BÖLÜM 25. KABA SEÇİM ALGORİTMASI

Kaba seçim algoritması, birden fazla safhadan oluşan bir algoritmadır. Her safhada, gelecekte oluşacak sonuçlara bakılmaksızın sonucu iyi olacağına inanılan bir karar verilir. Bunun anlamı, yerel optimum seçilmesidir. Bu algoritmanın mantığı "**şimdi ne alabiliyorsan al**" stratejisine dayanmaktadır. Algoritma bitime gittiğinde, yerel optimum, global optimuma eşit olması umulur. Eğer bu durum oluşuyorsa, algoritma doğrudur; aksi halde alt-optimal çözüm oluşuyor demektir. Eğer en iyi çözüm gerekli değilse, kaba seçim algoritması bir yaklaşık çözüm üretmesi amacıyla kullanılır ve bu durumda kullanılan algoritmalar fazla kompleks olmaz.

Gerçek hayatta kaba seçim algoritması ile çözülebilecek çok sayıda problem vardır. Bunlardan birincisi n lirayı elde ederken 25 kuruşluk, 50 kuruşluk, 1 liralık, 5 liralık ve 10 liralık paralardan minimum sayıda kullanılması problemidir. Bu problemin Türkiye para birimi için kaba seçim algoritması çalışırken, bazı ülkelerin para birimleri için çalışmaz.

Benzer şekilde yerel çözüm seçimi yaparak global çözüme ulaşması amacı trafik probleminde çalışmaz.

25.1. Kaba Seçim Stratejisinin Elemanları

Kaba seçim yönteminde, seçimler dizisi sonucunda çözüm elde edilir. Her seçim o anki en iyi seçimdir (Yerel en iyi). Eğer bir en iyileme problemi kaba seçim yöntemi ile çözülebiliyorsa, bu nasıl ifade edilebilir? Bunun bir genel yolu yoktur, fakat kaba seçim stratejisinin 2 önemli özelliği vardır.

- Kaba seçim özelliği
- Optimal altyapı

Kaba Seçim Özelliği

Yerel optimal seçimler yapılarak global optimale ulaşılabilir. Bu seçim yapılırken daha önceki seçimlere göre değil de o anda en iyi seçim hangisi ise o seçim yapılır. Bu seçim alt problemlerin çözümlerine veya gelecekteki seçimlere bağlı değildir. Kaba seçim yöntemi genelden-özele yöntemine göre çalışır. Her seçim sonucunda, gelecek seçim biraz daha küçülür.

Bir seçim yapıldığında problem biraz daha küçülür. Bundan dolayı tümevarımla her adımda kaba seçiminin uygulanabileceğinin gösterilmesi gerekir.

Optimal Altyapı

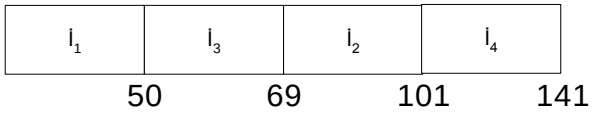
Eğer bir problemin çözümü, alt problemlerin optimal çözümlerini içeriyorsa, buna optimal altyapı denir. Optimal altyapı, kaba seçim algoritması ve dinamik programlama için temel teşkil eder. Bu özellik, problemin ara çözüm adımlarında global en iyi çözüm nedir tedirginliği yaşanmadan, o anda bulunan ara çözümler içerisinde en iyisi hangisi ise, o seçilir ve bunun sonucunda global en iyi çözüme ulaşılır. Bunun anlamı, altyapının çözümü global çözümü etkiliyor. Bu yöntemde global en iyi çözüme ulaşılacağı umuduyla, yerel en iyi çözüm seçilir.

25.2. Bir Basit Planlama Problemi

1 adet işlemcinin kullanıldığı bir ortamda $İ_1$, $İ_2$,..., $İ_N$ işlerinin sırası ile çalışma zamanları $z_1,z_2,...,z_N$ olsun. Ortalama çalışma zamanının minimum olması için bu işler bu işlemci üzerinde nasıl planlanmalıdır? Buradaki problemin özelliklerinden biri de başlayan bir iş bitime kadar kesintisiz olarak işlemci üzerinde icra edilmeye devam eder. Örneğin, dört tane iş ve bu işlerin çalışma zamanları Tablo 25.1' de görülmektedir. Bu işlerin bir işlemci üzerinde icra edilmeleri için gerekli olan bir plan Şekil 25.1'deki gibi olabilir ve ortalama zaman

Tablo 25.1. *İş ve zamanları*

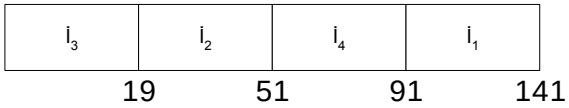
İş	Zaman
$İ_1$	50
$İ_2$	32
$İ_3$	19
$İ_4$	40



Şekil 25.1. *Birinci plan.*

$$\frac{50+69+101+141}{4}=90.25$$

olur. Optimum çözüm ise, Şekil 25.2' deki gibi olur ve bu planın ortalama zaman değeri



Şekil 25.2. *İkinci plan.*

$$\frac{19+51+91+141}{4}=75.5$$

olur. Optimum çözümün elde edildiği çözümde süresi kısa olan iş önce başlar mantığı kullanılmıştır. Bu mantığın, bu problem için her zaman optimal çözüm sağlanacağı ispatlanabilir. İş listesi $İ_1,İ_2,...,İ_N$ ve ilgili zaman listesi $z_1,z_2,...,z_N$ olmak üzere ilk iş z_1 zamanda biter; ikinci iş z_1+z_2 zamanda biter; üçüncü iş $z_1+z_2+z_3$ zamanda biter, bu şekilde devam edilerek toplam zaman (maliyet)

$$C = \sum_{k=1}^{N} (N - k + 1) z t_k$$
$$= (N+1) \sum_{k=1}^{N} z_k - \sum_{k=1}^{N} k z_k$$

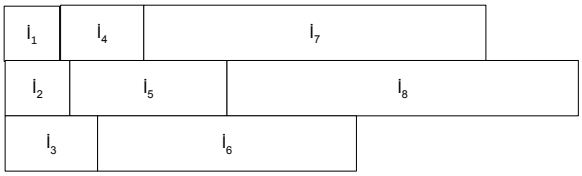
olur. İlk terim işlerin sırasından bağımsızdır ve ikinci terim ise, iş sırasından bağımsız olmadığından optimal çözüme etki eden kısımdır. Örneğin, $x>y$ ve $z_x<z_y$ olması durumunda $İ_x$ ile $İ_y$ işlerinin yer değiştirmesi sonucunda ikinci terimde artış meydana gelir. Böylece, herhangi bir iş planında zaman azalmayan monotonik değilse, elde edilen çözüm alt-optimaldir. Geriye kalan tek çözüm zamanı en kısa olan iş ile başlamaktır.

Bu problem birden fazla işlemcinin olduğu duruma genişletilebilir. İş listesi $İ_1,İ_2,...,İ_N$ ve ilgili zaman listesi sırası ile $z_1,z_2,...,z_N$ olan bir durumda P tane işlemci olsun. Bu durumda çalışma süresi kısa olan işlerin daha önce seçildiği kabul edilsin. Örneğin, işlemci sayısı P=3 iken, iş ve ilgili zaman listesi Tablo 25.2' de görülmektedir.

Tablo 25.2. *İş ve zamanları*

İş	Zaman
$İ_1$	7
$İ_2$	8
$İ_3$	10
$İ_4$	19
$İ_5$	27
$İ_6$	50
$İ_7$	63
$İ_8$	64

Bu işleri üç işlemci üzerinde planlama durumlarından biri Şekil 25.3' te görüldüğü gibidir.

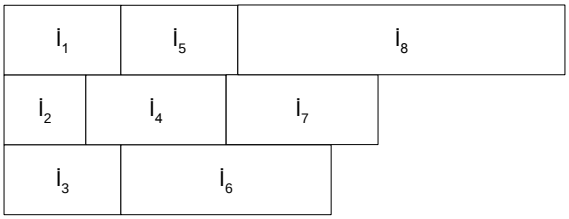


Şekil 25.3. *Üç işlemci için işlemlerin icra planı (birinci plan).*

Bu plan için ortalama zamanı

$$\frac{(7+26+89)+(8+35+99)+(10+60)}{8}=41{,}75$$

olur. Bu problem için diğer bir çözümde Şekil 25.4' te görüldüğü gibi olabilir.



Şekil 25.4. *Üç işlemci için işlemlerin icra planı (ikinci plan).*

İkinci planın ortalama zamanı

$$\frac{(7+34+98)+(8+27+90)+(10+60)}{8}=41{,}75$$

olur. Her iki planlamanın ortalama zamanı arasında çok küçük bir fark vardır. Birden fazla işlemci olduğu durumda, işler zamanlarına göre sıralanır ve sıra ile işlemcilere dağıtım yapılır. Bir çevrim bittikten sonra tekrar ilk başlanılan işlemciden başlanır ve bir çevrim daha bitirilir. Bu şekilde devam edilerek dağıtım işlemi gerçekleştirilir.

25.3. Maksimum Ağırlıklı Orman

G=(V,E) bir çizge olsun ve amaç toplam ayrıt ağırlığı maksimum olan bir orman elde etmektir ve G çizgesi, $\forall e_i \in E$ için $w(e_i) \geq 0$ olan bir çizgedir.

İlk olarak G çizgesi bağlı olsun ve bu çizgenin minimum açılım ağacı |V|-1 ayrıta sahip olacağından, elde edilecek ormanda en fazla |V|-1 ayrıt içerir. İşlemlerden kullanılacak olan uzaklık fonksiyonu

$d_{ij}=W-w_{ij}$, $\forall (v_i,v_j) \in E$ ve $W=\max_{i,j}\{w_{ij}\}$

şeklinde tanımlanır. T ağacı G çizgesinin açılım ağacı olsun. Toplam uzaklıklar d(T) şeklinde gösterilir ve toplam ağırlık w(T) ile ilişkilidir. Toplam ağırlık

$W(T)=(|V|-1)W-d(T)$

şeklinde bir kullanılan bir fonksiyon olsun. Buradan, minimum açılım ağacı d ile ilişkili iken, maksimum ağırlıklı ormanda w ile ilişkilidir. Eğer G çizgesi bağlı değilse, durum fazla değişmez. Bunun çözümü, her parçanın maksimum açılım ağacının bulunması ile elde edilir.

Algoritma 25.1. Orman(G)

▷ G yönsüz çizgesinden maksimum ağırlığa sahip bir çizge elde edilip.

```
1-  F←∅
2-  E≠∅ olduğu sürece
3-     (u,v)∈E ve ağırlığı maksimum olsun.
4-     E=E-{(u,v)}
5-  Eğer u ve v (V,F)' nin aynı parçasında değillerse,
6-     F=F∪{(u,v)}
```

$\{(u_1,T_1), (u_2,T_2), ..., (u_k,T_k)\}$, V kümesi üzerinde tanımlanmış orman olsun ve (v,u) ayrıtı maksimum ağırlığı olan u_1 terk eden ayrıt olsun.

$$T = \sum_{i=1}^{k} T_k$$

ağacını içeren bütün ormanlar arasında (v,u) ayrıtını içeren bir tane optimal orman vardır. Ormanı seçen algoritmamız algoritma 25.1' de görülmektedir.

25.4. Aktivite Seçme Problemi

Birden fazla yarışan aktivite arasında kaynak planlamasını yapılmak isteniyor.

Amaç: Eleman sayısı maksimum olan ve elemanları çarpışmayan kümesi elde etmek. S = {1,2,…,n} aktivite kümesi olsun. Kabul:(ortam kabulü) Herhangi bir anda sadece bir aktivite aktif olabilir. Her aktivitenin başlangıç (S_i) ve bitiş (f_i) zamanları belli ve $s_i \leq f_i$.

Eğer i aktivitesi seçilirse bu aktivite $[s_i,f_j)$ zaman aralığında yer alır. Eğer i ve j aktiviteleri için $[s_i,f_i) \cap [s_j,f_j) = 0$ veya $s_i \geq f_j$ veya $s_j \geq$ise i ve j aktiviteleri uyumlu aktivitelerdir. Eğer aktiviteler bitiş zamanlarına göre sıralı değillerse, bu işlem gerçekleştirilir ve zamanı $\Theta(nlgn)$ olur. Sıralama işleminin sonucunda aktivitelerin bitiş zamanları

$f_1 \leq f_2 \leq \ldots\ldots \leq f_n$

şeklinde olur. s ve f zaman dizileri olsun.

Bu algoritmanın $T(n)=\Theta(n)$ olur. Eğer sıralama işlemi de düşünülürse $T(n)=\Theta(nlgn)$ olur. Verilen algoritmanın doğru çalıştığı şu şekilde gösterilebilir. S={1,2,3,…,n} olsun. Aktiviteler bitiş zamanlarına göre sıralı olduğundan 1 aktivitesi en erken bitecek olan aktivitedir.

Bundan dolayı 1. aktivite ile başlanır (Kaba seçimi). A⊆S ve optimal çözüm olsun. A'nın ilk elemanı k olsun. Eğer k = 1 ise, A kaba seçimi ile başlıyor. Eğer k ≠ 1 ise , B⊆S olan ve ikinci bir optimal çözümün olduğu gösterilmelidir ve kaba seçimi ile başlamalıdır.

Algoritma 25.2. Aktivite(G)

▹Kaba seçim algoritması ile aktivite seçme problemini çözen algoritma. **A**:Seçilen aktiviteler kümesi, **j**: A ya en son yapılan eklemeyi gösterir, Bundan dolayı f_j en büyük bitiş zamanıdır ve $f_j = \max \{k : k \in A\}$.

```
1- n ← uzunluk[s]
2- A← {1}
3- j ← 1
4- i←2,...,n için
5-   eğer s_i ≥ f_j ise
6-     A ← A U {i}
7-     j ← i
8- sonuç←A
```

B=A–{k}∪{1} olursa , f_1, f_k olduğundan , B'deki elemanlar ayrık olur ve A ile aynı eleman sayısına sahiptir. Bundan dolayı B'de optimal our. Bu şekilde kaba seçimi ile başlayan optimal çözüm her zaman için vardır. Kaba seçimi ile 1 seçilir ve bunda sonraki adımda, eğer S probleminin optimal çözümü A ise, A' = A – {1} çözümü de S' ={i∈S : $s_i \geq f_1$ } probleminin optimal çözümü olur. Eğer S' problemine B' optimal çözümü ise, (|B'|>|A'|), B = B' U{1} çözümü S probleminin çözümü olur ve |B|>|A| olur. Bu durum A çözümünün optimal olması ile çelişir.

25.5. Huffman Kodları

Renksiz ve özelliksiz karakterlerden oluşan dosyaların sıkıştırılması için kullanılan kodlardır. Dosyadaki bir karakterin tekrar sayısına o

karakterin frekansı denir ve karakterlerin tekrar frekansından faydalanarak ikili kodlar üretilir. İki tip kodlama vardır: Sabit uzunluklu ve değişken uzunluklu

		a	b	c	d	e	f
Frekans	:	45	13	12	16	9	5
Sabit uzunluk	:	000	001	010	011	100	101
Değişken uzunluk	:	0	101	100	111	1101	1100

Şekil 25.5. *Karakterlerin frekansları ve kodlama durumları (sabit ve değişken uzunluklu).*

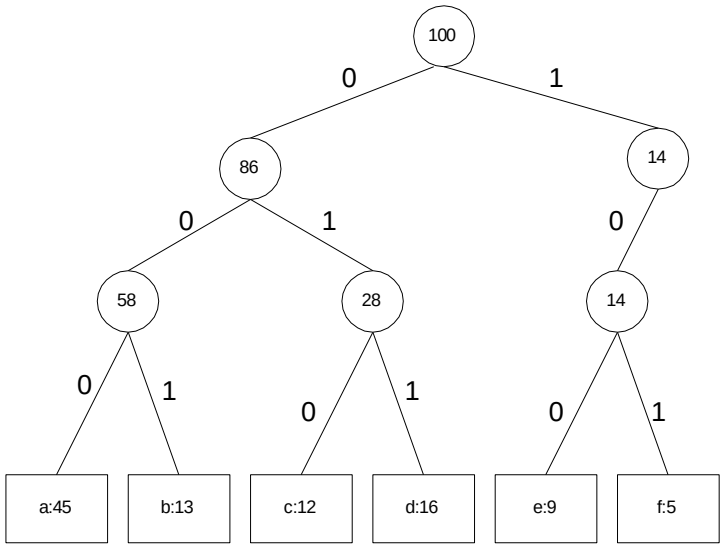
Değişken uzunluklu kodlamada, frekansı en yüksek olan karaktere en kısa kod ve diğer karakterlere benzer mantıkla kod verilirse, sabit uzunluklu kodlamaya göre daha başarılı bir sonuç elde edilir.

Kodlama ve kodlamayı çözmek için ön ek kodlamadan faydalanılır. Kodlama her zaman için daha kolaydır. Kodu çözmek, sabit uzunluklu kodlamada oldukça kolaydır; değişken uzunluklu kodlamada ise, biraz daha zahmetlidir. Sıkıştırılmış bir dosyanın kod ağacı tam dolu ikili ağaç ise, bu sıkıştırma işlemi optimaldir; aksi halde optimal değildir.

Sabit uzunluklu kodlama optimal değildir, çünkü ağacı tam dolu ikili ağaç değildir.

Eğer ağaç C alfabesinde üretilmiş ise, bu durumda yaprak sayısının |C| olması gerekir ve ara düğümlerde |C|-1 olmalıdır.

$c \in C$ olsun ve f(c) bu karakterin frekansını göstersin. T ağacıda ön ek kod ağacı olsun. $d_T(c)$ ise c karakterinin T ağacındaki derinliği olsun ve aynı zamanda c karakterinin kod uzunluğunu gösterir. Dosyanın kodlandıktan sonraki uzunluğu

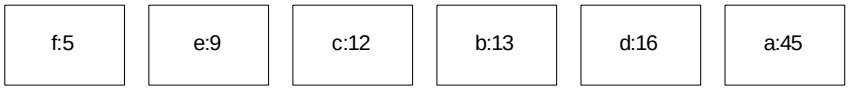


Şekil 25.6. *Sabit uzunluklu kodlama.*

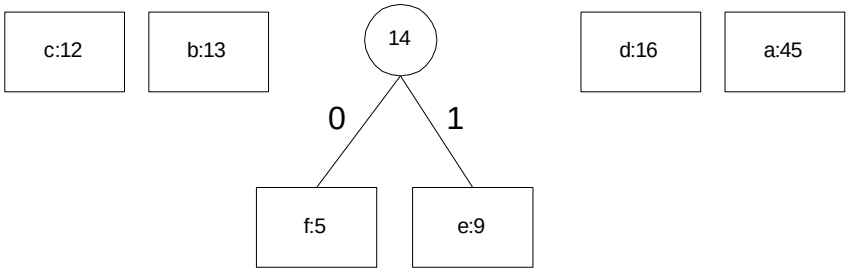
$$B(T) = \sum_{c \in C} f(c) d_T(c) \text{ olur.}$$

Huffman Kodu Üretme

Frekansa göre sıralanır ve her biri bir yaprağı temsil eder. En küçük iki değer bir atada toplanır ve geriye kalan yapraklar ile ata tekrar sıralanır.



Şekil 25.7. *Değişken uzunluklu kodlama için frekansların sıralanması.*

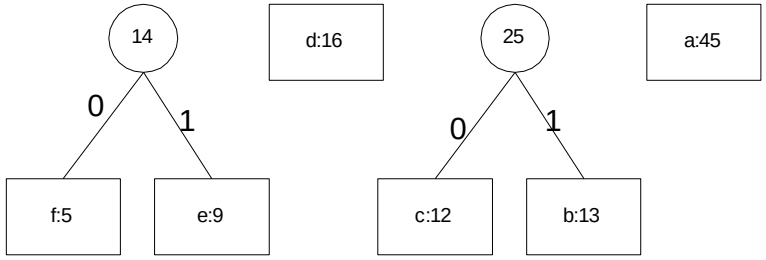


Şekil 25.8. *Değişken uzunluklu kodlama.*

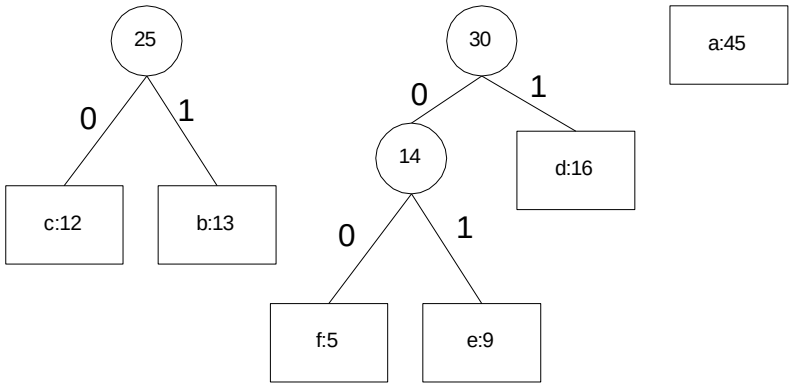
Algoritma 25.3. Huffman(C)

▹C alfabe olmak üzere bu algoritma Huffman kodu üretir.

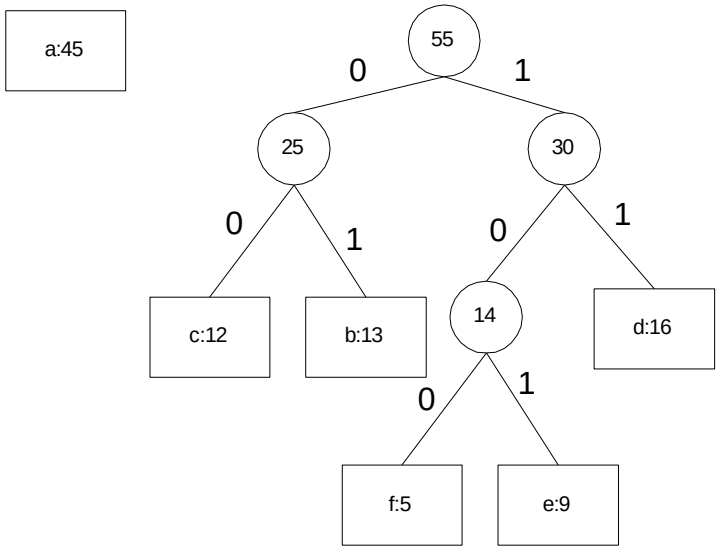
```
1. n←|C|
2. Q←C
3. i←1, …, n-1
4.     z←Düğüm_Oluştur()
5.     x←Sol(z)←Minimum_Al(Q)
6.     y←Sağ(z)←Minimum_Al(Q)
7.     f(z)←f(x) + f(y)
8.     Ekle(Q,Z)
9.  Sonuç←Minimum_Al(Q)
```



Şekil 25.9. *Değişken uzunluklu kodlama.*



Şekil 25.10. *Değişken uzunluklu kodlama.*



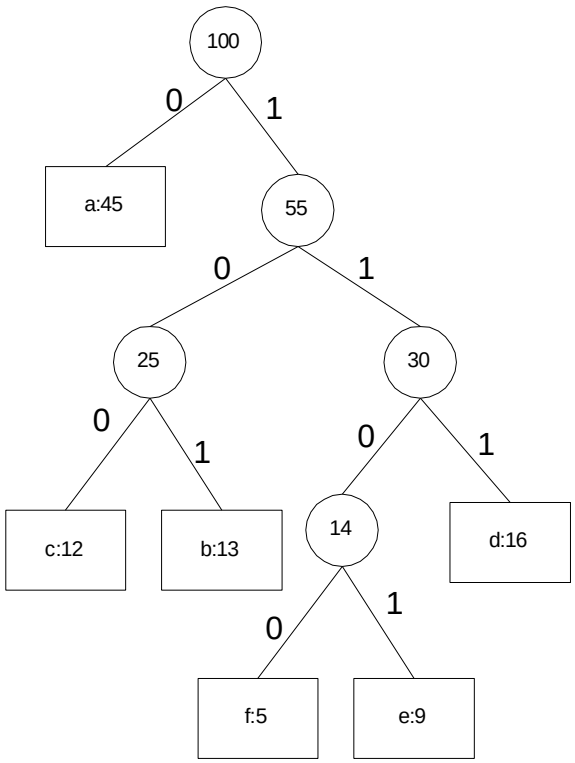
Şekil 25.11. *Değişken uzunluklu kodlama.*

Öneri 25.1.
C alfabe için f(c) frekansı göstersin ve x ve y en küçük iki frekansa sahip olsun. x ve y'nin kod uzunluğu aynı ve son bitleri farklı olan bir optimal kodlama vardır.

İspat
T gelişigüzel optimal önek kod ağacı olsun. Amaç bu ağacı değiştirerek en derindeki düğümün çocukları x ve y olacak şekilde yeni bir optimal ağaç elde etmektedir. b ve c karakterleri T ağacında en uzun koda sahip olsunlar.

f(b)≤ f(c) olması genelliği bozmaz ve benzer şekilde f(x)≤f(y) olsun. f(x) ve f(y) en küçük iki değer ve f(b) ile f(c) ise gelişigüzeldir. Bundan dolayı f(x)≤ f(b) olur ve f(y) ≤ f(c) olur.

T ve T' ağaçlarındaki kod miktarı farkı



Şekil 25.12. *Değişken uzunluklu kodlama.*

$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\
&= f(x) d_T(x) + f(b) d_T(b) - f(x) d_{T'}(x) - f(b) d_{T'}(b) \\
&= f(x) d_T(x) + f(b) d_T(b) - f(x) d_T(b) - f(b) d_T(x) \\
&= \big(f(b) - f(x)\big)\big(d_T(b) - d_T(x)\big) \\
&\geq 0
\end{aligned}
$$

Çünkü $f(b) - f(x) \geq 0$ ve $d_T(b) - d_T(x) \geq 0$. Benzer şekilde y ve c yer değiştirirse maliyet artmaz ve

$B(T') - B(T'') \geq 0$

olur.
Böylece $B(T'') \leq B(T)$, T optimal olduğundan $B(T'') = B(T)$ olur. T'' bir optimal ağaçtır ♦

Öneri 25.2.
T, C alfabesi üzerinde tanımlanmış tam dolu bir ikili ağaç olsun ve $\forall c \in C$ için f(c) tanımlı olsun. T ağacı içerisinde atası z olan iki tane x ve y gibi yaprak olsun. T'=T-{x,y} ağacında z bir karakter olarak düşünülüp frekansı f(z) = f(x)+f(y) şeklinde tanımlanmış olsun ve C'=C–{x,y}∪{z} alfabesinin optimal kod ağacı T' ağacıdır.

İspat
İlk olarak B(T)' nin B(T') cinsinden ifade edilebileceği gösterilsin.

$\forall c \in C-\{x,y\}$ için

$$d_T(c) = d_{T'}(c) \text{ ve } f(c)d_T(c) = f(c)d_{T'}(c)$$

olur.

$$d_T(x) = d_T(y) = d_{T'}(z) + 1$$

olduğundan

$$\begin{aligned} f(x)d_T(x) + f(y)d_T(y) &= \big(f(x)+f(y)\big)\big(d_{T'}(z)+1\big) \\ &= f(z)d_{T'}(z) + \big(f(x)+f(y)\big) \end{aligned}$$

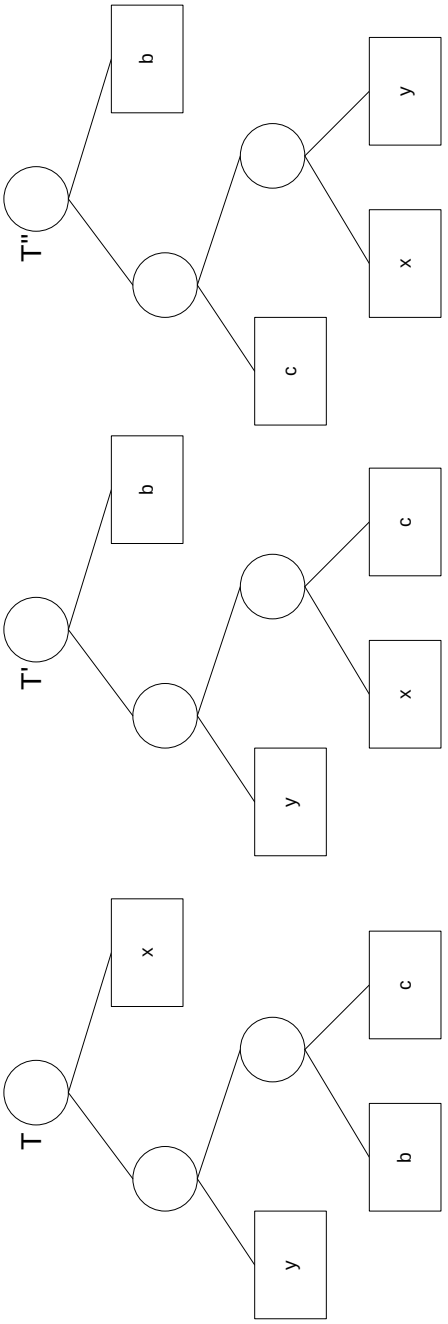
Buradan

B(T)=B(T')+f(x)+f(y)

olur. Eğer T' ağacı C' alfabesi için optimal değilse, T'' gibi bir ağaç vardır ve B(T'')<B(T') olur. z∈C' olduğundan, bu karakter T'' ağacının yaprağı olur ve

B(T'')+f(x)+f(y)<B(T)

olur ve bu T ağacının optimal olması ile çelişir ♦



Şekil 25.13. *En küçük frekanslara sahip olan iki karakter en derinde olur.*

25.6. Kaba Seçim Yönteminin Teorik Temelleri

Matroidler

Bir matroid M = (S, I) şeklindeki bir sıralı ikilidir ve aşağıdaki şartları sağlar.

- *S kümesi sonlu ve boş değil*
- *I boş olmayan S kümesinin alt kümelerinin bir ailesi olup, S kümesinin bağımsız alt kümeleri, eğer $B \in I$ ve $A \subseteq B$ ise AI. Eğer I bu özelliği sağlıyorsa , I'ya kalıtsal denir. Doğal olarak $\varnothing$ kümesi I'nın bir elemanıdır.*
- *Eğer $A \in I$, $B \in I$ ve $|A|<|B|$ ise, $\exists x$, $x \in B–A$ öyle ki $A \cup \{x\} \in I$ olur. Buna M değiş tokuş özelliğini sağlıyor denir.*

Diğer bir gösterimde yönsüz çizgeler için:

G= (V,E) çizgesi için grafik matroid $M_G = (S_G, I_G)$ şeklinde verilir.

- *$S_G = E$*
- *Eğer $A \subseteq E$ ise, $A \in I_G \Leftrightarrow A$ devresizdir. Bir ayrıtlar kümesi bağımsızdır ancak ve ancak bu ayrıtlar kümesi bir ormandır. Grafik matroidler minimum açılım ağacı ile ilişkilidirler.*

Teorem 25.1.
Eğer G yönsüz bir çizge ise, $M_G = (S_G, I_G)$ bir matroid 'dir.

İspat
S_G=E sonlu bir kümedir. I_G kalıtsal özelliğini korur, çünkü ormanın alt kümesi yine orman olur. Diğer bir deyişle devresiz bir kümede ayrıt silinmesi devre üretmez.

Geriye değiş-tokuş özelliğinin sağlanıp sağlanmadığının gösterilmesi gerekir. A ve B, G çizgesinin ormanları olsunlar ve $|B|>|A|$. k tane ayrıt içeren bir ormanda $|V|$-k tane ağaç vardır. Bunu ispatlamak için $|V|$ ağaç ile başlanır ve hiç ayrıt yoktur. Her ayrıt eklemede bir ağaç eksilir. Böylece A ormanı $|V|-|A|$ tane ağaç içerir ve B ormanı da $|V|-|B|$ tane ağaç içerir.

B ormanı daha az ağaç içerdiğinden B içerisindeki bir ağacın düğümleri A içerisindeki en az iki ağaç tarafından paylaşılırlar ve bu ağaç T olsun . (u,v) ayrıtı içerisinde ve u ve v düğümleri A içerisinde farklı ağaçlarda olan en az böyle bir ayrıt vardır. (u,v) ayrıtının A ormanına eklenmesi devre oluşturmaz ♦

M=(S,I) verilsin ve eğer x elemanı A'ya bağımsızlığı bozmadan eklenebiliyorsa, $x \in A$ ve $A \in I$, x elemanına genişleme denir. Diğer bir deyişle $A \cup \{x\} \in I$ olur. Eğer A, M metroidin bağımsız alt kümesi ise ve genişleme yoksa , A'ya maksimum denir.

Teorem 25.2.
Bir matroidin maksimum bağımsız alt kümelerinin eleman sayıları aynıdır.

İspat
A, M' nin maksimum alt kümesi ve daha geniş olan B alt kümesi olsun. Değiş-tokuş özelliğinden $x \in B-A$ 'dan A'nın genişletilebilir olduğu görülmektedir. Bu bir çelişkidir ♦

Buna örnek M_G 'nin maksimal alt kümesi açılım ağacıdır. Ağırlıklı matroid, her elemanına bir değer eşleştirilmiştir.

$A \subseteq S$ için $w(A) = \sum_{x \in A} w(x)$

olur.

25.7. Ağırlıklı Matroidler Üzerine Kaba Seçim Algoritması

Birçok problem, ağırlıklı matroidin maksimum ağırlıklı bağımsız alt kümesi olarak ifade edilebilir. M=(S,I) ağırlıklı matroid olsun. w(A) maksimum olacak şekilde $A \in I$ bulunmak isteniyor. Bu kümeye matroidin optimal alt kümesi denir. $\forall x \in S$ için $w(x)>0$ olduğundan w(A) maksimum olması için A' nın mümkün olan bütün elemanlara sahip olması gerekir.

G= (V,E) çizgesinin minimum açılım ağacı, böyle bir problem olarak ele alınabilir. $\forall e \in E$ için $w(e)>0$ ve $w_0>w(e)$ olsun . Bu durumda minimum açılım ağacını bulmak için herhangi bir $e \in E$ ayrıtı için $w'(e)=w_0-w(e)$ tanımlanır ve bunun toplamı maksimum yapılırsa , kullanılan ayrıtlar minimum ağırlıklı açılım ağacını teşkil ederler.

A kümesi için $w'(A)=(|V|-1)w_0-w(A)$

olur.

Algoritma 25.4. Kaba(M,V)

▷ M=(S,I) matroidinde maksimum ağırlık bağımsız alt kümesi elde edilecektir.

1. $A \leftarrow \varnothing$
2. S kümesi, w' ya göre büyükten küçüğe sıralanır.
3. $\forall x \in S$ için w(x) artmayan sırada alınır ve aşağıdaki kodu tekrar et
4. eğer $A \cup \{x\} \in I$ ise
5. $A \leftarrow A \cup \{x\}$
6. Sonuç←A

Eğer |S|=n ise, algoritmanın sıralama kısmı Θ(nlgn) olur. 4.satır S'nin her elemanı için çalışır ve her seferinde $A \cup \{x\}$ bağımsız olup olmadığı kontrol edilir. Eğer her kontrol Θ(f(n)) zamanını alıyorsa, toplam zaman Θ(nlgn+nf(n)) olur.

Öneri 25.3.
M=(S,I) ağırlıklı bir metroid ve w ağırlık fonksiyonu olsun ve S sıralı (büyükten küçüğe). x, S' nin ilk bağımsız elemanı olsun (eğer böyle bir eleman varsa). Eğer x varsa , S' nin A gibi bir optimal alt kümesi vardır ve x'i içerir.

İspat
(kaba seçim özelliği)

Eğer böyle bir eleman yoksa, tek bağımsız eleman ∅ olur. Aksi halde B boş olmayan bir optimal alt küme olsun. $x \notin B$' nin elemanı olmadığı kabul edilsin, aksi halde A=B olur. B' de ağırlığı w(x)' ten büyük eleman yoktur. $y \in B$ ise {y} bağımsızdır, çünkü $B \in I$ ve kalıtsal özelliğine sahiptir. Buradan $w(x) \geq w(y)$, $\exists y \in B$ olur. A şu şekilde inşa edilir.

A={x} ile başla.

Değiş-tokuş özelliğinden A'ya eklenebilecek eleman bulunur ve A'ya eklenir. Bu işlem |A|=|B| oluncaya kadar devam eder. Sonra bazı $y \in B$ için $A=B-\{y\} \cup \{x\}$, böylece

$$w(A)=w(B)-w(y)+ w(x)$$
$$\geq w(B)$$

B optimal olduğundan , A'da optimal olur ♦

Öneri 25.4.
M=(S,I) bir matroid olsun. Eğer $x \in S$ ve x elemanı, ∅ kümesinin genişlemesi değilse, x elemanı S kümesinin herhangi bir alt kümesi A'nında genişlemesi olmaz.

İspat
İspat, olmayan ergi yöntemine göre yapılsın. x elemanı A altkümesinin genişlemesi olsun ve ∅ kümenin genişlemesi olmasın. A altkümesinin genişlemesi x olduğuna göre $A \cup \{x\}$ bağımsız olur. I kalıtsal olduğundan, {x} bağımsız olmalıdır. Bu bir çelişki olur ♦

Öneri 25.5.
M=(S,I) ağırlıklı matroidi için KABA seçim algoritmasının seçmiş olduğu ilk eleman x olsun. Amaç geriye kalan M'=(S',I') olan matroid içerisinde maksimum ağırlıklı bağımsız alt kümeyi elde etmektir ve

$$S'=\{y \in S:(x,y) \in I\}$$
$$I'=\{B \subseteq S-\{x\}:B \cup \{x\} \in I\}$$

ve M' için verilen ağırlık fonksiyonu , M matroidi içinde geçerlidir ve M' için sadece S' ile sınırlanmıştır.

İspat
Eğer A kümesi M matroidin maksimum ağırlıklı alt kümesi ise, A'=A-{x} (A x'i içeriyor) kümesi ise M' matroidin bağımsız alt kümesi olur. Tersine, M' matroidin bağımsız alt kümesinden $A=A' \cup \{x\}$ elde edilir ve bu da M matroidin bağımsız alt kümedir. Her iki durumda da $w(A)=w(A')+ w(x)$ olduğundan x elemanını içeren M içerisindeki maksimum ağırlıklı çözüm, M' içindeki maksimum ağırlıklı çözümü üretir ♦

Teorem 25.3.
w ağırlık fonksiyonu ve M=(S,I) ağırlıklı matroid ise, KABA(M,w) optimal çözüm verir.

25.8. Görev Planlama Problemi

Burada amaç birim zamanlı görevleri tek işlemci üzerinde planlayarak ve zamanını geçiren (zamanı geldiğinde başlamayan görevler) görevlere ceza vermektir. S={1,2,3,….,n } birim zamanlı görevler kümesi olsun. İşlemci üzerinde görev planlaması, S kümesinin bir permutasyonunu elde etmektir. Bütün görevler birim zamanlı olduklarından

İlk görev 0→1 biter
İkinci görev 1→2 biter
Üçüncü görev 2→3 biter
.....
n. görev n-1→n biter

Görevlerin başlama zamanları $d_1, d_2, \ldots,d_n$ şeklinde verilsin ve bu zamanların özelliği $1 \leq d_1 \leq n$ olmalarıdır. $w_1, w_2,\ldots, w_n$ ağırlıkları göstersin ve $\forall i$, $w_i \geq 0$ olur. Zamanında bitmeyen göreve w_i cezası uygulanır. Kendi zamanından sonra biten görevlere "geç" görevler denir. Diğer durumda göreve "erken" görev denir. En basit algoritma "Erken-ilk" yapısıdır. Bir planlama kanonik yapıya sahip olmalıdır. Eğer aşağıdaki şartları sağlayan plana

- Erken görevler geç görevlerden her zaman önde yer alırlar.
- Erken görevler azalmayan tarihlerine göre planlanırlar.

kanonik plan denir veya bu plan kanonik olma özelliğini sağlıyor. Bunun için ilk olarak görevler erken-ilk yapısına dönüştürülür. Eğer i ve j görevleri i,j şeklinde yer alıyorlar ve $d_j<d_i$ ise, i ve j yer değiştirirler. Amaç A gibi bir plan bulmak ve bu plan optimal olmalıdır. Bir kere A elde edildikten sonra kanonik formu elde etmek daha kolaydır.

Eğer bir planda "geç" olan hiç görev yoksa, bu plana bağımsız plan denir. Açık olarak erken görevlerin oluşturduğu küme bağımsızdır ve I bağımsız olan bütün kümeleri içersin. t = 1,2,..,n için $N_t(A)$: t zamanında veya daha önce biten görevleri gösterir.

Öneri 25.6.

A görevler kümesi için aşağıdakiler denktir

1. *A bağımsız bir kümedir.*
2. *$t=1,2,\ldots,n$ için $N_t(A)\leq t$*
3. *A ' daki görevlerin zamanları azalmayan sırada ise "geç" görev almaz.*

İspat

Eğer $N_t(A)>t$ ise "geç" görev olmayan plan yapılamaz. Böylece $(1)\Rightarrow(2)$ olur. Eğer (2) varsa, (3)' te sağlanır ♦

Teorem 25.4.

S kümesi, birim zamanlı bir görev kümesi ve I bütün bağımsız görevlerin kümesi ise, (S,I) bir matroiddir.

İspat

Bağımsız kümenin her alt kümesi de bağımsız olur. Değiş-tokuş özelliğini ispatlamak için A ve B bağımsız kümeler olsunlar ve $|B|>|A|$ olsun. k en büyük t olsun ve $N_t(B)\leq N_t(A)$ olmak üzere $N_n(B)=B$ ve $N_n(A)=A$, fakat $|B|>|A|$ şeklinde kabul edilmişti. Buradan $k<n$ ve $N_j(B)>N_j(A)$, $\forall j$ için $k+1\leq j\leq n$ olur. Böylece B kümesinde tarihi k+1 olan görev sayısı A kümesinde tarihi k+1 olan görev sayısından daha fazla olacaktır. $x\in B-A$ ve bu elemanın tarihi k+1 olsun ve $A'=A\cup\{x\}$

olsun. Bu durumda A' kümesinin bağımsız olduğunun gösterilmesi gerekir. $1 \le t \le k$ için $N_t(A')=N_t(A) \le t$ olur, çünkü A bağımsızdır. $k<t \le n$ için $N_t(A') \le N_t(A) \le t$ olur, çünkü B bağımsızdır ♦

Örnek

S: Sonlu bir küme ve P: S kümesinin alt kümelerinin koleksiyonu olmak üzere eğer $A \in P$ ve $B \subset A$ ise, $B \in P$ olur. $w: S \rightarrow R^+$ ağırlık fonksiyonu ve $w: Z^S \rightarrow R^+$ ve

$$w(A) = \sum_{e \in A} w(e) \text{ ve } A \subseteq S \text{ olur.}$$

Amaç $A_0 \subseteq S$ şeklinde bir küme bulmak ve

- $A_0 \in P$
- P' nin bütün elemanları içinde $w(A_0)$ maksimum olacaktır.

Bu problem problem (P,w) şeklinde olsun.

1.a) $\{x_1\} \in P$ ve $w(x_1) \ge x_i$ ve $\forall x_i$, $\{x_i\} \in P$ ise, x_1 seçilir.
1.b) Eğer böyle bir x_1 yoksa bitir.
2.a) $\{x_2\} \in P$ ve $w(x_2) \ge x_i$, $\forall x_i$, $x_i \ne x_1$ ve $\{x_i\} \in P$ ($\{x_1,x_2\} \in P$) ise, x_2 seçilir.
2.b) Eğer öyle bir x_2 yoksa bitir.
.
.
.
k. a) $\{x_k\} \in P$ ve $w(x_k) \ge x_i$, $\forall x_i$, $x_i \ne x_1, x_2, \ldots, x_{k-1}$ ve $\{x_i\} \in P$, $\{x_1, x_2, \ldots, x_{k-1}, x_k\} \in P$ ise, x_k seçilir.
k. b) Eğer böyle bir x_k yoksa, bitir.

25.9.Bir Lineer Programlar Sınıfı için Kaba Seçim Algoritması

S={1,2,…,n} ve U kümesi, R^S kümesinin alt kümesi olsun ve $c=\{c_1,c_2,\ldots,c_n\}$ noktası $R^S=R^n$ uzayının bir noktası olsun. Amaç U üzerinde cx çarpımını maksimum yapacak bir c noktası bulmaktır.

Herhangi bir U ve c için Kaba seçim algoritması, $G(c,U)=(g_1,g_2,\ldots,g_n)\in U$ noktasını aşağıdaki yönteme göre bulur.

$c\in R^n$ ve $\Pi(c)=\{i_1,i_2,\ldots,i_n\}$ noktası $\{1,2,\ldots,n\}$ kümesinin tek permutasyonu olur ve bu permutasyon aşağıdaki özellikleri sağlar.

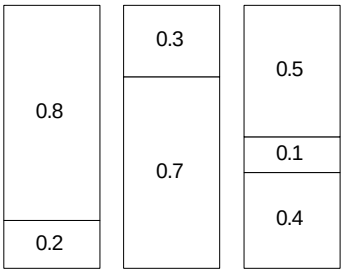
$$|c_{i_1}|\geq|c_{i_2}|\geq|c_{i_3}|\geq\cdots\geq|c_{i_n}|$$

$$|c_{i_j}|=|c_{i_k}| \quad \text{ve} \quad j<k \quad \text{ise,} \quad i_j<i_k \quad \text{olur.}$$

25.10. Yaklaşık Kutu Doldurma

Bu bölümde, bu problemi çözecek birkaç algoritma verilecektir ve bu algoritmaların hepsi yaklaşık çözüm bulacaktır. Elde edilen çözümlerin optimal yakın oldukları ispatlanacaktır.

N tane boyutları s_1, s_2, s_3, …s_n olan nesne verilsin. Bütün boyutlar $0\leq s_i\leq 1$ şartını sağlar. Her kutu birim kapasiteye sahip ve amacımız bu nesneleri minimum sayıda kutuya yerleştirmektir. Boyutları 0.2 , 0.5 , 0.4 , 0.7 , 0.1 , 0.3 , 0.8 olan nesneleri Şekil 25.14' te görüldüğü gibi kutulara doldurulurlar ve yapılan bu kutu doldurma optimaldir, çünkü boyutlar toplamı 3'tür. Bu problemin iki şekli vardır. Birinci şeklinde bir sonraki nesne üretilmeden o anki nesne kutuya yerleştirilir. Bu şekline çevrim-içi (on-line) denir. İkinci şekline ise çevrim-dışı (off-line) denir ve bu problemde bütün nesneler alınmadan herhangi bir işlem yapılmaz.



Şekil 25.14. *0.2 , 0.5 , 0.4 , 0.7 , 0.1 , 0.3 , 0.8 nesnelerin kutulara yerleşimleri.*

Çevrim-İçi

Sınırsız hesaplama imkanı olmasına rağmen çevrim-içi algoritmasının optimal çözüm verip vermediğinin düşünülmesi ilk iştir. Şu unutulmamalıdır ki bir sonraki nesne üretilmeden o anki nesne yerine yerleştirilir ve daha sonra bu karar değiştirilmez.

Çevrim-içi algoritma, her zaman doğru sonuç vermez. M tane küçük nesne dizisi (I_1) verilsin ve bu nesnelerin ağırlıkları $\frac{1}{2}-\varepsilon$ olsun. Bundan sonra ağırlığı $\frac{1}{2}+\varepsilon$ olan M tane büyük nesne girişten gelsin. ($0<\varepsilon<0.01$). Dikkat edilince, eğer bir kutuya bir büyük ve bir küçük nesne yerleştirilirse, bu nesneler M tane kutuya yerleştirilir. Bu işlemi yapan bir A algoritmasının var olduğu kabul edilsin. A algoritmasının M tane küçük nesne içeren I_1 dizisi üzerinde işlemi düşünülsün ve bu nesneler $\left\lceil \frac{M}{2} \right\rceil$ tane kutuya doldurulurlar. I_2 dizisi $\left\lceil \frac{M}{2} \right\rceil$ kutuya doldurulmalıdır. Bununla birlikte A her nesneyi farklı bir kutuya yerleştirir. Çünkü I_1 dizisini ilk yarısı için yapıldığı gibi I_2 dizisi üzerinde A algoritması aynı sonucu verir. Bunun anlamı, I_2 için lazım olan optimal kutu sayısının iki katı kadar kutu A algoritması tarafından kullanılır. Çevrim-içi kutu doldurma probleminin optimal algoritması yoktur.

Teorem 25.5.

Çevrim-içi-kutu-doldurma algoritmasının en az 4/3 optimal kutu sayısı kullanmaya zorlayan girişler vardır.

İspat

M çift olsun ve A algoritması I_1 dizisi üzerinde çalışıyor olsun.

$$I_1 = \underbrace{\text{küçük nesne}}_{M \text{ tane}} \cup \underbrace{\text{büyük nesne}}_{M \text{ tane}}$$

A algoritmasının M. nesneyi işleme aldıktan sonra ne yapar? A algoritması önceden b tane kutu kullanıyor olsun. Bu noktada optimal kutu sayısı M/2 olur, çünkü her kutuya iki tane nesne yerleştirilir. 4/3 performans garantisinden

$$\frac{2b}{M} < \frac{4}{3}$$

olur. Bütün nesneler yerleştirildikten sonra A algoritmasının performans düşünülsün. b. kutu bir tane nesne içerdikten sonra bütün kutular üretilir, çünkü küçük nesneler ilk b tane kutuya yerleştirilir ve iki büyük nesne bir kutuya sığmayabilir. Çünkü ilk b kutuda en fazla iki tane nesne vardır ve geriye kalanlar birer nesneye sahiptirler; 2M tane nesneyi paketlemek için 2M-b tane kutu gerekir. 2M tane nesneyi optimal olarak paketlemek için M tane kutu kullanılır bu durumda

$$\frac{2M-b}{M} < \frac{4}{3}$$

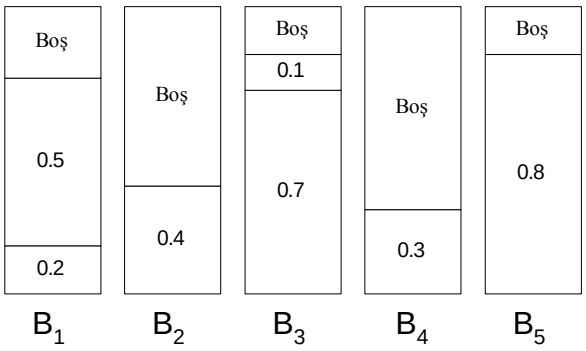
olur ve

$$\frac{2b}{M} < \frac{4}{3} \text{ ve } \frac{2M-b}{M} < \frac{4}{3} \Rightarrow 2 < \frac{4}{3} + \frac{b}{M}$$
$$\Rightarrow 4 < \frac{8}{3} + \frac{2b}{M} \Rightarrow 4 < \frac{8}{3} + \frac{4}{3} \Rightarrow 4 < 4$$

İlk eşitsizlikten $\frac{b}{M} < \frac{2}{3}$ elde edilir ve ikinci eşitsizlikten $\frac{b}{M} > \frac{2}{3}$ elde edilir ve bu bir çelişkidir. Böylece 4/3' ten daha iyi bir kutu doldurma yapacak algoritma yoktur. En fazla optimal sayının iki katını garanti eden üç tane algoritma vardır.

Son-Uyan

En basit algoritmadır. Bir nesne işlenirken, bu nesnenin en son kutuya konulan nesne ile aynı kutuya sığıp sığmayacağı kontrol edilir. Eğer sığıyorsa, oraya yerleştirilir; aksi halde yeni bir kutu oluşturulur (bu algoritmanın en kötü durumu da analiz edilebilir). 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8 nesnelerin kutulara doldurulması Şekil 25.15' te görüldüğü gibi olur. B_1 kutusuna 0.4 nesnesi sığmadığından B_2 kutusu oluşturulur. Bu algoritma bu şekilde devam eder.



Şekil 25.15. *0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8 nesnelerin kutulara yerleştirilmeleri.*

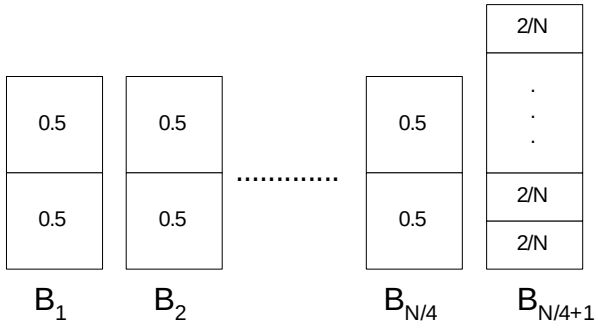
Teorem 25.6.
I nesneler listesini kutulamak için gerekli olan optimal kutu sayısı M olsun. Son-Uyan algoritması 2M kutudan fazla kutu hiçbir zaman kullanmaz. Son-Uyan algoritmasının 2M-2 kutu kullandığı diziler vardır.

İspat
B_j ve B_{j+1} kutuları bitişik olsunlar. B_j ve B_{j+1} kutularındaki nesnelerin boyutlarının toplamı 1'den büyüktür; çünkü diğer durumda bu nesneler B_j kutusuna sığardı. Eğer bu sonuç bütün bitişik kutulara uygulanırsa, en fazla uzayın yarısı boşa harcanmıştır. Böylece Son-Uyan algoritması, optimal kutu sayısının iki katı (en fazla) kadar kutu kullanılır.

Bu sınır sıkı bir sınırdır; N tane nesnenin boyutu s_i=0.5 olduğu kabul edilsin. Eğer i tek ise, $s_i = \frac{2}{N}$; eğer i çift ise, N sayısının 4 bölünebildiği kabul edilsin. Her birinde 2 tane nesne olan N/4 tane kutu olsun ve boyutu $\frac{2}{N}$ olan $\frac{N}{2}$ tane nesnenin içinde olduğu 1 kutu vardır ve toplam

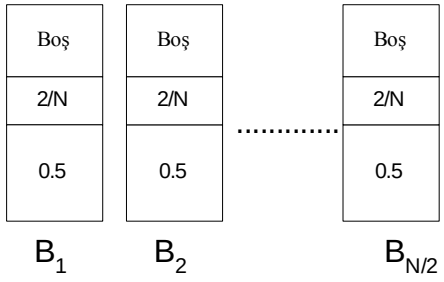
$$\left\lfloor \frac{N}{4} \right\rfloor + 1$$

olur. 0.5, $\frac{2}{N}$, 0.5, $\frac{2}{N}$, ... dizisinin optimal kutulanması Şekil 25.16' da olduğu gibi olur.



Şekil 25.16. *0.5, $\frac{2}{N}$, 0.5, $\frac{2}{N}$, ... nesneler dizisinin kutulanması.*

0.5, $\frac{2}{N}$, 0.5, $\frac{2}{N}$, ... nesneler dizisinin Son-Uyan algoritmasına göre kutulanması Şekil 25.17' de olduğu gibi olur.

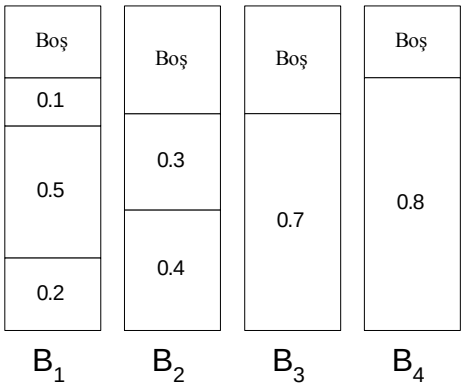


Şekil 25.17. *0.5, $\frac{2}{N}$, 0.5, $\frac{2}{N}$, ... nesneler dizisinin Son-Uyan algoritması ile kutulanması.*

İlk-Uyan

Bu yöntemde her gelen nesne için kutular baştan sona doğru kontrol edilir. Eğer alternatif (uygun kutu) kutu kalmazsa, yeni kutu

oluşturulur. 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8 nesneleri için İlk-Uyan algoritmasının sonucu Şekil 9' da görüldüğü gibi olur.

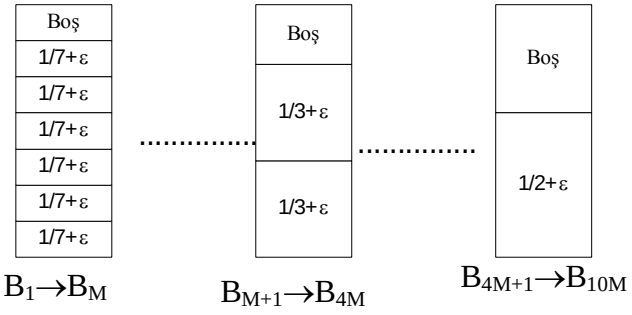


Şekil 25.18. *0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8 nesnelerinin İlk-Uyan algoritmasına göre kutulara doldurulması.*

Her nesne için kutular kontrol edilir ve bu durumda algoritmanın mertebesi $O(N^2)$ olur. Bu algoritma O(NlgN) mertebesinde uygulanabilir.

Her hangi bir anda boş alan kısmı yarıdan fazla olan kutu sayısı en fazla 1 olur, çünkü eğer ikinci yarıdan fazla boş ise, nesne birinci kutuya sığar. Böylece bu yöntemde en fazla kullanılacak kutu sayısı optimal kutu sayısın iki katı olur.

Teorem 25.7.
I nesneler listesini kutulamak için gerekli optimal kutu sayısı M olsun. İlk-Uyan algoritması $\left\lceil \frac{17M}{10} \right\rceil$ tane kutudan fazla kutu kullanmaz. İlk-Uyan algoritmasının $\frac{17}{10}(M-1)$ kutu kullandığı nesne listeleri vardır.

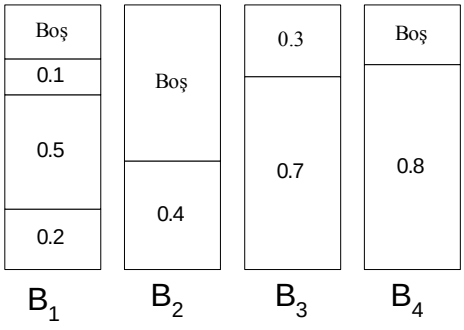


Şekil 25.19. $\frac{1}{7}+\varepsilon$, $\frac{1}{7}+\varepsilon$, ..., $\frac{1}{3}+\varepsilon$, $\frac{1}{3}+\varepsilon$, ..., $\frac{1}{2}+\varepsilon$, $\frac{1}{2}+\varepsilon$, ... *nesnelerinin İlk-Uyan algoritması ile kutulanmaları.*

İlk-Uyan algoritmasında 6M kutu yerine 10M kutu kullanılması durumudur. Girişte 6M tane boyutu 1/7+ ε olan nesne var, bundan sonra 6M tane boyutu 1/3+ ε nesne var ve son olarak 6M tane 1/2+ ε nesne vardır.

En-İyi-Uyan

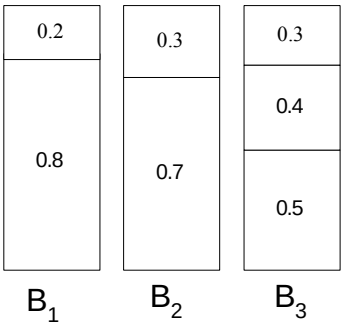
Bu yöntemde gelen nesne en iyi hangi kutuya sığıyorsa, o kutuya yerleştirilir. 0.2,0.5,0.4,0.7,0.1,0.3,0.8 dizisinin kutulara yerleşimi Şekil 25.20' de görüldüğü gibidir ve 0.3 nesnesi B_3 yerleştirilir.



Şekil 25.20. *0.2,0.5,0.4,0.7,0.1,0.3,0.8 nesnelerin En-İyi-Uyan algoritmasına göre kutulanması.*

Çevrim-İçi Algoritması

Bütün nesneleri inceleme imkanı varsa, daha iyi sonuç elde edilir. 0.8,0.7,0.5,0.4,0.3,0.2,0.1 nesnelerinin kutulara yerleştirilmesi Şekil 12' de olduğu gibi olur.



Şekil 25.21. *0.8,0.7,0.5,0.4,0.3,0.2,0.1 nesnelerin çevrim-içi algoritmasına göre kutulara yerleştirilmeleri.*

Bu yöntemde, İlk-Uyan-Azalan ve En-İyi-Uyan-Azalan algoritmaları kullanılır. Giriş boyutları azalana doğru sıralanır. Eğer optimalde M tane kutu kullanılıyorsa, İlk-Uyan-Azalan algoritması $\frac{4M+1}{3}$ tane kutu kullanır ve bu kutu sayısından fazla kutu kullanmaz.

Sonuç, iki tane gözleme bağlıdır. İlki, boyutu 1/3' ten büyük olan nesneler ilk M kutuya yerleştirilir. Bunun anlamı ekstra kutularda kalan nesneler en fazla 1/3 boyutuna sahip olurlar. İkinci gözlemde, ekstra kutularda olan nesnelerin sayısı en fazla M-1 olur. İki sonuç birleştirilince $\left\lceil \frac{M-1}{3} \right\rceil$ ekstra kutu gerekir.

Öneri 25.7.

N tane nesnenin boyutları $s_1,s_2....s_n$ (azalana doğru sıralı) olsun ve optimal kutu sayısı M olsun. İlk-Uyan-Azalan algoritmasına göre ekstra kutulara konulan nesnelerin boyutları en fazla 1/3 olur.

İspat

i. Nesne M+1 kutusuna konulan ilk nesne olsun.Amaç $s_i<=1/3$ olduğunun gösterilmesidir. $s_i>1/3$ olduğu kabul edilsin. Bu durumda s_1,s_2, ..., $s_{i-1}>1/3$ olur ve $B_1....B_M$ kutularında en fazla iki (her birinde) tane nesne vardır. (i-1). nesnesi yerleştirildikten sonra sistemin durumu düşünülebilir. Bazı kutularda bir ve bazılarında iki tane nesne vardır.

B_x: İki nesne (x_1,x_2), $1<=x<y<=M$
B_y: Bir nesne (y_1)

$x_1 \geq y_1$ olur, çünkü sıralı durumda x1 daha önce gelmektedir ve $x_2 \geq s_i$ olur.Bunun sonucunda $x_{1+}x_2 \geq y_1+s_i$ olur. Bunun sonucunda s_i nesnesinin B_y kutusuna yerleştirilebileceği görülmektedir.Yapılan kabule göre bu mümkün değildir. Böylece eğer $s_i>1/3$ ise, s_i işleme alındığında ilk M kutu düzenlenir öyle ilk j kutu bir nesneye sahip ve bundan sonraki M-j kutular ikişer nesneye sahiptir.

Açıkça s_1,s_2, ……, s_j nesnelerinden herhangi ikisi bir kutuya yerleşemezler. İlk-Uyan algoritmasına göre $s_{j+1},s_{j+2},….,s_i$ nesneleri ilk j kutularına yerleştirilemezler. Bunun sonucunda optimal kutulamada ilk j tane kutu bu nesneleri barındıramazlar. Boyutları $s_{j+1},s_{j+2},….,s_{i-1}$ nesneleri M-j kutularının bir kısmına yerleştirilirler ve bu şekildeki nesne sayısı 2(M-j) olur.

Eğer $s_i>1/3$ ise, s_i nesnesini M tane kutudan birisine yerleştirme yolu yoktur. Açıkça s_i nesnesinin j kutudan birine girmesi mümkün değildir. Geriye kalan M-j kutuya yerleştirmek için M-j kutuya 2(M-j)+1 tane nesnenin dağılımına ihtiyaç vardır. Böylece bazı kutularda boyutları 1/3'den büyük olan üç tane nesne olmalıdır ve bu mümkün değildir. Bu bir çelişkidir ve $s_i \leq 1/3$ olur ♦

Öneri 25.8.

Ekstra kutuya yerleştirilen nesne sayısı en fazla M-1 olur.

İspat

Ekstra kutuya yerleştirilen nesne sayısı en az M olsun. $\sum_{i=1}^{N} s_i \le M$ olduğu bilinmektedir. Çünkü bütün nesneler M tane kutuya sığıyor. $1 \le j \le M$ için B_j kutusu toplam ağırlığı W_j olan nesnelerle doldurulduğu kabul edilsin. İlk M ekstra nesnenin boyutları $x_1, x_2, \ldots\ldots, x_M$ olsun. İlk M kutudaki ve ekstra M kutudaki nesneler bütün nesnelerin alt kümesidir.

$$\sum_{i=1}^{N} s_i \ge \sum_{j=1}^{M} W_j + \sum_{j=1}^{M} x_j \ge \sum_{j=1}^{M} \left(W_j + x_j\right)$$

olur. Şimdi $W_j + x_j > 1$ olur. Aksi halde x_j nesnesinin B_j kutusuna yerleştirilmesi gerekir. Böylece

$$\sum_{i=1}^{N} s_i > \sum_{j=1}^{M} 1 > M$$

olur, fakat bu imkansızdır. Eğer N tane nesne M tane kutuya yerleştirilebiliyorsa, bunun sonucunda en fazla M-1 tane ekstra kutu olmalıdır ♦

Teorem 25.8.
I nesne listesi optimal olarak M kutuda paketlenebiliyor olsun. İlk-Uyan-Azalan algoritması en fazla (4M+1)/3 tane kutu kullanır.

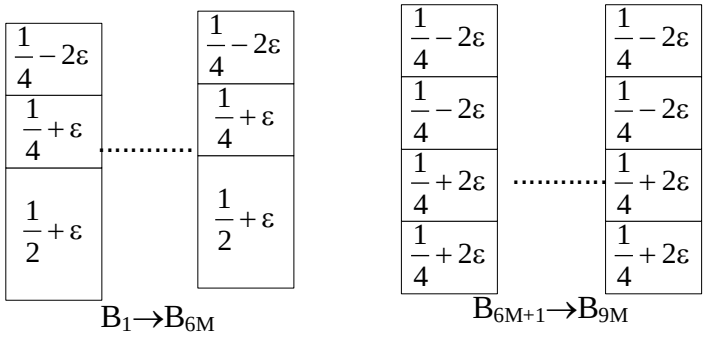
İspat
Boyutu en fazla 1/3 olan M-1 tane ekstra nesne vardır. Böylece en fazla $\left\lceil \frac{M-1}{3} \right\rceil$ tane ekstra kutu olmalıdır. Toplam kullanılan kutu sayısı

$$\left\lceil \frac{4M-1}{3} \right\rceil \le \frac{4M+1}{3}$$

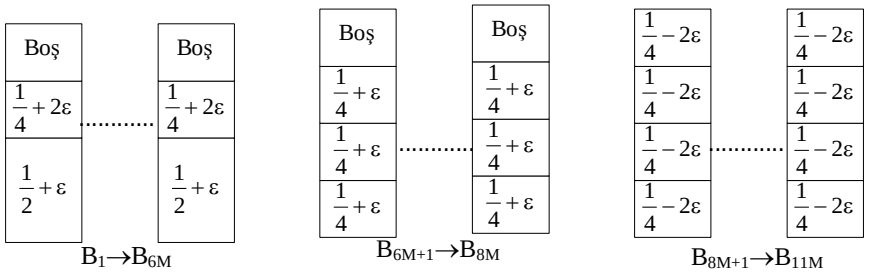
olur ♦

Teorem 25.9.
I nesne listesi optimal olarak M tane kutuda paketlenebiliyor olsun. İlk-Uyan-Azalan algoritması $\frac{11}{9}M+4$ tane kutudan fazla kutu kullanmaz. İlk-Uyan-Azalan algoritmasının $\frac{11}{9}M$ tane kutu kullanacağı girişler vardır.

İspat
Üst sınır belirleme kompleks işlemler gerektirir. Alt sınır belirlemek için 6M tane boyutu $\frac{1}{2}+\varepsilon$ olan nesne, ondan sonra boyutu $\frac{1}{4}+2\varepsilon$ olan 6M tane nesne, ondan sonra boyutu $\frac{1}{4}+\varepsilon$ olan 6M tane nesne ve son olarak 12M tane boyutu $\frac{1}{4}-2\varepsilon$ olan nesne olsun.



Şekil 25.22. *Nesnelerin optimal olarak kutulanması.*



Şekil 25.23. *İlk-Uyan-Azalan algoritmasına göre nesnelerin kutulanması.*