayan faktörler kümesini belirtir.

❒ **Geri izleme araması** – Geri izleme araması (backtracking search), bir faktör grafiğinin maksimum ağırlık atamalarını bulmak için kullanılan bir algoritmadır. Her adımda, atanmamış bir değişken seçer ve değerlerini özyineleme ile arar. Dinamik düzenleşim (yani değişkenlerin ve değerlerin seçimi) ve bakış açısı (yani tutarsız seçeneklerin erken elenmesi), en kötü durum çalışma süresi üssel olarak olsa da grafiği daha verimli aramak için kullanılabilir: $O(|\text{Domain}|^n)$.

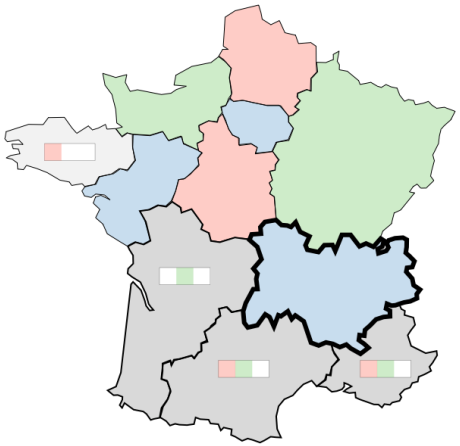
❒ **İleri kontrol** – Tutarsız değerleri komşu değişkenlerin etki alanlarından öncelikli bir şekilde ortadan kaldıran sezgisel bakış açısıdır. Aşağıdaki özelliklere sahiptir:

- Bir X_i değişkenini atadıktan sonra, tüm komşularının etki alanlarından tutarsız değerleri eler.
- Bu etki alanlardan herhangi biri boş olursa, yerel geri arama araması durdurulur.
- Eğer X_i değişkenini atamazsak, komşularının etki alanını eski haline getirilmek zorundadır.

❒ **En kısıtlı değişken** – En az tutarlı değere sahip bir sonraki atanmamış değişkeni seçen, değişken seviyeli sezgisel düzenleşimdir. Bu, daha verimli budama olanağı sağlayan aramada daha önce başarısız olmak için tutarsız atamalar yapma etkisine sahiptir.

❒ **En düşük kısıtlı değer** – Komşu değişkenlerin en yüksek tutarlı değerlerini elde ederek bir sonrakini veren seviye düzenleyici sezgisel bir değerdir. Sezgisel olarak, bu prosedür önce çalışması en muhtemel olan değerleri seçer.

Not: uygulamada, bu sezgisel yaklaşım tüm faktörler kısıtlı olduğunda kullanışlıdır.



Yukarıdaki örnek, en kısıtlı değişken keşfi ve sezgisel en düşük kısıtlı değerin yanı sıra, her adımda ileri kontrol ile birleştirilmiş geri izleme arama ile 3 renk probleminin bir gösterimidir.

❒ **Ark tutarlılığı** – X_l değişkeninin ark tutarlılığının (arc consistency) X_k'ye göre her bir $x_l \in \text{Domain}_l$ için geçerli olduğu söylenir:

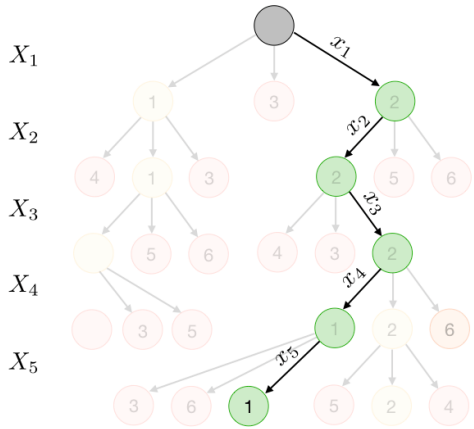
- X_l'in birleşik faktörleri sıfır olmadığında,
- en az bir $x_k \in \text{Domain}_k$ vardır, öyle ki X_l ve X_k arasında sıfır olmayan herhangi bir faktör vardır.

❒ **AC-3** – AC-3 algoritması, tüm ilgili değişkenlere ileri kontrol uygulayan çok adımlı sezgisel bir bakış açısıdır. Belirli bir görevden sonra ileriye doğru kontrol yapar ve ardından işlem sırasında etki alanının değiştiği değişkenlerin komşularına göre ark tutarlılığını ardı ardına uygular.

Not: AC-3, tekrarlı ve özyinelemeli olarak uygulanabilir.

3.1.3 Yaklaşık yöntemler

❒ **Işın araması** – Işın araması (beam search), her adımda K en üst yollarını keşfederek, $b = |\text{Domain}|$ dallanma faktörünün n değişkeninin kısmi atamalarını genişleten yaklaşık bir algoritmadır.
Aşağıdaki örnek, $K = 2$, $b = 3$ ve $n = 5$ parametreleri ile muhtemel ışın aramasını göstermektedir.



Not: $K = 1$ açgözlü aramaya (greedy search) karşılık gelirken $K \to +\infty$, BFS ağaç aramasına eşdeğerdir.

❒ **Tekrarlanmış koşullu modlar** – Tekrarlanmış koşullu modlar (ICM, iterated conditional modes), yakınsamaya kadar bir seferde bir değişkenli bir faktör grafiğinin atanmasını değiştiren yinelemeli bir yaklaşık algoritmadır. i adımında, X_i'ye, bu değişkene bağlı tüm faktörlerin çarpımını maksimize eden v değeri atanır.

Not: ICM yerel minimumda takılıp kalabilir.

❒ **Gibbs örneklemesi** – Gibbs örneklemesi (Gibbs sampling), yakınsamaya kadar bir seferde bir değişken grafik faktörünün atanmasını değiştiren yinelemeli bir yaklaşık yöntemdir. i adımında:

- her bir $u \in \text{Domain}_i$ olan öğeye, bu değişkene bağlı tüm faktörlerin çarpımı olan bir ağırlık $w(u)$ atanır,
- v'yi w tarafından indüklenen olasılık dağılımından örnek alır ve X_i'ye atanır.

Not: Gibbs örneklemesi, ICM'nin olasılıksal karşılığı olarak görülebilir. Çoğu durumda yerel minimumlardan kaçabilme avantajına sahiptir.

3.1.4 Faktör grafiği dönüşümleri

❒ **Bağımsızlık** – A, B, X değişkenlerinin bir bölümü olsun. A ve B arasında kenar yoksa A ve B'nin bağımsız olduğu söylenir ve şöyle ifade edilir:

$$\boxed{A,B \text{ bağımsız } \iff A \perp\!\!\!\perp B}$$

Not: bağımsızlık, alt sorunları paralel olarak çözmemize olanak sağlayan bir kilit özelliktir.

❒ **Koşullu bağımsızlık** – Eğer C'nin şartlandırılması, A ve B'nin bağımsız olduğu bir grafik üretiyorsa A ve B verilen C koşulundan bağımsızdır. Bu durumda şöyle yazılır:

$$\boxed{A \perp\!\!\!\perp B|C}$$

❒ **Koşullandırma** – Koşullandırma, bir faktör grafiğini paralel olarak çözülebilen ve geriye doğru izlemeyi kullanabilen daha küçük parçalara bölen değişkenleri bağımsız kılmayı amaçlayan bir dönüşümdür. $X_i = v$ değişkeninde koşullandırmak için aşağıdakileri yaparız:

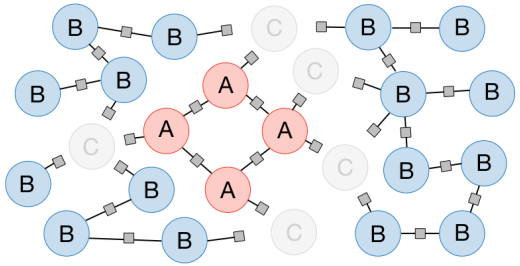
- X_i'ye bağlı tüm f_1,...,f_k faktörlerini göz önünde bulundurun
- X_i ve f_1,...,f_k öğelerini kaldırın
- $j \in \{1,...,k\}$ için $g_j(x)$ ekleyin:

$$\boxed{g_j(x) = f_j(x \cup \{X_i : v\})}$$

❒ **Markov blanket** – $A \subseteq X$ değişkenlerin bir alt kümesi olsun. MarkovBlanket(A)'i, A'da olmayan A'nın komşuları olarak tanımlıyoruz.

❒ **Önerme** – $C = \text{MarkovBlanket}(A)$ ve $B = X \backslash (A \cup C)$ olsun. Bu durumda:

$$\boxed{A \perp\!\!\!\perp B|C}$$



❒ **Eliminasyon** – Eliminasyon, X_i'yi grafikten ayıran ve Markov blanket de şartlandırılmış küçük bir alt sorunu çözen bir faktör grafiği dönüşümüdür:

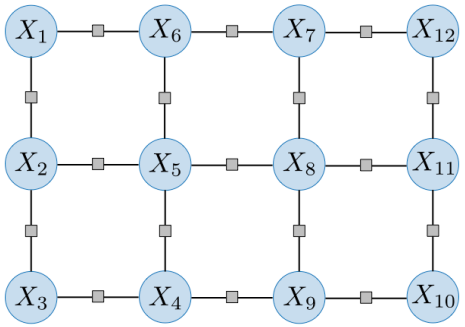
- X_i'ye bağlı tüm $f_{i,1}$,...,$f_{i,k}$ faktörlerini göz önünde bulundurun
- X_i ve $f_{i,1}$,...,$f_{i,k}$ kaldır
- $f_{\text{new},i}(x)$ ekleyin, şöyle tanımlanır:

$$\boxed{f_{\text{new},i}(x) = \max_{x_i} \prod_{l=1}^{k} f_{i,l}(x)}$$

❒ **Ağaç genişliği** – Bir faktör grafiğinin ağaç genişliği (treewidth), değişken elemeli en iyi değişken sıralamasıyla oluşturulan herhangi bir faktörün maksimum ilişki derecesidir. Diğer bir deyişle,

$$\boxed{\text{Treewidth} = \min_{\text{orderings}} \max_{i \in \{1,...,n\}} \text{arity}(f_{\text{new},i})}$$

Aşağıdaki örnek, ağaç genişliği 3 olan faktör grafiğini gösterir.



Not: en iyi değişken sıralamasını bulmak NP-zor (NP-hard) bir problemdir.

3.2 Bayesçi ağlar

Bu bölümün amacı koşullu olasılıkları hesaplamak olacaktır. Bir sorgunun kanıt verilmiş olma olasılığı nedir?

3.2.1 Giriş

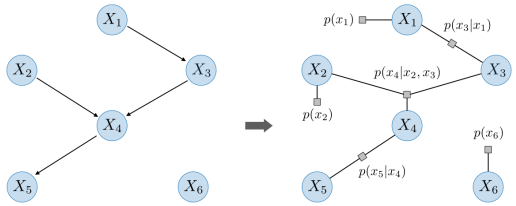
❒ **Açıklamalar** – C_1 ve C_2 sebeplerinin E etkisini yarattığını varsayalım. E etkisinin durumu ve sebeplerden biri (C_1 olduğunu varsayalım) üzerindeki etkisi, diğer sebep olan C_2'nin olasılığını değiştirir. Bu durumda, C_1'in C_2'yi açıkladığı söylenir.

❒ **Yönlü çevrimsiz çizge** – Yönlü çevrimsiz bir çizge (DAG, directed acyclic graph), yönlendirilmiş çevrimleri olmayan sonlu bir yönlü çizgedir.

❒ **Bayesçi ağ** – Her düğüm için bir tane olmak üzere, yerel koşullu dağılımların bir çarpımı olarak, $X = (X_1,...,X_n)$ rasgele değişkenleri üzerindeki bir ortak dağılımı belirten yönlü bir çevrimsiz çizgedir:

$$\boxed{P(X_1 = x_1,...,X_n = x_n) \triangleq \prod_{i=1}^{n} p(x_i|x_{\text{Parents}(i)})}$$

Not: Bayesçi ağlar olasılık diliyle bütünleşik faktör grafikleridir.



❒ **Yerel olarak normalleştirilmiş** – Her $x_{\text{Parents}(i)}$ için tüm faktörler yerel koşullu dağılımlardır. Bu nedenle yerine getirmek zorundalar:

$$\boxed{\sum_{x_i} p(x_i|x_{\text{Parents}(i)}) = 1}$$

Sonuç olarak, alt-Bayesçi ağlar ve koşullu dağılımlar tutarlıdır.

Not: yerel koşullu dağılımlar gerçek koşullu dağılımlardır.

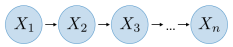
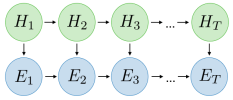
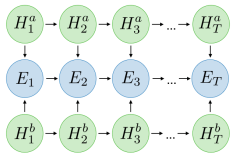
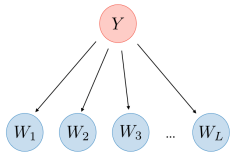
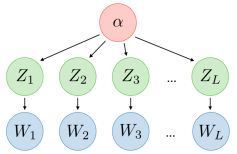
❒ **Marjinalleşme** – Bir yaprak düğümünün marjinalleşmesi, o düğüm olmaksızın bir Bayesçi ağı sağlar.

3.2.2 Olasılık programları

❒ **Konsept** – Olasılıklı bir program değişkenlerin atanmasını randomize eder. Bu şekilde, ilişkili olasılıkları açıkça belirtmek zorunda kalmadan atamalar üreten karmaşık Bayesçi ağlar yazılabilir.

Not: Olasılık programlarına örnekler arasında Gizli Markov modeli (HMM, hidden Markov model), faktöriyel HMM, naif Bayes (naive Bayes), gizli Dirichlet tahsisi (LDA, latent Dirichlet allocation), hastalıklar ve semptomları belirtirler ve stokastik blok modelleri bulunmaktadır.

❒ **Özet** – Aşağıdaki tablo, ortak olasılıklı programları ve bunların uygulamalarını özetlemektedir:

Program	Algoritma	Gösterim	Örnek
Markov Modeli	$X_i \sim p(X_i\|X_{i-1})$		Dil modelleme
Gizli Markov Modeli (HMM)	$H_t \sim p(H_t\|H_{t-1})$ $E_t \sim p(E_t\|H_t)$		Nesne izleme
Faktöriyel HMM	$H_t^o \underset{o\in\{a,b\}}{\sim} p(H_t^o\|H_{t-1}^o)$ $E_t \sim p(E_t\|H_t^a,H_t^b)$		Çoklu nesne izleme
Naif Bayes	$Y \sim p(Y)$ $W_i \sim p(W_i\|Y)$		Belge sınıflandırma
Gizli Dirichlet Tahsisi (LDA)	$\alpha \in \mathbb{R}^K$ dağılım $Z_i \sim p(Z_i\|\alpha)$ $W_i \sim p(W_i\|Z_i)$		Konu modelleme

3.2.3 Çıkarım

❒ **Genel olasılıksal çıkarım stratejisi** – $E = e$ kanıtı verilen Q sorgusunun $P(Q|E = e)$ olasılığını hesaplama stratejisi aşağıdaki gibidir:

- Adım 1: Q sorgusunun ataları olmayan değişkenlerini ya da marjinalleştirme yoluyla E kanıtını silin
- Adım 2: Bayesçi ağı faktör grafiğine dönüştürün
- Adım 3: kanıtın koşulu $E = e$
- Adım 4: Q sorgusu ile bağlantısı kesilen düğümleri marjinalleştirme yoluyla silin
- Adım 5: olasılıklı bir çıkarım algoritması çalıştırın (kılavuz, değişken eleme, Gibbs örneklemesi, parçacık filtreleme)

❒ **İleri-geri algoritma** – Bu algoritma, L boyutunda bir HMM durumunda herhangi bir $k \in \{1, ..., L\}$ için $P(H = h_k|E = e)$ (düzeltme sorgusu) değerini hesaplar. Bunu yapmak için 3 adımda ilerlenir:

- Adım 1: için $i \in \{1,...,L\}$, hesapla $F_i(h_i) = \sum_{h_{i-1}} F_{i-1}(h_{i-1})p(h_i|h_{i-1})p(e_i|h_i)$
- Adım 2: için $i \in \{L,...,1\}$, hesapla $B_i(h_i) = \sum_{h_{i+1}} B_{i+1}(h_{i+1})p(h_{i+1}|h_i)p(e_{i+1}|h_{i+1})$
- Adım 3: için $i \in \{1,...,L\}$, hesapla $S_i(h_i) = \frac{F_i(h_i)B_i(h_i)}{\sum_{h_i} F_i(h_i)B_i(h_i)}$

$F_0 = B_{L+1} = 1$ kuralı ile. Bu prosedürden ve bu notasyonlardan anlıyoruz ki

$$\boxed{P(H = h_k|E = e) = S_k(h_k)}$$

Not: bu algoritma, her bir atamada her bir kenarın $h_{i-1} \to h_i$'nin $p(h_i|h_{i-1})p(e_i|h_i)$ olduğu bir yol olduğunu yorumlar.

❒ **Gibbs örneklemesi** – Bu algoritma, büyük olasılık dağılımını temsil etmek için küçük bir dizi atama (parçacık) kullanan tekrarlı bir yaklaşık yöntemdir. Rasgele bir x atamasından Gibbs örneklemesi, $i \in \{1,...,n\}$ için yakınsamaya kadar aşağıdaki adımları uygular:

- Tüm $u \in \text{Domain}_i$ için, x atamasının $w(u)$ ağırlığını hesaplayın, burada $X_i = u$
- Örnekleme w : $v \sim P(X_i = v|X_{-i} = x_{-i})$ ile uyarılmış olasılık dağılımından
- Set $X_i = v$

Not: X_{-i}, $X\backslash\{X_i\}$ ve x_{-i}, karşılık gelen atamayı temsil eder.

❒ **Parçacık filtreleme** – Bu algoritma, bir seferde K parçacıklarını takip ederek gözlem değişkenlerinin kanıtı olarak verilen durum değişkenlerinin önceki yoğunluğuna yaklaşır. K boyutunda bir C parçacığı kümesinden başlayarak, aşağıdaki 3 adım tekrarlı olarak çalıştırılır:

- Adım 1: teklif - Her eski parçacık $x_{t-1} \in C$ için, geçiş olasılığı dağılımından $p(x|x_{t-1})$ örnek x'i alın ve C' ye ekleyin.
- Adım 2: ağırlıklandırma - C' nin her x değerini $w(x) = p(e_t|x)$ ile ağırlıklandırın, burada e_t t zamanında gözlemlenen kanıttır.
- Adım 3: yeniden örnekleme - w ile indüklenen olasılık dağılımını kullanarak C' kümesinden örnek K elemanlarını C cinsinden saklayın: bunlar şuanki x_t parçacıklarıdır.

Not: bu algoritmanın daha pahalı bir versiyonu da teklif adımındaki geçmiş katılımcıların kaydını tutar.

❒ **Maksimum olabilirlik** – Yerel koşullu dağılımları bilmiyorsak, maksimum olasılık kullanarak bunları öğrenebiliriz.

$$\boxed{\max_\theta \prod_{x \in \mathcal{D}_{\text{train}}} p(X = x; \theta)}$$

❒ **Laplace yumuşatma** – Her d dağılımı ve $(x_{\text{Parents}(i)},x_i)$ kısmi ataması için, $\text{count}_d(x_{\text{Parents}(i)},x_i)$'a λ ekleyin, ardından olasılık tahminlerini almak için normalleştirin.

❒ **Beklenti-Maksimizasyon** – Beklenti-Maksimizasyon (EM, expectation-maximization) algoritması, olasılığa art arda bir alt sınır oluşturarak (E-adım) tekrarlayarak ve bu alt sınırın (M-adımını) optimize ederek θ parametresini maksimum olasılık tahmini ile tahmin etmede aşağıdaki gibi etkin bir yöntem sunar:

- E-adım: her bir e veri noktasının belirli bir h kümesinden geldiği gerideki $q(h)$ durumunu şu şekilde değerlendirin:

$$\boxed{q(h) = P(H = h|E = e; \theta)}$$

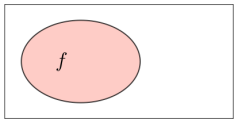
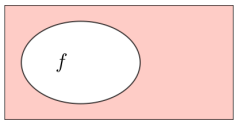
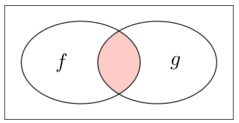
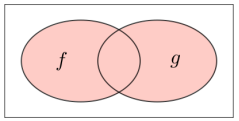
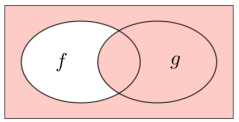
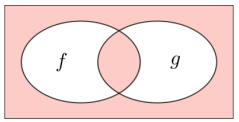
- M-adım: maksimum olasılığını belirlemek için e veri noktalarındaki küme özgül ağırlıkları olarak gerideki olasılıklar $q(h)$ kullanın.

4 Mantık-temelli modeller

Ayyüce Kızrak ve Başak Buluz tarafından çevrilmiştir

4.1 Temeller

❒ **Önerme mantığının sözdizimi** – f,g formülleri ve $\neg, \wedge, \vee, \to, \leftrightarrow$ bağlayıcılarını belirterek, aşağıdaki mantıksal ifadeleri yazabiliriz:

Adı	Sembol	Anlamı	Gösterim
Doğrulama	f	f	
Dışlayan	$\neg f$	f değil	
Kesişim	$f \wedge g$	f ve g	
Birleşim	$f \vee g$	f veya g	
Implication	$f \to g$	eğer f'den g çıkarsa	
İki koşullu	$f \leftrightarrow g$	f ve g'nin ortak olduğu bölge	

Not: bu bağlantılar dışında tekrarlayan formüller oluşturulabilir.

❒ **Model** – w modeli, ikili sembollerin önermeli sembollere atanmasını belirtir.

Örnek: $w = \{A : 0,B : 1,C : 0\}$ doğruluk değerleri kümesi, A, B ve C önermeli semboller için olası bir modeldir.

❒ **Yorumlama fonksiyonu** – Yorumlama fonksiyonu $\mathcal{I}(f,w)$, w modelinin f formülüne uygun olup olmadığını gösterir:

$$\boxed{\mathcal{I}(f,w) \in \{0,1\}}$$

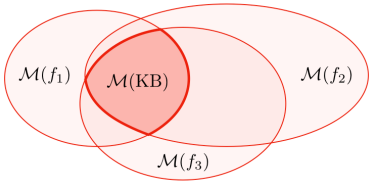
❒ **Modellerin seti** – $\mathcal{M}(f)$, f formülünü sağlayan model setini belirtir. Matematiksel konuşursak, şöyle tanımlarız:

$$\boxed{\forall w \in \mathcal{M}(f), \quad \mathcal{I}(f,w) = 1}$$

4.2 Bilgi temelli

❒ **Tanım** – Bilgi temeli KB (knowledge base), şu ana kadar düşünülen tüm formüllerin birleşimidir. Bilgi temelinin model kümesi, her formülü karşılayan model dizisinin kesişimidir. Diğer bir deyişle:

$$\boxed{\mathcal{M}(\text{KB}) = \bigcap_{f \in \text{KB}} \mathcal{M}(f)}$$



❒ **Olasılıksal yorumlama** – f sorgusunun 1 olarak değerlendirilmesi olasılığı, f'yi sağlayan bilgi temeli KB'nin w modellerinin oranı olarak görülebilir, yani:

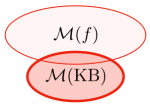
$$\boxed{P(f|\text{KB}) = \frac{\displaystyle\sum_{w \in \mathcal{M}(\text{KB}) \cap \mathcal{M}(f)} P(W = w)}{\displaystyle\sum_{w \in \mathcal{M}(\text{KB})} P(W = w)}}$$

❒ **Gerçeklenebilirlik** – En az bir modelin tüm kısıtlamaları yerine getirmesi durumunda KB'nin bilgi temelinin gerçeklenebilir olduğu söylenir. Diğer bir deyişle:

$$\boxed{\text{KB karşılanabilirlik} \iff \mathcal{M}(\text{KB}) \neq \varnothing}$$

Not: $\mathcal{M}(KB)$, bilgi temelinin tüm kısıtları ile uyumlu model kümesini belirtir.

❒ **Formüller ve bilgi temeli arasındaki ilişki** – Bilgi temeli KB ile yeni bir formül f arasında aşağıdaki özellikleri tanımlarız:

Adı	Matematiksel formülü	Gösterim	Notlar
KB f içerir	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \mathcal{M}(\text{KB})$		- f yeni bir bilgi getirmiyor - Ayrıca KB $\models f$ yazıyor
KB f içermez	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \varnothing$		- Hiçbir model f ekledikten sonra kısıtlamaları yerine getirmiyor - KB $\models \neg f$'ye eşdeğer
f koşullu KB	$\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \neq \varnothing$ ve $\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \neq \mathcal{M}(\text{KB})$		- f KB'ye aykırı değil - f KB'ye önemsiz miktarda bilgi ekliyor

❒ **Model denetimi** – Bir model denetimi algoritması, KB'nin bilgi temelini girdi olarak alır ve bunun karşılanabilir olup olmadığını çıkarır.

Not: popüler model kontrol algoritmaları DPLL ve WalkSat'ı içerir.

❒ **Çıkarım kuralı** – $f_1,...,f_k$ ve sonuç g yapısının çıkarım kuralı şöyle yazılmıştır:

$$\boxed{\frac{f_1,...,f_k}{g}}$$

❒ **İleri çıkarım algoritması** – Çıkarım kurallarından Rules, bu algoritma mümkün olan tüm$f_1,...,f_k$'den geçer ve eşleşen bir kural varsa, KB bilgi tabanına g ekler. Bu işlem KB'ye daha fazla ekleme yapılamayana kadar tekrar edilir.

❒ **Türetme** – f'nin KB içerisindeyse veya Rules kurallarını kullanarak ileri çıkarım algoritması sırasında eklenmişse, KB'nin Rules ile f (KB $\vdash f$ yazılır) türettiğini söylüyoruz.

❒ **Çıkarım kurallarının özellikleri** – Çıkarım kurallarının kümesi Rules aşağıdaki özelliklere sahip olabilir:

Adı	Matematiksel formülü	Notlar
Sağlamlık	$\{f : \text{KB} \vdash f\} \subseteq \{f : \text{KB} \models f\}$	- Çıkarılan formüller KB arafından sağlanmıştır - Her defasında bir kural kontrol edilebilir - "*Gerçeğinden başka bir şey yok*"
Tamlık	$\{f : \text{KB} \vdash f\} \supseteq \{f : \text{KB} \models f\}$	-Ya KB 'yi içeren formüller ya bilgi tabanında zaten vardır, ya da ondan çıkarılan değerlerdir - "*Tüm gerçek*"

4.3 Önerme mantığı

Bu bölümde, mantıksal formülleri ve çıkarım kurallarını kullanan mantık tabanlı modelleri inceleyeceğiz. Buradaki fikir ifade ve hesaplamanın verimliliğini dengelemektir.

❒ **Horn cümlesi** – p_1,...,p_k ve q önerme sembollerini not ederek, bir Horn cümlesi şu şekildedir (matematiksel mantık ve mantık programlamada, kural gibi özel bir biçime sahip mantıksal formüllere Horn cümlesi denir):

$$\boxed{(p_1 \wedge ... \wedge p_k) \longrightarrow q}$$

Not: q = false olduğunda, "hedeflenen bir cümle" olarak adlandırılır, aksi takdirde "kesin bir cümle" olarak adlandırırız.

❒ **Modus ponens** – f_1,...,f_k ve p önermeli semboller için modus ponens kuralı yazılır:

$$\boxed{\frac{f_1,...,f_k, \quad (f_1 \wedge ... \wedge f_k) \longrightarrow p}{p}}$$

Not: her uygulama tek bir önermeli sembol içeren bir cümle oluşturduğundan, bu kuralın uygulanması doğrusal bir zaman alır.

❒ **Tamlık** – KB'nin sadece Horn cümleleri içerdiğini ve p'nin zorunlu bir teklif sembolü olduğunu varsayalım, Hornus cümlelerine göre modus ponenleri tamamlanmıştır. Modus ponens uygulanması daha sonra p'yi türetir.

❒ **Konjunktif normal form** – Bir konjonktif normal form (CNF, conjunctive normal form) formülü, her bir cümlenin atomik formüllerin bir ayrıntısı olduğu cümle birleşimidir.

Açıklama: başka bir deyişle, CNF'ler $\vee$ ait $\wedge$ bulunmaktadır.

❒ **Eşdeğer temsil** – Önerme mantığındaki her formül eşdeğer bir CNF formülüne yazılabilir. Aşağıdaki tabloda genel dönüşüm özellikleri gösterilmektedir:

Kural adı		Başlangıç	Dönüştürülmüş
Eleme	$\leftrightarrow$	$f \leftrightarrow g$	$(f \rightarrow g) \wedge (g \rightarrow f)$
	$\rightarrow$	$f \rightarrow g$	$\neg f \vee g$
	$\neg\neg$	$\neg\neg f$	f
Dağıtma	$\neg$ üzerine $\wedge$	$\neg(f \wedge g)$	$\neg f \vee \neg g$
	$\neg$ üzerine $\vee$	$\neg(f \vee g)$	$\neg f \wedge \neg g$
	$\vee$ üzerine $\wedge$	$f \vee (g \wedge h)$	$(f \vee g) \wedge (f \vee h)$

❒ **Çözünürlük kuralı** – f_1,...,f_n ve g_1,...,g_m önerme sembolleri için, p, çözümleme kuralı yazılır:

$$\boxed{\frac{f_1 \vee ... \vee f_n \vee p, \quad \neg p \vee g_1 \vee ... \vee g_m}{f_1 \vee ... \vee f_n \vee g_1 \vee ... \vee g_m}}$$

Not: her uygulama, teklif sembollerinin alt kümesine sahip bir cümle oluşturduğundan, bu kuralı uygulamak için üssel olarak zaman alabilir.

❒ **Çözünürlük tabanlı çıkarım** – Çözünürlük tabanlı çıkarım algoritması, aşağıdaki adımları izler:

- Adım 1: Tüm formülleri CNF'ye dönüştürün
- Adım 2: Tekrar tekrar, çözünürlük kuralını uygulayın
- Adım 3: False türetilmişse tatmin edici olmayan dönüş yapın

4.4 Birinci dereceden mantık

Buradaki fikir, daha kompakt bilgi sunumları sağlamak için değişkenleri kullanmaktır.

❒ **Model** – Birinci mertebeden mantık haritalarında bir w modeli:

- nesnelere sabit semboller
- nesnelerin dizisini sembolize etmek için tahmin

❒ **Horn cümlesi** – x_1,...,x_n değişkenleri ve a_1,...,a_k,b atomik formüllerine dikkat çekerek, bir boynuz maddesinin birinci derece mantık versiyonu aşağıdaki şekildedir:

$$\boxed{\forall x_1,...,\forall x_n, \quad (a_1 \wedge ... \wedge a_k) \rightarrow b}$$

❒ **Yer değiştirme** – Bir yerdeğiştirme değişkenleri terimlerle eşler ve Subst(θ,f) yerdeğiştirme sonucunu f olarak belirtir.

❒ **Birleştirme** – Birleştirme f ve g'nin iki formülünü alır ve onları eşit yapan en genel ikameyi θ verir:

$$\boxed{\text{Unify}[f,g] = \theta} \quad \text{öyle ki} \quad \boxed{\text{Subst}[\theta,f] = \text{Subst}[\theta,g]}$$

Not: Unify[f,g] eğer böyle bir θ yoksa Fail döndürür.

❒ **Modus ponens** – x_1,...,x_n değişkenleri, a_1,...,a_k ve a'_1,...,a'_k atomik formüllerine dikkat ederek ve θ = Unify($a'_1 \wedge ... \wedge a'_k, a_1 \wedge ... \wedge a_k$) modus ponenlerin birinci dereceden mantık versiyonu yazılabilir:

$$\boxed{\frac{a'_1,...,a'_k \quad \forall x_1,...,\forall x_n (a_1 \wedge ... \wedge a_k) \rightarrow b}{\text{Subst}[\theta, b]}}$$

❒ **Tamlık** – Modus ponens sadece Horn cümleleriyle birinci dereceden mantık için tamamlanmıştır.

❒ **Resolution rule** – $f_1, ..., f_n, g_1, ..., g_m, p, q$ formüllerini not ederek ve θ = Unify(p,q) ifadesini kullanarak, çözümleme kuralının birinci dereceden mantık sürümü yazılabilir:

$$\boxed{\frac{f_1 \vee ... \vee f_n \vee p, \quad \neg q \vee g_1 \vee ... \vee g_m}{\text{Subst}[\theta, f_1 \vee ... \vee f_n \vee g_1 \vee ... \vee g_m]}}$$

❒ **Yarı-karar verilebilirlik** – Birinci dereceden mantık, sadece Horn cümleleriyle sınırlı olsa bile, yarı karar verilebilir eğer:

- KB $\models f$ ise f sonsuz zamanlıdır
- KB $\not\models f$ ise sonsuz zamanlı olabilirliği gösteren algoritma yoktur