EGE ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

(YÜKSEK LİSANS TEZİ)

TAMSAYILI PROGRAMLAMA PROBLEMLERİ İÇİN GARANTİ DEĞERLİ ALGORİTMALAR

Aslı GÜLER

Matematik Anabilim Dalı
Bilim Dalı Kodu: 619.03.03

Sunuş Tarihi: 01.08.2008

Tez Danışmanı: Prof. Dr. Urfat G. NURİYEV

Bornova-İZMİR

Aslı GÜLER tarafından Yüksek Lisans tezi olarak sunulan **"Tamsayılı Programlama Problemleri İçin Garanti Değerli Algoritmalar"** başlıklı bu çalışma E.Ü Lisansüstü Eğitim ve Öğretim Yönetmeliği ile E.Ü Fen Bilimleri Enstitüsü Eğitim ve Öğretim Yönergesi'nin ilgili hükümleri uyarınca tarafımızdan değerlendirilerek savunmaya değer bulunmuş ve 01/08/2008 tarihinde yapılan tez savunma sınavında aday oybirliği / oyçokluğu ile başarılı bulunmuştur.

Jüri Üyeleri:	**İmza**
Jüri Başkanı : Prof. Dr. Urfat NURİYEV	**…….**
Raportör Üye: Yrd. Doç. Dr. Burak ORDİN	**……**
Üye : Doç. Dr. Refail KASIMBEYLİ	**…….**

ÖZET

TAMSAYILI PROGRAMLAMA PROBLEMLERİ İÇİN GARANTİ DEĞERLİ ALGORİTMALAR

GÜLER, Aslı

Yüksek Lisans Tezi, Matematik Bölümü

Tez Yöneticisi: Prof. Dr. Urfat G. Nuriyev

Ağustos 2008, 61 sayfa

Bilindiği gibi çoğu tamsayılı optimizasyon problemi NP-tam problemdir ve P=NP olmadığı sürece bu problemler için polinom zamanda kesin çözüm veren algoritmalar bulma olasılığı azdır. Bu nedenle, son yıllarda NP-tam problemler için garanti değerli algoritmalar tasarlamak ilgi çekici konulardan biri haline gelmiştir.

Bu çalışmada teknik ve ekonomik alanlarda birçok uygulamaları olan tek boyutlu ***Tamsayılı Sırt Çantası Problemleri*** incelenmiş olup bu problemler için greedy algoritmaları önerilmiştir. Algoritmaların ürettiği sonuçların optimal çözüme ne kadar yakın olduğunun bulunması amacıyla, önerilen algoritmaların garanti değerleri hesaplanmıştır. Ayrıca *Tamsayılı Maksimizasyon Sırt Çantası Problemi* için tümleyen problem tanımlanmış ve bu probleme dayanarak daha önce hesaplanan garanti değerin iyileştirilmesi amaçlanmıştır.

Anahtar Kelimeler: Tamsayılı programlama, Sırt çantası problemleri Greedy algoritmalar, Garanti değer, En kötü durum analizi, Tümleyen problem.

ABSTRACT

ALGORITHMS WITH GUARANTEE VALUE FOR INTEGER PROGRAMMING PROBLEMS

GÜLER, Aslı

MSc in Mathematics

Supervisor: Prof. Dr. Urfat G. Nuriyev

August 2008, 61 pages

It is known that many integer optimization problems are NP-complete and there is little probabilty to find algorithms for these problems that give exact solutions in polinomial time unless P=NP. As a result, designing algorithms with guarantee value for NP-complete problems has become one of the attractive subjects in recent years.

In this thesis, one-dimensional ***Integer Knapsack Problems***, which have many applications in technical and economic area, have been studied; then greedy algorithms have been proposed for these problems. Guarantee values of these algorithms have been calculated in order to determine how much close the results returned by the algorithms are to optimal solutions. Furthermore, complementary problem for *Integer Maximization Knapsack Problem* has been defined and in terms of this problem, it has been aimed to improve guarantee value calculated before.

Keywords: Integer programming, Knapsack problems, Greedy algorithms, Guarantee value, Worst-case analysis, Complementary problem.

TEŞEKKÜR

Çalışmalarım boyunca bana destek veren, sabrını esirgemeyen, değerli bilgisini, zamanını ve görüşlerini paylaşan sayın hocam Prof. Dr. Urfat G. NURİYEV'e, yüksek lisans öğrenimimde beni destekleyen Türkiye Bilimsel ve Teknolojik Araştırma Kurumu'na (TÜBİTAK) ve her zaman arkamda olan, bana cesaret veren aileme teşekkürlerimi sunarım.

İÇİNDEKİLER

Sayfa

İÇİNDEKİLER (devam)

İÇİNDEKİLER (devam)

ŞEKİLLER DİZİNİ

Şekil Sayfa

ÇİZELGELER DİZİNİ

Çizelge Sayfa

1. GİRİŞ

Tamsayılı doğrusal programlama, en geniş anlamıyla, normal olarak sürekli tanımlanmış değişkenlerin, kesikli değerler alan değişkenler biçiminde tanımlanmasıdır ve gerçek problemlerde en sık karşılaşılan durumlara çözüm aramaktadır. "Ayrık Optimizasyon" ya da "Kombinatorik Optimizasyon" problemlerinin bir alt sınıfı olarak da bilinen tamsayılı programlama problemlerinin mühendislik, iktisat ve askeri alanlar gibi bir çok alanda uygulamaları mevcuttur ve bu nedenle en yaygın çalışılan konulardan biri olmuştur.

Bilindiği gibi çoğu tamsayılı programlama problemi NP-tam problemdir. Bu problemler için P=NP olmadığı sürece polinom zamanda kesin çözüm veren algoritmaların bulunması ümit edilmemektedir. Dolayısıyla bu durum, çözüm için etkin ve hızlı çalışan yaklaşım algoritmaları gereksinimini ortaya çıkarmış ve son yıllarda garanti değerli algoritmaların tasarımını ilgi çekici konulardan biri haline getirmiştir.

Bu çalışmada genel olarak tamsayılı doğrusal programlama problemleri ve çözüm yöntemleri ele alınmıştır; özel olarak ise sırt çantası problemleri ile garanti değerli greedy algoritmalar anlatılmıştır.

Tamsayılı doğrusal programlama, doğrusal programlamanın uzantısı olduğu için ikinci bölümde öncelikle doğrusal programlamaya yer verilmiştir. Bu bölümde doğrusal programlama modelleri, çözüm yöntemleri ve kullanım alanları gibi konular incelenmiştir.

Üçüncü bölümde tamsayılı programlamaya giriş yapılmış ve genel tanımı ile tamsayılı programlama problemlerinin türleri üzerinde durulmuştur. Ayrıca çalışmamızla ilgili olduğundan, özel olarak sırt çantası problemleri anlatılmıştır.

Dördüncü bölümde tamsayılı problemlerin çözüm yöntemleri olan kesin ve yaklaşık çözüm yöntemleri üzerinde durulmuş ve bu sınıflara ait bazı yöntemler kısaca anlatılmıştır. Yine çalışmamızla ilgili olması dolayısıyla, greedy algoritmaları ve garanti değer konuları incelenmiştir.

Beşinci bölümde 0-1 KP' nin maksimizasyon versiyonu için önerilmiş olan bir greedy algoritması, bu problem için tanımlanan tümleyen problem ve önceki algoritmanın garanti değerini iyileştirmek için tümleyen probleme göre tasarlanmış başka bir greedy algoritması incelenmiştir.

Altıncı bölümde, öncelikle tamsayılı minimizasyon sırt çantası problemi için önerdiğimiz greedy algoritması ve bu algoritmanın garanti değer hesabına yer verilmiştir. Daha sonra maksimizasyon sırt çantası için bir algoritma önerilmiş ve garanti değeri hesaplanmıştır. Hesaplanan bu garanti değerin daha da iyileştirilebileceği düşünülerek maksimizasyon probleminin tümleyeni oluşturulmuştur. Elde edilen tümleyen problem için yeni bir greedy algoritması önerilmiştir ve bu algoritmanın da garanti değeri hesaplanarak bazı teoremler yardımıyla garanti değerdeki iyileşme incelenmiştir.

Son bölümde ise elde edilen sonuçlar ile algoritmaların karmaşıklıklarına yer verilmiş ve sonuçların geometrik yorumu gösterilmiştir.

2. DOĞRUSAL PROGRAMLAMA

Bu çalışmada kullanılacak olan tamsayılı programlama problemleri, doğrusal programlama problemlerinin özel halidir. Bu nedenle, bu bölümde öncelikle doğrusal programlamanın gelişimi, kullanım alanları ve doğrusal programlama problemlerinin modellenmesi ile çözüm yöntemleri incelenecektir.

2.1. Doğrusal Programlamanın Gelişimi ve Kullanım Alanları

Doğrusal programlama bir matematiksel model olarak ortaya çıkmış olup 2. Dünya Savaşı boyunca, maliyet ve gelir planlamasında, ordunun masrafını azaltmak ve düşmana verilen zararın artırımı gibi konularda kullanılarak gelişme sağlamıştır.

1947 yılında yayınlanan *simpleks algoritması* ile George Dantzig, aynı yıl içinde dualite teorisini geliştiren John von Neumann ve Dantzig'den önce benzer yöntemleri ekonomi alanında kullanan Rus Matematikçi Leonid Kantorovich doğrusal programlamanın kurucuları olarak gösterilirler.

Doğrusal programlama çeşitli nedenlerden dolayı optimizasyonun önemli bir alanıdır. Yöneylem araştırmasında birçok problem doğrusal programlama problemi olarak ifade edilelebilir. Benzer olarak, ekonomi

ve iş yönetiminde, eğitim, petrol işletmeleri, gıda sektörü, finans yönetimi gibi çeşitli alanlarda geliri en büyüklemek ya da maliyeti en küçüklemek vb. amaçlarla sıklıkla kullanılır.

2.2. Doğrusal Programlama Modeli ve Çözüm Yöntemleri

Bir sistemin bileşenlerinin simgelerle tanımlanıp, bunlar arasındaki ilişkilerin fonksiyonlarla gösterimine **matematiksel model**; sistemin yöneticisinin kontrolü altında olup **karar değişkeni** olarak isimlendirilen değişkenlere, hangi değerlerin verilmesi gerektiğini belirlemek amacıyla kullanılan matematiksel modellere de **karar modeli** denir. Karar vericinin kontrolü dışında değer alan bileşenlere parametre; modelde karar değişkenleri ya da karar değişkenleri ile parametreler arasındaki zorunlu ilişkilerin her birine ise **kısıt** adı verilir (Kara, 1991). Karar verici, karar değişkenlerinin fonksiyonunu en büyüklemek (gelir, kâr vb. için) ya da en küçüklemek (gider, maliyet vb.) ister. En büyüklenen ya da en küçüklenen bu fonksiyon **amaç fonksiyonu** adını alır (Winston, 1991).

Bir doğrusal programlama modeli, doğrusal bir amaç fonksiyonu ve tümü doğrusal olarak ifade edilmiş kısıtlardan oluşur. Örneğin m tane kısıt, n tane karar değişkeninden oluşan bir doğrusal programlama modeli aşağıdaki gibi ifade edilir:

$$\sum_{j=1}^{n} a_{ij} x_j (\leq , = , \geq) b_i \quad i = 1,...,m$$

$$x_j \geq 0, \qquad j = 1,...,n \qquad (2.2.1)$$

kısıtları altında (k.a)

$$enb\ Z^* = \sum_{j=1}^{n} p_j x_j$$

Burada;

x_j : j. karar değişkenine atanacak değer,

p_j : Bir birim j. karar değişkeninin amaç fonksiyonuna katkısı (gelir, kar, maliyet vb.),

b_i : i. kaynak miktarı,

a_{ij} : Bir birim x_j için gerekli i. kaynak veya girdi,

olarak tanımlıdır.

Bir doğrusal programlama modelinin çözüm yaklaşımları,

- Grafik çözüm,
- Analitik çözüm,

- Ardışık sayısal çözüm

başlıkları altında toplanabilir (Kara, 1991).

Bu yöntemlere kısaca değinmek gerekirse; grafik çözüm en fazla üç karar değişkeni olduğunda uygulanabilmektedir. Gerçek hayatta karşılaşılan problemlerin çok sayıda karar değişkeni içeriyor olması grafik çözümün ancak kavramların anlaşılabilirliği için kullanılmasına neden olmaktadır. Analitik çözüm ise en iyi çözümün varlığının biliniyor olması halinde kullanılabilir ki çoğunlukla bu durum mümkün değildir. Bunun yanı sıra değişken ve kısıt sayısı arttıkça oluşan denklem sisteminin boyutu ve dolayısıyla işlem yoğunluğu artar.

Bu iki yöntemin gerektirdiği ihtiyaçlar doğrultusunda yapılan çalışmalar ardışık çözüm yöntemlerini geliştirmiştir. Bunlardan ilki ve en yaygını "Simpleks Algoritması" olup "Karmarkar Algoritması" da en çok bilinenler arasında sayılabilir (Kara, 1991).

3. TAMSAYILI DOĞRUSAL PROGRAMLAMA

Bu bölümde tamsayılı programlama problemlerinin genel tanımı ve türleri verilmiş olup çalışmamızla ilgili özel olarak sırt çantası problemlerine yer verilmiştir**.**

3.1. Tamsayılı Programlamanın Genel Tanım ve Türleri

Bir problemde tüm değişkenlerin tamsayı olması isteniyorsa bu problem *saf tamsayılı programlama* (TP) problemi olarak adlandırılır.

Tamsayılı programlama tekniği, doğrusal programlamanın bir uzantısı olup doğrusal programlamada meydana gelebilecek gerçekçi olmayan sonuçları ortadan kaldırmayı amaçlar. Bazı doğrusal programlama modellerinde sonuçların tamsayı çıkmaması problemin gerçek hayattaki problemlere uygunluğunu bozmaktadır. Örneğin bir üretim probleminde sonuçların 1.2 cep telefonu, 3.6 kamera gibi kesirli çıkması gerçekçi olmamaktadır. Bu sonuçların tamsayıya yuvarlatılması akla gelebilecek ilk yöntem olabilir; ancak böyle bir yol takip etmek çözümü optimallikten oldukça uzaklaştırabilir. Tamsayılı programlama tekniği, kısıtları bozmadan sonucun tamsayı olmasını sağlamaktadır (Bakır ve Altunkaynak, 2003).

Tamsayı değerli değişkenler içeren modeller çoğunlukla büyük ölçekli planlama modelleridir. Örneğin üretim problemleri, gezgin satıcı

problemleri, sırt çantası, hat dengeleme vb. problemler tamsayılı programlamaya göre modellenebilir.

Tamsayılı programlama problemlerinin formülasyonu, sürekli değişkenli matematiksel modellerin formülasyonuyla benzerlik göstermesine rağmen, bazı kısıtlamalar hesaplama bakımından tamsayılı problemleri daha zor hale getirir. Çoğu tamsayılı problem NP-zor sınıfına aittir; dolayısıyla genel doğrusal modeller en kötü durumda etkin olarak çözülebilirken, tamsayılı programlama problemleri üstel hesaplama zamanı gerektirebilir.

Tamsayılı programlama problemlerinde uygun çözüm bölgesi ne sürekli ne de konvekstir. Problemin uygun çözüm noktaları, çözüm alanının uç noktalarına hatta bölgenin sınırları üzerine düşmeyebilir. Bu nedenle uç noktalarda çözüm arayan doğrusal programlama algoritmaları tamsayılı programlama problemleri için direkt olarak uygulanamaz.

Genel olarak tek boyutlu bir tamsayılı maksimizasyon probleminin matematiksel modeli aşağıdaki gibi ifade edilebilir:

$$\sum_{j \in J} a_j x_j \leq b$$

$x_j \geq 0$ ve tamsayı, $j \in J$

k.a

$$enb\ Z^* = \sum_{j \in J} p_j x_j$$

Eğer sadece bazı değişkenlerin tamsayı olması isteniyorsa bu problem *karma tamsayılı problem* olarak adlandırılır.

Örnek 3.1

$x_1 + 2x_2 \leq 8$
$x_1 \geq 0$ ve tamsayı
$x_2 \geq 0$
k.a

$enb\ Z^* = 4x_1 + x_2$

0-1 tamsayılı programlama tamsayılı programlamanın özel bir halidir, öyle ki değişkenler rasgele değerler yerine sadece 0 veya 1 değerini alabilir. Bu problem de NP-zor olarak sınıflandırılır ve bu problemin karar versiyonu Karp'ın 21 NP-tam probleminden biridir (Garey and Johnson, 1979).

Örnek 3.2

$15x_1 + 35x_2 + 20x_3 \leq 60$
$x_1, x_2, x_3 = 0$ veya 1
k.a

$enb\ Z^* = 32x_1 + 45x_2 + 25x_3$

3.2. Sırt Çantası Problemleri

Sırt çantası problemleri (Knapsack problems- KP) tamsayılı programlama problemleri içinde en basit olan türdür. Bunun nedeniyse problemin tanımının anlaşılabilir olması ve gerçek hayatta karşılaşılan bir çok durumun bu probleme bağlı olarak kolayca ifade edilebilmesidir.

Son yıllarda sırt çantası problemleri yoğun olarak çalışılmaktadır. Bunun nedeni teorisyenler açısından problemin basit yapısı olup uygulama alanındakiler içinse bir çok endüstriyel duruma uygulanabilmesidir. Örneğin sermaye bütçeleme, proje seçimi, kargo yükleme, kesme gibi problemler KP' nin en klasik uygulamalarıdır ve bu problem çerçevesinde çözülebilirler.

Problem sadece tek bir kısıt ve sadece pozitif katsayılarla en basit tamsayılı modeldir; ancak değişkenlerin tamsayı olma şartının eklenmesi sırt çantası problemini NP-tam sınıfına yerleştirir.

Sırt çantası problemi, bir maksimizasyon problemi için en basit prototip olması nedeniyle yüz yıllardır çalışılmaktadır. 1897 yılında G.B. Mathews çeşitli kısıtların nasıl tek bir sırt çantası kısıtında toplanabildiğini göstermiştir. Bu, genel bir tamsayılı problemin sırt çantası problemine indirgemesinin ilk örneklerinden biridir ve dolayısıyla sırt çantası problemini çözmenin en az bir tamsayılı problemi çözmek kadar zor olduğunu ispatlar. Salkin ve De Kluyver (1975) tamsayılı doğrusal programların KP' ye dönüştürülmesinde çok sayıda endüstriyel

uygulama ve sonuçlarını sunmuşlardır. Martello ve Toth (1979) 0-1 KP için kesin algoritmalar ve bunların ortalama performanslarını araştırmışlar sonrasında ise bu çalışmayı diğer KP türleri ve yaklaşık algoritmalar için genişletmişlerdir (1987).

Sırt Çantası Problemi Nedir?

• Bir tur için sırt çantasını hazırlayan ve hangi eşyaları yanına alması gerektiğini belirlemek zorunda olan bir dağcı düşünün. Tur için kullanışlı olabilecek bir sürü nesne ve kapasitesi sınırlı olan bir de çanta var. Bu nesnelerin her biri 1'den n'ye kadar numaralandırılmış ve her birinin sağladığı yarar $p_j > 0$ sayısıyla, ağırlıkları ise $a_j > 0$ sayısıyla ölçülüyor. Bu kişi çantasını maksimum faydayla ve çanta kapasitesi olan 'c' yi aşmayacak şekilde doldurmak istiyor. Hangi eşyaları almalı?

• Bir yatırımcının elinde "c" kadar belirli miktarda parası olsun ve bu parayı da kazanç sağlayabileceği işlere yatırmak istesin. Kararları için öncelikle olası yatırımların uzun bir listesini hazırlıyor ki burada a_j, o işe yatırması gereken para; p_j ise o işten beklenen net kazanç olarak belirtiliyor (Burada risk durumu hesaba katılmıyor.). Nereye yatırım yapmalı?

Yukarıda anlatılan her iki problem de KP çerçevesinde çözülen problemlerdir. Problemin formal tanımı ve TP formülasyonu ise aşağıdaki gibidir:

Yanımıza almak istediğimiz *n* adet nesne olsun ve bu nesneleri *j* ile indeksleyelim, $J = \{1, 2, ..., n\}$,

p_j : j. nesnenin sağladığı fayda,

a_j : j. nesnenin ağırlığı,

b : Sırt çantasının toplam kapasitesi

olsun. (Genellikle tüm bu değerler pozitif tamsayı olarak alınır.) Burada amaç, nesneler kümesinden, toplam faydanın maksimum olduğu ve toplam ağırlığın b kapasitesini geçmediği bir alt küme seçmektir.

kısıtlar:
$$\begin{cases} \sum_{j \in J} a_j x_j \le b & (3.2.1) \\ x_j \in \{0,1\}, \quad j \in J & (3.2.2) \end{cases}$$

amaç fonksiyonu: $enb\ Z^* = \sum_{j \in J} p_j x_j$

Karar değişkenlerinin tanımı bakımından sırt çantası problemini genel olarak üç türde sınıflandırabiliriz:

1. 0/1 Sırt Çantası Problemleri

2. Negatif olmayan tamsayılı sırt çantası problemleri (Kısaca tamsayılı diyeceğiz.)

3. Sınırlı tamsayılı sırt çantası problemleri

Yukarıdaki formülasyon birinci tür KP problemidir. İkinci türdeki sırt çantası problemleri içinse (3.2.2) kısıtı yerine

$$x_j \geq 0 \text{ ve tamsayı, } j \in J \qquad (3.2.2)'$$

kısıtı kullanılır. (Bu problem *sınırsız tamsayılı KP* olarak da bilinir.)

Verilen problemde çantaya yerleştirilecek nesnelerin sayıları sınırlandırılmış olabilir. Bu durumda problem sınırlı tamsayılı KP' dir ve bu modelde (2) kısıtı,

$$0 \leq x_j \leq c_j \text{ ve tamsayı, } j \in J \qquad (3.2.2)''$$

kısıtıyla yer değiştirir ve dolayısıyla problem çözümünde her hangi bir x_j nesnesi en fazla c_j değeriyle yer alır.

Problem içerdiği kısıt sayısı bakımından ise

1. Bir boyutlu sırt çantası problemleri

2. Çok boyutlu sırt çantası problemleri

olarak sınıflandırılır.

Eğer KP' de sadece bir tane ana kısıt varsa bu tür problemler "Bir boyutlu sırt çantası problemi" olarak adlandırılır; yukarıdaki formülasyon ve tanım bu türe örnektir.

Varsayalım ki her bir maddenin hem ağırlığı hem de çantada kaplayacağı hacim değeri verilmiş olsun. Dağcının çantası taşıyacağı ağırlık bakımından sınırlı olduğu gibi hacim bakımından da sınırlı olsun. Dolayısıyla problem şimdi toplam hacim ve toplam ağırlık bakımından iki kısıt altındadır. Bu türden bir modelin genel formu, m tane kısıt içerecek şekilde aşağıdaki gibi tanımlanabilir:

$$\sum_{j \in J} a_{ij} x_j \leq b_i \quad i = 1, ..., m$$

$$x_j \in \{0,1\}, \quad j \in J$$

k.a

$$enb\ Z^* = \sum_{j \in J} p_j x_j$$

Yukarıda bahsedilen KP çeşitlerinin yanı sıra çalışılan bazı diğer sırt çantası problemleri aşağıdaki gibidir:

- Minimizasyon sırt çantası problemi
- Katlı sırt çantası problemi
- Çok seçmeli sırt çantası problemi

- Kuadratik sırt çantası problemi

- Çok amaçlı sırt çantası problemi

Özel olarak, minimizasyon KP için sonlu sayıdaki nesneler kümesinin öyle bir alt kümesi seçilmelidir ki nesnelerin toplam maliyeti minimum ve toplam ağırlık en az d kadar olsun. Bu durumda yeni model 0-1 tamsayılı için aşağıdaki gibi olur:

$$\sum_{j\in J} a_j y_j \geq d$$

$$y_j \in \{0,1\},\ \ j \in J$$

k.a

$$enk\ Z^* = \sum_{j\in J} p_j y_j$$

Bir diğer problem sınırlı tamsayılı minimizasyon sırt çantası problemi esas KP' nin bir dönüşümüdür ve daha sonraki bölümlerde incelenecektir.

4. TP PROBLEMLERİNİ ÇÖZMEK İÇİN KULLANILAN YÖNTEMLER

Günümüzde, matematiksel modellerin özelliklerini göz önüne alarak ayrı ayrı tamsayılı problem sınıflarını çözmek için çok sayıda yöntem; ayrıca kesin ve yaklaşık çözümler üreten bazı genel algoritma şemaları bilinmektedir. Ancak göz önünde bulundurulması gereken bir nokta vardır; tecrübeler göstermiştir ki bu problemler hesaplama açısından en zor çözülen problemlerin başında gelir.

Tamsayılı problemlerin çözüm yöntemleri kesin ve yaklaşık çözüm yöntemleri olarak ikiye ayrılmaktadır. *Kesin yöntemler* optimal çözümü garanti ederler; ancak gerek hesaplama zamanı gerekse sarf edilen çaba (zaman ya da maliyet) nedeniyle sonuçlar her zaman tatmin edici değildir. Böyle durumlarda *yaklaşık yöntemler* kullanılabilir. Yaklaşık algoritmalar optimal çözüme ulaşmasalar da makul ölçüde yaklaşarak kısa sürede çözüm üretebilirler.

4.1. Kesin Yöntemler

Bu sınıftaki bazı yöntemler aşağıdaki gibi sayılabilir:

- Sayımlama (Enumeration),
- Kesen düzlemler (Cutting planes),

- Dinamik programlama (Dynamic Programming),
- Dal-sınır (Branch and bound),
- Dal-kesme (Branch and cut).

Sayımlama

Sayma metodu tüm olasılıkları saymaya dayanan bir yöntemdir. Bulunan tüm olasılıklar için bir amaç fonksiyonu değeri hesaplanır ve sonunda bunlar arasından en iyi olan çözüm seçilir. Bu yöntem az değişkene sahip problemler için kullanışlı olmasına karşın, değişken sayısı arttıkça incelenen olasılıklar üstel biçimde artar. Örneğin n değişkenli sırt çantası problemini ele alalım; burada incelenmesi gereken olasılık sayısı 2^n dir. Yani n değeri arttıkça işlem yoğunluğu artar ve çok büyük n değerleri için şu andaki mevcut bilgisayarlarla çözüm süresi yüz yıllar alabilir.

Kesme yöntemi

Kesme yöntemi tamsayılı doğrusal problemleri çözmek için ilk önerilmiş yöntemdir. Yöntemin şeması ilk defa Dantzig tarafından açıklanmış sonrasında ise Gomory tarafından geliştirilmiştir.

Kesme yöntemi, tamsayılı programlama probleminin doğrusal programlama problemine gevşetilmesine dayanır. Yani tamsayılı bir problemdeki tüm kısıtlar olduğu gibi ele alınırken, değişkenlerin tamsayı olması kısıtı kaldırılır. Bu şekilde doğrusal programlama metotlarından her hangi biri kullanılarak sonuç bulunur; bulunan sonuç tamsayı ise sorun yoktur; değilse tamsayı olmayan çözümleri dışarıda bırakacak aynı zamanda tüm tamsayı çözümleri koruyacak yeni bir kısıt (kesme) eklenir. Bu işlemlere optimal tamsayı çözüm bulunana kadar devam edilir.

Kesme yöntemlerinin genelde düzensiz olduğu bilinmektedir; ayrıca tüm verilerin simpleks tablo şeklinde saklanması büyük boyutlu problemler için sorun yaratmaktadır.

Dinamik programlama

Kesin çözüm yöntemlerinin içinde dinamik programlama özel bir yere sahiptir. Bu yöntem, örtüşen alt problemler ve optimal alt yapı özelliklerini sergiler; dolayısıyla toplam işlem sayısını ve kullanılan zamanı azaltır. Dinamik programlama algoritmalarının karmaşıklığı teorik olarak oldukça kolay hesaplanabilir ve bu değer reel karmaşıklığa yakındır (Cormen et al., 2001). Ancak iteratif bir yöntem olması dolayısıyla kısıt ve değişken sayısının çok olması durumunda ya da çok büyük katsayılar için etkili değildir.

Dal-sınır yöntemi

Dal-sınır algoritması ise optimal çözüme ulaşmak için sistemli bir şekilde uygun çözümleri saymaya dayanır. Ancak sayma yöntemindeki gibi tüm olasılıklar değil, olasılıkların daha küçük bir kısmı incelenir, ümit vermeyen bazı olasılıklar sayma aşamasında kesip atılır. Dal-sınır algoritmasının avantajı hızlı olması, verilmiş her hangi bir tamsayılı problemin özelliklerinin göz önünde bulundurulması ve yöntemin içinde başka tamsayılı metotların kullanılmasına imkan vermesidir. Hesaplama sürecinde ortaya çıkan çözümler genellikle iyi yaklaşık çözümlerdir. Ancak bir dezavantajı bilgisayar belleğine daha çok gerek duymasıdır. Ayrıca değişken sayısının artması hesaplama süresini üstel olarak artırır ve bulunan çözümün optimal çözüm olduğunu ispatlamak çalışma zamanından daha fazla zaman gerektirebilir (Papadimitriou and Steiglitz, 1982).

Dal-kesme yöntemi

Dal-kesme algoritması hibrid bir yaklaşım olup dal-sınır algoritması ile kesme yönteminin birleşmesinden oluşur. Verilen problem için öncelikle kesme yöntemi kullanılır, bu süreç bir tamsayı çözüm bulana kadar veya daha fazla kesme bulunamayıncaya kadar devam eder. Bu aşamada dal-sınır algoritması devreye girer, bulunan tamsayıya göre dallanarak çözüm bulununcaya kadar çalışmasını sürdürür.

4.2. Yaklaşık Yöntemler

Yaklaşık yöntemler, optimizasyonda kesin yöntemlerin tersine optimal çözümü garanti etmezler. Bununla birlikte;

- Verilen problemin kesin çözüm yönteminin olmadığı durumlarda tek seçenek yaklaşık çözümlerdir.

- Uzun zamanda bulunan kesin çözümdense kısa zamanda bulunan yaklaşık çözüm daha değerlidir.

- Bilinen kesin çözüm yöntemleri yeteri kadar iyi değil; çünkü çoğu problem çözümü için zorluklarla karşılaşıyorsa;

- Kesin çözüm yönteminin getirdiği yoğun hesaplama ve çözüm zamanı söz konusu ise yaklaşık yöntemler tercih edilebilir.

4.2.1. Sezgiseller

Bilgisayar bilimlerinde sezgisel algoritma, ele alınan problemin özelliğine göre tasarlanmış bir algoritmadır ve çözümün doğruluğunun kanıtlanabilir olup olmadığını göz ardı eder. Ancak genellikle en iyi mümkün çözüme makul ölçüde yaklaşarak oldukça hızlı çözümler üretir. Sezgiseller en kötü durum düşünüldüğünde çok başarısız performans gösterebilirler; hatta kötü bir problem örneği seçildiğinde optimumdan

çok uzak bir sonuç döndürebilir ve/veya üstel çalışma zamanı gerektirebilir. Bununla beraber iyi sezgiseller bir problemin çoğu örneğinde en iyi yaklaşım algoritmalarının performansını geride bırakabilirler (Ausiello et al., 1999).

Literatürde yer alan sezgisellerin çoğu ya yinelemeli olarak tek bir uygun çözüm inşa eden "*yapıcı sezgisel*" ya da uygun çözümlerin bir kümesini inceleyip en iyi sonucu döndüren "*sayma sezgiseli*" dir. Yapıcı sezgiseller güçlü olarak problem bağımlıdırlar ve genellikle polinom zaman gerektirirler. Diğer yandan sayma sezgiselleri incelenen küme çok büyük olduğunda üstel çalışma zamanı gerektirebilirler.

Bazı iyi bilinen sezgiseller şunlardır:

- Greedy algoritmalar,
- Yerel arama (Local search),
- Genetik algoritmalar,
- Tabu araması (Tabu search),
- Karınca kolonisi (Ant colony) vs.

Bu tezde greedy türündeki algoritmalara yer verilmiştir.

Yerel arama

Yerel arama algoritmaları bazı diğer algoritmalar tarafından bulunmuş bir başlangıç çözümle çalışmaya başlar ve daha iyi bir komşu çözüme taşınma yoluyla yinelemeli olarak geçerli çözümü geliştirir. Çözümü daha fazla geliştirebilecek hiç bir çözüm bulunamadığı takdirde algoritma bir yerel optimumda sonlanır (Aarts and Lenstra, 1997).

Özel bir problem için bir yerel arama algoritması elde etmek için başlangıç uygun çözümü bulan bir algoritma ve bir komşuluk yapısına ihtiyaç vardır (Her hangi bir y uygun çözümü için y'nin tüm komşularını bulmak gerekmez; ancak tek bir tane iyileştirici komşu bulunması yeterlidir.). Açıktır ki NP-zor problemler için optimal çözüme ulaşmayı sağlayan bir komşuluk yapısı polinom zamanda bulunamaz; bu durumda yaklaşık çözüm üreten yerel arama algoritmaları araştırılır.

Genetik algoritmalar

Genetik algoritmalar en iyi olanın yaşamasını ve doğal seçim mekanizmasını temel alan algoritmalardır. Bu yöntemde bireylerin popülasyonuyla işe başlanır, 0-1' lerden oluşan her birey parametre uzayında bir noktayı temsil eder. Her nesilde her bir birey için amaç fonsksiyonunun sonucu değerlendirilir ve daha iyi olan bireyler seçilerek yeni bir popülasyon elde edilir. Özel olarak, yeni bir nesil aşağıdaki üç aşamayla elde edilir:

- Kalite (sağlamlık) ölçümü,
- Seçim,
- Yeni bireylerin oluşturulması.

Tabu araması

Tabu araması hafıza esaslı bir arama stratejisidir. Önceki aşamalarda elde edilen bilgi daha sonraki aşamalardaki yönelimleri belirlemek için kullanılmaktadır. Bu yöntemde amaç, miyop yaklaşımın neden olduğu risklerden kaçınmak için basit yerel aramayı genelleştirmektir. Tabu araması, sonuçta daha iyi bir yerel optimuma götürebilir düşüncesiyle bulunduğu çözümden daha kötü sonuç veren bir komşuya geçişe izin verir.

Greedy algoritmalar

Sezgisel bir yöntem olan Greedy türündeki algoritmalar tasarlama kolaylığı ve optimum çözüme sıkça iyi yaklaşımlar vermesi bakımından oldukça kullanışlıdır. Eğer verilen bir problem sınıfı için tasarlanan greedy algoritmasının optimal değeri ürettiği ispatlanabilirse daha hızlı olan greedy türündeki bu algoritmanın diğer optimizasyon yöntemlerine tercih edilmesi kaçınılmazdır. Greedy algoritmalar her zaman olmasa da bazı problemler için optimal çözümü verirler (Cormen et al., 2001). Bu

yöntem oldukça güçlü bir yöntem olup problemlerin geniş bir sınıfı için iyi çalışırlar. Örneğin, Dijkstra' nın en kısa yol algoritması, Chvátal' ın küme örtüsü sezgiseli ve Kruskal algoritması iyi bilinen greedy algoritmalarından bazılarıdır.

Genel olarak greedy algoritmalarının özellikleri aşağıdaki şekilde verilebilir:

- Bir seferde tek bir karar verir.
- Karar verirken yerel bilgiyi kullanır.
- Kararı verirken o an için en fazla faydayı almaya bakar; o nedenle açgözlü olarak adlandırılır. Başka bir deyişle global optimal çözüme götürebilir umuduyla yerel optimal seçimler yapar (Cormen et al., 2001).
- Açgözlülüğü hızlı karar vermeyi gerektir, herhangi bir adımda çözüm kümesinde elde edilen bir değer takip eden adımlarda değiştirilmez. Yani greedy algoritmaları asla seçimlerini tekrardan düşünmez.

Greedy yöntemi başlangıç olarak nesneleri bazı kriterlere göre sıraya koyar ve boş kümeden başlayarak çözüm kümesini genişletir. Yani yukarıda verilen ilk özellik kullanılır: Tek seferde tek nesne için karar verilir. Eğer sondaki çözüm uygun oluyorsa nesne o anki geçerli çözüme eklenir; aksi halde bir daha düşünülmemek üzere elenir.

Geedy algoritmasının çalışma zamanı başlangıçta n tane nesnenin sıralanması ve n tane uygunluk testi nedeniyle $O(n)+O(n\log n)=O(n\log n)$' dir (Martello and Toth, 1999). Ayrıca yaklaşık çözümün kalitesi başlangıçtaki sıralamaya bağlıdır. Açıktır ki her problem örneği için her zaman optimal bir sıralama vardır; ancak hesaplama açısından zor bir problemin tüm örnekleri için böyle bir sıralamayı polinom zamanda bulmak pek olası değildir. Bununla birlikte bazı durumlarda, greedy yönteminin iyi yaklaşık çözümler bulmasını sağlayacak basit sıralamalar bulunabilir (Ausiello et al., 1999).

4.2.2. Yaklaşım algoritmaları

Önemli uygulama alanlarında karşılaşılanlar da dahil olmak üzere çoğu tamsayılı optimizasyon problemi NP-zor problemdir. Dolayısıyla $P \neq NP$ olduğu sürece bu problemler için kesin çözüm aramak sadece zaman kaybıdır. Bu nedenle de polinom zamanlı algoritmalar kullanarak yaklaşık çözümler üretmek bilgisayar bilimleri ve matematiğin ilgi çekici konularından biri haline gelmiştir. NP-zor problemler kesin çözümler bulmak için elverişli değillerse de yakın optimal çözümler bulunabilir. Dolayısıyla yaklaşım algoritmalarının tasarımı aslında kesin çözüm veren algoritmaların tasarım işleminden pek de farklı değildir (Vazirani, 2001).

Tanım 4.1*:* Bir $\varepsilon -$ *yaklaşık algoritması,* Z^* optimal değeriyle verilen bir maksimizasyon [minimizasyon] probleminin her bir örneği için $Z^*(1-\varepsilon) \le Z$ [minimizasyon için $Z \le (1+\varepsilon)Z^*$] olacak şekilde bir Z çözüm değeri veren algoritmadır.

Tanım 4.2*:* Bir *polinomiyal zamanlı yaklaşım şeması* (Polynomial time approximation scheme- PTAS), belirli bir $\varepsilon > 0$ hata parametresi için $\varepsilon -$ yaklaşık algoritmadır ve çalışma zamanı n' ye bağlı bir polinomdur. Bilinen bütün PTAS lar tüm mümkün aday çözümlere bakarak optimal çözümde yer alan belirli bir nesneler kümesini tahmin etme fikrini izlerler (Kellerer et al., 2004).

KP için ilk PTAS Sahni tarafından önerilmiştir (1975). n, nesnelerin sayısı; k, verilen her hangi bir negatif olmayan tamsayı olmak üzere bu şemanın zaman karmaşıklığı $O(n^{k+1})$' dir.

Tanım 4.3*: Tam polinomiyal zamanlı yaklaşım şeması* (Fully polynomial time approximation scheme- FPTAS) ise PTAS' tan farklı olarak n ve $1/\varepsilon$' nun bir polinom zamanında çalışır. Problemdeki faydaları ölçeklendirme temel yaklaşımı izlenir; böylece farklı fayda değerlerinin sayısının azaltılmasıyla oluşan bu örneğe dinamik programlama yöntemi uygulanır. FPTAS daha çok yapısal bakımdan sırt çantası problemine benzeyen problemler için bilinmektedirler.

Ibarra ve Kim, KP için polinom zamanlı bir FPTAS elde etmişlerdir (1975). Bu algoritmada öncelikle nesneler katkılarına göre

büyük ya da küçük olarak sınıflandırılır. Daha sonra dinamik programlama kullanılarak sadece büyük olan nesneler için problem çözülür ve ardından greedy aşamasıyla algoritma sonlanır. Algoritmanın zaman karmaşıklığı belirli bir ε değeri için $O(n/\varepsilon^2)+O(n\log n)$' dir.

Teorem 4.1: Güçlü NP-tam problemleri için NP=P olmadığı sürece bir FPTAS yoktur.

Sırt çantası problemi güçlü NP-tam problem değildir ve FPTAS'a sahip olduğu bilinmektedir. Bununla birlikte güçlü olmayan her NP-tam problem için bir FPTAS olup olmadığı bilinmemektedir.

Bir yaklaşım algoritması için akla gelen en önemli sorulardan biri problemin doğru yaklaşılabilirliği; yani polinom zamanda ulaşılabilen en iyi performans oranıdır. Çeşitli problemler için yaklaşım algoritmaları, oran kalitesi ya da en kötü durum hata sınırı bakımından birbirlerinden oldukça farklıdır. Eğer bir algoritma her hangi bir ε sabitine göre $\varepsilon-$yaklaşık oluyorsa bu algoritma iyi olarak düşünülür. Yalnız karşılaşılan bir zorluk, elde edilmiş bir yaklaşım sonucunun daha da geliştirilip geliştirilemeyeceğinin belirlenmesi ve geliştirlirse ne boyutta olduğunu belirlemektir (Hochbaum, 1997).

4.2.3. Performans analizi ve garanti değer

Bir problem için tasarlanan bir algoritmanın çözüme ne kadar yaklaştığı, ne kadar çalışma zamanı ve belleğe ihtiyaç duyduğu bazı analizler sonucu belirlenir. Açıktır ki optimal çözüm üreten her algoritmanın performansı aynı derecede iyi değildir. Bununla birlikte sadece yaklaşık sonuç veren bir algoritma daha iyi bir hesaplama zamanıyla bu açığını kapatmalıdır.

Temel olarak, algoritmaları nitelendirmenin üç yolu vardır. İlk yaklaşım, algoritmayı farklı problem örneklerine uygulayıp sonuçları karşılaştırmaktır. Bu yöntemde yazılım, donanım ve uygulama kalitesi gibi faktörlerin çalışmayı etkilememesi için çok fazla özen gösterilmelidir. Ayrıca uygun test problemlerinin seçimi de büyük bir sorundur. İkinci yol, algoritmanın ortalama çalışma zamanını araştırmaktır. Burada gerçek hayattan örnekleri yansıtan uygun bir olasılıksal model bulmak başlı başına özveri gerektirir. Karşılaşılan teknik güçlüklerin yanı sıra sonuçların anlamlı olabilmesinin problemin boyutunun yeterince büyük olmasıyla ya da algoritmanın yeterince çok sayıda problem örneğine uygulanmasıyla bağlantılı olması da bir dezavantajdır.

Bir algoritmanın performansını değerlendirmede kullanılan üçüncü ve en yaygın olan yöntem çalışma zamanının ***en kötü durum*** analizidir. Bu yöntemde verilen bir problem ailesi için algoritmanın bulduğu çözümün en kötü durumda optimalden en fazla ne kadar farklı olduğu

belirlenir (Fisher, 1980). Burada problemin giriş parametrelerinden bağımsız öyle bir Δ değeri bulunur ki verilmiş ailedeki her bir bireysel maksimizasyon [minimizasyon] problemi için *f,* algoritmanın bulduğu çözüm değeri; *R,* optimal çözüm değeri olmak üzere $\Delta \geq f/R$ [$\Delta \leq f/R$] olur. Δ değeri genellikle algoritmanın ***garanti değeri*** olarak adlandırılır. İlerleyen bölümlerde 0-1 ve tamsayılı sırt çantası problemleri için bazı özel durumlar gösterilmiştir; bu durumlarda greedy algoritmalar bu problemler için bilinenlerden daha iyi garanti değerler verir.

5. 0-1 KP İÇİN GREEDY ALGORİTMALARI

Günümüzde sırt çantası problemleri üzerine birçok çalışma mevcuttur. Bu bölümde 0-1 sırt çantası probleminin maksimizasyon versiyonu için daha önce önerilmiş bir algoritma incelenecektir. Ardından, bu problem için tanımlanmış tümleyen problem ve bu probleme dayanarak tasarlanmış farklı bir algoritma ile garanti değerin iyileştirilmesi bazı teoremler yardımıyla incelenecektir.

5.1. 0-1 Maksimizasyon KP ve AMAX Algoritması

0-1 maksimizasyon sırt çantası problemi için $\Delta \geq 1/2$ garanti değerli Greedy algoritmaları bilinmektedir (Nuriyev, 1986). Bu algoritmalardan biri aşağıdaki gibi verilmiştir.

$$R = \max\left\{ \sum_{j\in J} p_j x_j \,\middle|\, \sum_{j\in J} a_j x_j \leq b,\ x_j \in \{0,1\},\ j \in J \right\}, \qquad J = \{1,2,...,n\}$$

için R optimal çözüm f ise algoritmanın bulduğu çözüm olsun. Genelliği kaybetmeden;

$$b, a_j \in Z^+ \ \text{ve}\ a_j < b,\ j \in J,$$
$$\sum_{j\in J} a_j > b$$

olduğu ve değişkenlerin

$$\frac{p_1}{a_1} \geq \frac{p_2}{a_2} \geq \ldots \geq \frac{p_n}{a_n} \tag{5.1.1}$$

şeklinde sıralı olduğu kabul edilir. Bu durumda nesneler AMAX algoritması için $J = \{1, \ldots, n\}$ sırasında alınır.

AMAX algoritması

A1) $\sum_{j=1}^{k} a_j \leq b < \sum_{j=1}^{k+1} a_j$ durumunu sağlayan *k* indisini bulunur.

A2) $f = \max\left\{\sum_{j=1}^{k} p_j, p_{k+1}\right\}$

A3) $f = \sum_{j=1}^{k} p_j$ ise $\begin{cases} x_j = 1, & j = \overline{1,k} \\ x_j = 0, & j = \overline{k+1,n} \end{cases}$

$f = p_{k+1}$ ise $\begin{cases} x_{k+1} = 1 \\ x_j = 0, & i \neq k+1 \end{cases}$

A4) SON.

0-1 maksimizasyon KP için

$$\frac{1}{2} R \leq f \leq R \tag{5.1.2}$$

olduğu gösterilmiştir (Nuriyev, 1986).

5.2. 0-1 KP için Tümleyen Problem ve AMİN Algoritması

Bazı optimizasyon problemleri minimizasyon ya da maksimizasyon olarak denk biçimde ele alınabilir; bu durumda biri diğerinin tümleyeni olarak bilinir (Güntzer and Jungnickel, 2000). Yukarıdaki problemin tümleyeni şu şekilde ifade edilmiştir (Nikitin ve Nuriyev, 1983) :

$B = \sum_{j \in J} a_j,\ \ \bar{b} = B - b,\ \ y_j = 1 - x_j$ olmak üzere,

$$\bar{R} = \min \left\{ \sum_{j \in J} p_j y_j \,\middle|\, \sum_{j \in J} a_j y_j \le \bar{b},\ x_j \in \{0,1\},\ j \in \bar{J} \right\}$$

0-1 Minimizasyon KP için $\Delta \le 2$ garanti değerli algoritmalar da mevcuttur (Nuriyev, 1986). Bu algoritmalardan biri aşağıdaki gibi olup (5.1.1) kabulü geçerli olduğundan nesneler $\bar{J} = \{n, n-1, ..., 1\}$ sırasında alınmaktadır.

AMİN algoritması

A1) $\bar{f} = \sum_{j \in \bar{J}} p_j$ olsun.

A2) $\bar{k} = \min \left\{ l \left| \sum_{j=1}^{l} a_j > \bar{b} \right. \right\}$ bulunur.

A3) $L = \sum_{j=1}^{\bar{k}} p_j$

A4) $\bar{f} = \min \left\{ \bar{f}, L \right\}, \ \bar{J} = \bar{J} / \bar{k}$

A5) $\sum_{j \in \bar{J}} a_j \geq \bar{b}$ ise Adım2' ye geçilir.

A6) SON.

0-1 minimizasyon KP için

$$\bar{R} \leq f \leq 2\bar{R} \qquad (5.2.1)$$

olduğu gösterilmiştir (Nuriyev, 1986).

5.3. 0-1 KP için Bazı Teoremler

Aşağıdaki teoremlerin doğruluğu (Nuriyev, 1986), (Nuriyev ve Dündar, 1998) ve (Nuriyev vd., 2004) çalışmalarında gösterilmiştir.

Teorem 5.3.1: $P = \sum_{j=1}^{n} p_j$ olmak üzere $R + \bar{R} = P$ dir.

Teorem 5.3.2: $P - \bar{f} \geq f$ dir.

Teorem 5.3.3: $R \geq \bar{\bar{f}} \geq \begin{cases} (1/2)R, & \text{eğer } \alpha < 1/3 \text{ ise,} \\ (2\alpha/(1+\alpha))R, & \text{eğer } \alpha \geq 1/3 \text{ ise,} \end{cases}$

Burada $\bar{\bar{f}} = P - \bar{f}$ ve $\alpha = \bar{\bar{f}}/P$ dir.

Teorem 5.3.4: $\bar{R} \leq \bar{f} \leq \begin{cases} 2\bar{R}, & \text{eğer } \beta < 2/3 \text{ ise,} \\ (\beta/(2\beta-1))\bar{R}, & \text{eğer } \beta \geq 2/3 \text{ ise,} \end{cases}$

Burada $\beta = \bar{f}/P$ dir.

6. TAMSAYILI KP İÇİN GREEDY ALGORİTMALARI

Bu bölümde tamsayılı sırt çantası probleminin (Integer knapsack problem- IKP) minimizasyon ve maksimizasyon versiyonları ele alınmıştır. Bu problemler için greedy algoritmaları önerilmiş ve algoritmaların garanti değerleri hesaplanmıştır. Maksimum IKP için hesaplanan garanti değerin iyileştirilmesi amacıyla maksimum IKP'nin tümleyen problemi tanımlanmış ve probleme uygun tasarlanmış garanti değerli yeni bir greedy algoritması ile elde edilen sonuçlar incelenmiştir.

6.1. Minimizasyon KP için Greedy Algoritması (Tmin1)

$$\overline{R}_Z = \left\{ \sum_{j\in J} p_j x_j \mid \sum_{j\in J} a_j x_j \geq b,\ x_j \in Z^+ \cup \{0\},\ j \in \overline{J} \right\} \tag{6.1.1}$$

$\overline{J} = \{1,2,...,n\}$ ve $Z^+ = \{1,2,...\}$ şeklinde tanımlı problem için $\overline{R}_Z$ optimal çözüm değeri, $\overline{f}_Z$ algoritmanın bulduğu çözüm değeri ve $\overline{X}^G$ algoritmanın bulduğu çözüm vektörü olsun. Değişkenlerin,

$$\frac{p_1}{a_1} \leq \frac{p_2}{a_2} \leq ... \leq \frac{p_n}{a_n} \tag{6.1.2}$$

şeklinde sıralı olduğu kabul ediliyor.

Tmin1

A1) $x_1 = \left\lceil \frac{b}{a_1} \right\rceil$ değeri hesaplanır;

A2) $\overline{f}_Z = x_1 p_1$**;**

A3) $x_2 = x_3 = ... = x_n = 0$**;**

A4) $\overline{X}^G, \overline{f}_Z$ sonuçları yazdırılır**;**

A5) SON.

Tmin1 algoritmasının garanti değer hesabı

$\overline{f}_Z$, algoritmanın bulduğu çözüm değeri ve $\overline{\overline{R}}_Z$, problemin optimal çözüm değeri olmak üzere garanti değerin sınırlarını belirleyelim:

$\overline{f}_Z = \left\lceil \frac{b}{a_1} \right\rceil p_1$ ve $\overline{R}_Z \geq \left\lfloor \frac{b}{a_1} \right\rfloor p_1$ olduğu göz önünde bulunduruluyor.

$$\Delta = \frac{\overline{f}_Z}{\overline{\overline{R}}_Z} \leq \frac{\left\lceil \frac{b}{a_1} \right\rceil p_1}{\left\lfloor \frac{b}{a_1} \right\rfloor p_1} = \frac{\left\lfloor \frac{b}{a_1} \right\rfloor + 1}{\left\lfloor \frac{b}{a_1} \right\rfloor} = 1 + \frac{1}{\left\lfloor \frac{b}{a_1} \right\rfloor}$$

$$\left\lfloor \frac{b}{a_1} \right\rfloor = k \text{ dersek; } \Delta \le \frac{k+1}{k} = \omega$$

Dikkat edilirse en kötü durumda, k=1 için, algoritmanın garanti değeri 2 olacak; aksi halde $k \to \infty$ için daha iyi sonuçlar gönderecektir.

$$\overline{R}_Z \le \overline{f}_Z \le \omega \overline{R}_Z \tag{6.1.3}$$

6.2. Tmax Algoritması ve Garanti Değer Hesabı

$$R_Z = \max \left\{ \sum_{j \in J} p_j x_j \,\middle|\, \sum_{j \in J} a_j x_j \le b,\ x_j \in Z^+ \cup \{0\},\ j \in J \right\} \tag{6.2.1}$$

$J = \{1, 2, ..., n\}$, şeklinde tanımlı problem için R_Z optimal çözüm değeri, f_Z algoritmanın bulduğu çözüm değeri ve X^G algoritmanın bulduğu çözüm vektörü olsun.

Değişkenlerin (5.1.1)'deki gibi sıralı olduğu kabul ediliyor.

Tmax

A1) $k = 1$ ve $f_Z = 0$ atamaları yapılır;

A2) $x_k^G = \left\lfloor \frac{b}{a_k} \right\rfloor$;

A3) $f_Z = f_Z + p_k x_k^G$; $b = b - a_k x_k^G$;

A4) $k = k + 1$;

A5) $k > n$ ise Adım7' ye geçilir;

A6) $a_k \leq b$ ise Adım2' ye geçilir;

A7) $x_k^G = x_{k+1}^G = ... = x_n^G = 0$

A8) X^G, f_Z sonuçları yazdırılır;

A9) SON.

Tmax algoritmasının garanti değer hesabı

f_Z, algoritmanın bulduğu çözüm değeri ve R_Z, problemin optimal çözüm değeri olmak üzere garanti değer sınırlarını belirleyelim:

($\forall j \in J$ için $a_j \leq b$ kabul edilmiştir.)

$R_Z \le \frac{b}{a_1} p_1$ ve $f_Z \ge \left\lfloor \frac{b}{a_1} \right\rfloor p_1$ olduğundan;

$$\Delta \ge \frac{\left\lfloor \frac{b}{a_1} \right\rfloor p_1}{\frac{b}{a_1} p_1} \ge \frac{\left\lfloor \frac{b}{a_1} \right\rfloor}{\left\lfloor \frac{b}{a_1} \right\rfloor + \left\{ \frac{b}{a_1} \right\}} \ge \frac{\left\lfloor \frac{b}{a_1} \right\rfloor}{\left\lfloor \frac{b}{a_1} \right\rfloor + 1} = \frac{1}{1 + \frac{1}{\left\lfloor \frac{b}{a_1} \right\rfloor}}$$

Burada $\left\lfloor \frac{b}{a_1} \right\rfloor = m$ dersek sonuç $\frac{1}{1+\frac{1}{m}}$ olacak yani;

$$\Delta \ge \frac{1}{1+\frac{1}{m}} \quad \Rightarrow \quad \lim_{m \to \infty} \frac{m}{m+1} = 1$$

olarak bulunur ki bu durumda m değeri ne kadar büyürse sonuç o kadar iyi çıkacak; ancak en kötü durumda, m=1 için, algoritmamız 1/2 garanti değeriyle sonuç verecektir.

$$\left(\frac{m}{m+1} \right) = \alpha \qquad \Rightarrow \qquad \alpha R_Z \le f_Z \le R_Z \tag{6.2.2}$$

6.3. Maksimizasyon IKP için Tümleyen Problem

Yukarıdaki maksimizasyon probleminde her bir x_j için

$$n_j = \left\lfloor \frac{b}{a_j} \right\rfloor$$

değerini oluşturalım. Burada n_j, *j.* değişkenden en fazla kaç tane alınabileceğini gösterir.

$B = \sum_{j \in J} n_j a_j$, $\bar{b} = B - b$, $y_j = n_j - x_j$ olmak üzere tümleyen problemi şu şekilde oluştururuz:

$$\bar{\bar{R}}_Z = \min \left\{ \sum_{j \in \bar{J}} p_j y_j \,\middle|\, \sum_{j \in \bar{J}} a_j y_j \geq \bar{b},\ y_j \in Z^+ \cup \{0\},\ y_j \leq n_j,\ j \in \bar{\bar{J}} \right\} \qquad (6.3.1)$$

$\bar{\bar{J}} = \{1, 2, ..., n\}$. Görüldüğü üzere burada sınırlı tamsayılı bir minimizasyon problemi elde edilmiştir. $\bar{\bar{R}}_Z$, problemin optimal çözüm değeri; $\bar{\bar{f}}_Z$ ve Y^G sırasıyla algoritmanın bulduğu çözüm değeri ve çözüm vektörü; $\tilde{\bar{R}}$ ise değişkenlerin (2.2.1)'deki gibi sürekli olduğu minimizasyon probleminin optimal çözüm değeridir. Genelliği bozmadan,

a_j ve p_j, $j \in \overline{\overline{J}}$

değerlerinin pozitif tamsayı olduğu ve (6.1.2) sıralaması kabul edilmektedir.

6.3.1. Tmin algoritması ve garanti değer hesabı

Tmin

A1) $k = 1$ ve $\overline{\overline{f}}_Z = 0$ olarak atama yapılır;

A2) $y_k^G = \left\lceil \dfrac{\overline{b}}{a_k} \right\rceil$;

A3) $y_k^G > n_k$ ise $y_k^G = n_k$ olarak alınır, $\overline{\overline{f}}_Z = \overline{\overline{f}}_Z + y_k^G p_k$, $\overline{b} = \overline{b} - y_k^G a_k$;

aksi halde $\overline{\overline{f}}_Z = \overline{\overline{f}}_Z + y_k^G p_k$ olarak hesaplanır ve Adım6' ya gidilir;

A4) $k = k + 1$;

A5) $\overline{b} > 0$ ve $k \le n$ ise Adım2' ye gidilir;

A6) Y^G, $\overline{\overline{f}}_Z$ sonuçları yazdırılır;

A7) SON.

Tmin algoritmasının garanti değer hesabı

Öncelikle indis kümesinin birim katkılara göre artan sırada düzenlenmiş olduğunu kabul edelim, yani (6.1.2) sıralaması geçerli olsun.

Algoritmada öyle bir indis vardır ki o indise kadar $y_j^G = n_j$ olacak; ancak tam o indiste $y_k^G = \left\lfloor \frac{\bar{b}}{a_k} \right\rfloor$ alınarak algoritma sonlandırılacaktır. Bu indisi ***s*** olarak alırsak;

$s-1 = \max\left\{ k \left| \sum_{j=1}^{k} n_j a_j < \bar{b} \right. \right\}$ olur.

Dolayısıyla algoritmanın sonucunu şu şekilde ifade edebiliriz:

$$y_i^G = n_i, \qquad i = \overline{1, s-1}$$

$$y_i^G = 0, \qquad i = \overline{s+1, n}$$

$$y_s^G = \left\lfloor \frac{\bar{\bar{b}}}{a_s} \right\rfloor, \quad \left(\bar{\bar{b}} = \bar{b} - \sum_{i=1}^{s-1} n_i a_i \right)$$

$$\bar{\bar{f}}_Z = \sum_{i=1}^{s-1} n_i p_i + \left\lfloor \frac{\bar{\bar{b}}}{a_s} \right\rfloor p_s$$

Burada algoritmanın üçüncü adımına dikkat edilirse;

$$\left\lceil \frac{\bar{\bar{b}}}{a_s} \right\rceil \le n_s \text{ dir.} \qquad (6.3.2)$$

Problem sürekli olsaydı çözüm;

$$\tilde{\bar{R}} = \sum_{i=1}^{s-1} n_i p_i + \frac{\bar{\bar{b}}}{a_s} p_s$$

şeklinde olacaktı. Biliyoruz ki;

$$\tilde{\bar{R}} \le \bar{R}_Z \le \bar{f}_Z \;\Rightarrow\; 2\tilde{\bar{R}} \le 2\bar{R}_Z \le 2\bar{f}_Z$$

olacaktır. 2 durumu inceleyelim;

<u>1. durum</u>: s>1 ise;

$a_{\max} = \max\left\{ a_j \,|\, j = 1,...,n \right\}$ olsun.

$$\Delta = \frac{\overline{f}_Z}{\overline{R}_Z} \le \frac{\overline{f}_Z}{\overline{\tilde{R}}} = \frac{\sum_{i=1}^{s-1} n_i p_i + \left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil p_s}{\sum_{i=1}^{s-1} n_i p_i + \frac{\overline{\overline{b}}}{a_s} p_s} = \frac{\left(\sum_{i=1}^{s-1} n_i p_i + \frac{\overline{\overline{b}}}{a_s} p_s \right) + \left(\left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil p_s - \frac{\overline{\overline{b}}}{a_s} p_s \right)}{\sum_{i=1}^{s-1} n_i p_i + \frac{\overline{\overline{b}}}{a_s} p_s}$$

$$= 1 + \frac{\left(\left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil - \frac{\overline{\overline{b}}}{a_s} \right) p_s}{\sum_{i=1}^{s-1} n_i p_i + \frac{\overline{\overline{b}}}{a_s} p_s} \le 1 + \frac{\left(\left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil - \frac{\overline{\overline{b}}}{a_s} \right) p_s}{\sum_{i=1}^{s-1} \left\lfloor \frac{b}{a_i} \right\rfloor p_i} \le 1 + \frac{\left(\left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil - \frac{\overline{\overline{b}}}{a_s} \right) p_s}{\sum_{i=1}^{s-1} \frac{b - a_i}{a_i} p_i}$$

$$\le 1 + \frac{\left(\left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil - \frac{\overline{\overline{b}}}{a_s} \right) p_s}{\sum_{i=1}^{s-1} \frac{b - a_{\max}}{a_i} p_i} \le 1 + \frac{p_s}{(b - a_{\max}) \sum_{i=1}^{s-1} \frac{p_i}{a_i}}$$

$$\le 1 + \frac{p_s}{(b - a_{\max}) \frac{p_1}{a_1} \sum_{i=1}^{s-1} 1}$$

Burada $\frac{p_s}{\frac{p_1}{a_1}} = q$ dersek;

$$\Delta \le 1 + \frac{q}{(b - a_{\max}) \sum_{i=1}^{s-1} 1}$$

olacaktır.

2. durum: s=1 ise;

Bu durumda algoritma tek adımda sonlanıyor demektir. Yukarıda bulunan değerde s=1 için sonuç tanımsız olacağından yeni bir ispat gereklidir. Dikkat edilirse

$\bar{\bar{b}} = \bar{b}$ ve $\sum_{i=1}^{s-1} n_i p_i = 0;$ ayrıca

$\forall i \in I$ için $\left\lceil \frac{\bar{b}}{a_i} \right\rceil \geq 1, \quad \left\lfloor \frac{\bar{b}}{a_i} \right\rfloor \geq 1, \quad \frac{\bar{b}}{a_i} \geq 1$

dir.Bu durumda;

$$\bar{f}_Z = \sum_{i=1}^{s-1} n_i p_i + \left\lceil \frac{\bar{\bar{b}}}{a_s} \right\rceil p_s = \left\lceil \frac{\bar{b}}{a_1} \right\rceil p_1 \qquad \tilde{\tilde{R}} = \frac{\bar{b}}{a_1} p_1$$

$$\left\lceil \frac{\bar{b}}{a_1} \right\rceil = \left\lfloor \frac{\bar{b}}{a_1} \right\rfloor + 1 \leq \left\lfloor \frac{\bar{b}}{a_1} \right\rfloor + \left\lfloor \frac{\bar{b}}{a_1} \right\rfloor \leq 2 \left\lfloor \frac{\bar{b}}{a_1} \right\rfloor + 2 \left\{ \frac{\bar{b}}{a_1} \right\}$$

$$\bar{f}_Z = \left\lceil \frac{\bar{b}}{a_1} \right\rceil p_1 \leq \left(2 \left\lfloor \frac{\bar{b}}{a_1} \right\rfloor + 2 \left\{ \frac{\bar{b}}{a_1} \right\} \right) p_1 = 2\tilde{\tilde{R}} \leq 2\bar{R}_Z$$

Sonuç olarak, *Tmin* algoritması s=1 için ve s>1 olduğunda en kötü durumda; yani $\frac{q}{(b-a_{\max})\sum_{i=1}^{s-1}1}$=1 iken, 2 garanti değerini; aksi halde daha iyi sonuçlar verecektir. Garanti değeri δ ile gösterirsek;

$$\overline{R}_Z \le \overline{f}_Z \le \delta \overline{R}_Z \qquad (6.3.3)$$

şeklinde belirtebiliriz.

Burada $\delta = \begin{cases} 2, & \text{s=1 ise;} \\ (1+\gamma), & \text{s>1 ise;} \end{cases}$ ve $\frac{q}{(b-a_{\max})\sum_{i=1}^{s-1}1} = \gamma$ ' dir.

6.4. IKP için Bazı Teoremler

Teorem 6.1: $R_Z + \overline{\overline{R}}_Z = P \qquad (P = \sum_{i=1}^{n} n_i p_i)$

İspat: X ve Y sırasıyla R_Z ve $\overline{\overline{R}}_Z$ optimal değerleri için optimal çözüm vektörleri olsun.

$$\overline{\overline{R}}_Z = \sum_{i=1}^{n} y_i p_i = \sum_{i=1}^{n} (n_i - x_i)\, p_i = \sum_{i=1}^{n} n_i p_i - \sum_{i=1}^{n} x_i p_i = P - R_Z$$

$$R_Z + \overline{\overline{R}}_Z = P$$

Teorem 6.2: $f_Z + \overline{\overline{f}}_Z < P$

İspat: Farzedelim ki aşağıdaki (5.1.1) sıralaması

$\frac{p_1}{a_1} \geq \frac{p_2}{a_2} \geq ... \geq \frac{p_n}{a_n}$ geçerli olsun.

Yani *Tmax* algoritması nesneleri {1,2,…,n} sırasında alırken *Tmin* algoritması ise {n,n-1,...,1} sırasında alsın.

Tmax algoritmasına baktığımızda sadece x_1 değişkeni n_1 değerini alabilecek ve b sınırı sürekli güncellendiği için diğer değişkenler n_i 'den küçük değerler alacaktır. *Tmax* algoritmasında belirlediğimiz sınır k-1 idi. Yani (k-1). değişkene kadar $x_i = \left\lfloor \frac{b}{a_i} \right\rfloor$ (güncellenen b değerine göre), k. değişkenden itibarense $x_i = 0$ olacaktır.Buna göre;

$$f_Z < \sum_{i=1}^{k-1} n_i p_i$$

Minimalleştirme algoritması için

$$\overline{\overline{f}}_Z = \sum_{i=1}^{s-1} n_i p_i + \left\lceil \frac{\overline{\overline{b}}}{a_s} \right\rceil p_s$$

demiştik; ancak bu söylediğimiz (6.1.2) sıralaması kabulümüzde geçerliydi. Şimdi her şeyi (5.1.1) sıralamasına göre düzenler ve s=k olarak alırsak

$$\overline{\overline{f}}_Z = \left\lceil \frac{\overline{\overline{b}}}{a_k} \right\rceil p_k + \sum_{i=k+1}^{n} n_i p_i$$

olacaktır. Ayrıca (6.3.2)'den hatırlayacağımız üzere $\left\lceil \frac{\overline{\overline{b}}}{a_k} \right\rceil \le n_k$ idi. Bu durumda;

$$f_Z + \overline{\overline{f}}_Z < \sum_{i=1}^{k-1} n_i p_i + \left\lceil \frac{\overline{\overline{b}}}{a_k} \right\rceil p_k + \sum_{i=k+1}^{n} n_i p_i \le \sum_{i=1}^{k-1} n_i p_i + n_k p_k + \sum_{i=k+1}^{n} n_i p_i = \sum_{i=1}^{n} n_i p_i = P$$

$$f_Z + \overline{\overline{f}}_Z < P$$

Garanti Değerin İyileştirilmesi

(6.2.1) probleminin çözümü için (6.3.1) problemini ele alalım. *Tmin* algoritmasını bu problem için uygulayarak $\hat{f}_Z = P - \overline{\overline{f}}_Z$ diyelim.

Teorem 6.3: $R_Z \geq \hat{f}_Z \geq \begin{cases} \alpha R_Z, & \mu < \alpha/(2-\alpha) \text{ ise} \\ (2\mu/(1+\mu))R_Z, & \mu \geq \alpha/(2-\alpha) \text{ ise} \end{cases}$

Burada $\mu = \hat{f}_Z / P$' dir.

İspat: (6.3.3)' den en kötü durumda $\overline{\overline{f}}_Z \leq 2\overline{\overline{R}}_Z$ ve Teorem (6.1)' den $R_Z + \overline{\overline{R}}_Z = P$ idi. Bunlara göre;

$$\hat{f}_Z = P - \overline{\overline{f}}_Z \geq P - 2\overline{\overline{R}}_Z = P - 2\,(P - R_Z) = 2R_Z - P = 2R_Z - \hat{f}_Z/\mu \quad \text{olup;}$$

$$\hat{f}_Z \geq 2R_Z - \hat{f}_Z/\mu \;\Rightarrow\; \hat{f}_Z(1+1/\mu) \geq 2R_Z \;\Rightarrow\; \hat{f}_Z \geq (2\mu/(1+\mu))R_Z$$

bulunur. Teorem (6.2)' den $\hat{f}_Z = P - \overline{\overline{f}}_Z > f_Z$ ve (6.2.2)' den $f_Z \geq \alpha R_Z$ olduğu da göz önünde bulundurulursa;

$$\hat{f}_Z \geq f_Z \geq \alpha R_Z \tag{6.4.1}$$

Ancak bulunan $(2\mu/(1+\mu))$ değerine baktığımızda eğer $\mu < \alpha/(2-\alpha)$ ise bu değer α' dan küçük olup sınırlar (6.4.1)' deki gibi kalacaktır. Fakat $\mu \geq \alpha/(2-\alpha)$ olduğu anda $(2\mu/(1+\mu)) \geq \alpha$ olacağından teorem gerçeklenmiş olur.

Bu durumda görülür ki $\mu \geq \alpha/(2-\alpha)$ için önerilen algoritmanın garanti değeri, bilinen algoritmaların garanti değeri olan α ' dan daha iyidir.

Teorem 6.4: $\overline{\overline{R}}_Z \leq \overline{\overline{f}}_Z \leq \begin{cases} \delta \overline{\overline{R}}_Z, & \lambda < (\delta\alpha - \delta)/(\alpha - \delta) \text{ ise} \\ \left(\alpha\lambda/(\lambda + \alpha - 1)\right)\overline{\overline{R}}_Z, & \lambda \geq (\delta\alpha - \delta)/(\alpha - \delta) \text{ ise} \end{cases}$

Burada $\lambda = \overline{\overline{f}}_Z / P$ ' dir.

İspat: Teorem (6.3)' den $\hat{f}_Z \geq \alpha R_Z$ ve Teorem (6.1)' den $R_Z + \overline{\overline{R}}_Z = P$ idi. Bunlara göre;

$$\overline{\overline{f}}_Z = P - \hat{f}_Z \leq P - \alpha R_Z = P - \alpha\left(P - \overline{\overline{R}}_Z\right) = (1-\alpha)P + \alpha\overline{\overline{R}}_Z$$
$$= (1-\alpha)\overline{\overline{f}}_Z/\lambda + \alpha\overline{\overline{R}}_Z$$

$$\overline{\overline{f}}_Z - (1-\alpha)\overline{\overline{f}}_Z/\lambda \leq \alpha\overline{\overline{R}}_Z \Rightarrow \overline{\overline{f}}_Z\left(1 + (\alpha-1)/\lambda\right) \leq \alpha\overline{\overline{R}}_Z \Rightarrow \overline{\overline{f}}_Z \leq \left(\alpha\lambda/(\lambda+\alpha-1)\right)\overline{\overline{R}}_Z$$

elde edilir.

Bu durumda görülür ki $\lambda \geq (\delta\alpha - \delta)/(\alpha - \delta)$ için önerilen algoritmanın garanti değeri, bilinen algoritmaların garanti değeri olan δ ' dan daha iyidir.

6.5. Örnek Uygulama

Aşağıda verilen tamsayılı maksimizasyon problemi için öncelikle *Tmax* algoritmasını uygulayalım. Daha sonra ise problemin tümleyenini bularak bu probleme *Tmin* algoritmasını uygulayalım ve sonuçları karşılaştıralım.

$$9x_1 + 5x_2 + 3x_3 + 7x_4 + 10x_5 \le 52$$
$$x_j \in Z^+ \cup \{0\}, \quad j = \overline{1,5}$$
$$k.a$$

$$enb\ R_Z = 34x_1 + 18x_2 + 10x_3 + 21x_4 + 28x_5$$

tamsayılı problemi veriliyor. Bu problem için optimal çözüm değeri $R_Z = 192$ ve optimal çözüm vektörü $X = \{4,2,2,0,0\}$ olduğu bilinmektedir.

Örnekte (5.1.1) sıralaması mevcut olduğu için her hangi bir sıralama yapmaya gerek yoktur. Algoritmayı uygularsak ilk iterasyon için

$$x_1^G = \left\lfloor \frac{52}{9} \right\rfloor = 5 \quad f_Z = 0 + 5*34 \text{ ve } b = 52 - 45 = 7$$

olacaktır.

$a_2 = 5 < b = 7$ olduğundan ikinci iterasyonda;

$$x_2^G = \left\lfloor \frac{7}{5} \right\rfloor = 1 \quad f_Z = 170 + 1*18 \text{ ve } b = 7 - 5 = 2$$

olacaktır.

$a_3 = 3 > b = 2$ olduğundan algoritma burada sonlanacaktır. Yani algoritmanın bulduğu değer ve çözüm vektörü

$$f_Z = 188, \; X^G = \{5,1,0,0,0\} \text{ olmuştur.} \qquad (6.5.1)$$

$\left\lfloor \frac{b}{a_1} \right\rfloor = m = 5$ ve $\left(\frac{m}{m+1} \right) = \alpha = \frac{5}{6}$ olduğu hatırlanırsa (6.1.2)' ye göre aşağıdaki eşitsizlik gerçeklenmiştir:

$$\frac{5}{6} R_Z \le f_Z \le R_Z \;\rightarrow\; 170 \le 188 \le 192$$

Şimdi verilen problemin tümleyenini oluşturalım ve *Tmin* algoritmasını uygulayalım. Algoritma (6.1.2) sıralamasında çalıştığı için verileri buna göre yeniden düzenleyerek n_j değerlerini bulalım.

$n_j = \left\lfloor \frac{b}{a_j} \right\rfloor$ ve b=52 idi; buna göre bir tablo oluşturalım.

Çizelge 6.1 Tümleyen problem için kullanılacak veriler

	1	**2**	**3**	**4**	**5**
p_j	28	21	10	18	34
a_j	10	7	3	5	9
$n_j = \left\lfloor \frac{52}{a_j} \right\rfloor$	5	7	17	10	5

Tabloda nesneler (6.2.2)' ye göre düzenlenmiştir.

$$B = \sum_{j \in J} n_j a_j = 5*10+7*7+17*3+10*5+5*9 = 245,$$
$$\overline{b} = B - b = 245 - 52 = 193$$

olmak üzere problem şu şekilde ifade edilir:

$$10y_1 + 7y_2 + 3y_3 + 5y_4 + 9y_5 \geq 193$$
$$y_j \in Z^+ \cup \{0\},\ y_j \leq n_j, \quad j = \overline{1,5}$$
$$k.a$$

$$enk\ \overline{\overline{R}}_Z = 28y_1 + 21y_2 + 10y_3 + 18y_4 + 34y_5$$

İlk iterasyonda,

$y_1^G = \left\lceil \frac{\bar{b}}{a_1} \right\rceil = \left\lceil \frac{193}{10} \right\rceil = 20$ ancak $20 > n_1 = 5$ olduğu için $y_1^G = 5$ alınır ve $\bar{\bar{f}}_Z = 0 + 5*28 = 140,\ \bar{b} = 193 - 5*10 = 143$ bulunur.

İkinci iterasyonda,

$y_2^G = \left\lceil \frac{\bar{b}}{a_2} \right\rceil = \left\lceil \frac{143}{7} \right\rceil = 21$ ancak $21 > n_2 = 7$ olduğundan $y_2^G = 7,\ \bar{\bar{f}}_Z = 140 + 7*21 = 287$ ve $\bar{b} = 143 - 7*7 = 94$ bulunur.

Üçüncü iterasyonda,

$y_3^G = \left\lceil \frac{\bar{b}}{a_3} \right\rceil = \left\lceil \frac{94}{3} \right\rceil = 32$ ancak $32 > n_3 = 17$, dolayısıyla $y_3^G = 17,\ \bar{\bar{f}}_Z = 287 + 17*10 = 457$ ve $\bar{b} = 94 - 17*3 = 43$ olur.

Dördüncü iterasyonda,

$y_4^G = \left\lceil \frac{\bar{b}}{a_4} \right\rceil = \left\lceil \frac{43}{5} \right\rceil = 9$ bulunur; bu durumda $9 < n_4 = 10$ olduğu için algoritma $y_4^G = 9,\ \bar{\bar{f}}_Z = 457 + 9*18 = 619$ ve $\bar{b} = 43 - 45 = -2$ ile sonlanır.

Yani algoritmanın bulduğu değer ve çözüm vektörü

$$\overline{\overline{f}}_Z = 619, \quad Y^G = \{5,7,17,9,0\} \text{ olmuştur.} \tag{6.5.2}$$

Teorem (6.1)'den $R_Z + \overline{\overline{R}}_Z = P$ idi. Bu durumda tümleyen problemin optimal çözüm değeri $\overline{\overline{R}}_Z = P - R_Z = 615$ olmaktadır.

Dikkat edilirse

$$q = \frac{p_s}{\frac{p_1}{a_1}} = \frac{18}{\frac{28}{10}} \simeq 6.4, \quad \frac{q}{(b - a_{\max})(s-1)} = \gamma \text{ ve } a_{\max} = 10, \text{ s=4 için}$$

(6.3.3)'deki eşitsizlik sağlanır:

$$\overline{R}_Z \le \overline{f}_Z \le (1+\gamma)\overline{R}_Z \quad \rightarrow \quad 615 \le 619 \le (1+0.05)615 \simeq 646$$

Tümleyen problem yardımıyla çözdüğümüz başta verilen maksimizasyon probleminin sonucunu görebilmek için, sıralamayı da düşünerek, $x_j = n_j - y_j$ eşitliğini kullanalım. Bu durumda bulunan çözüm değeri ve çözüm vektörü

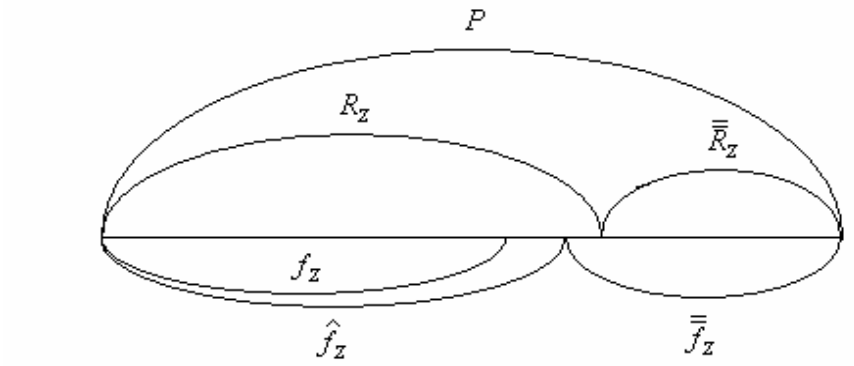
$$\hat{f}_Z = 188, \quad X^G = \{5,1,0,0,0\} \tag{6.5.3}$$

olmuştur.

7. SONUÇ

Tamsayılı programlama problemleri üzerine tasarlanan garanti değerli algoritmaları incelemek için yapılan bu çalışmada özel olarak, 0-1 sırt çantası ile tamsayılı sırt çantası problemlerinin maksimizasyon ve minimizasyon versiyonları incelenmiştir. Tamsayılı maksimizasyon sırt çantası problemi için bir greedy algoritması önerilerek garanti değeri hesaplanmıştır. Tümleyen problem kavramına dayanarak sınırlı tamsayılı minimizasyon problemi için yeni bir algoritma daha önerilmiş ve önceden hesaplanmış garanti değerin iyileştirilmesi amaçlanmıştır. Daha önce önerilen AMAX, AMİN algoritmaları ile önerdiğimiz Tmin1, Tmax ve Tmin algoritmalarının karmaşıklıkları, sıralamadan dolayı O(nlogn)' dir. Teoremler yardımıyla elde edilen çözümlerin geometrik yorumu aşağıdaki gibidir.

Şekil 7.1. Elde edilen çözümlerin geometrik yorumu



Burada;

$P = \sum_{j \in J} n_i a_i$,

R_Z = Tamsayılı maksimizasyon probleminin optimal çözüm değeri,

$\overline{\overline{R}}_Z$ = Sınırlı tamsayılı minimizasyon probleminin optimal çözüm değeri,

f_Z = Tmax algoritmasının bulduğu çözüm değeri,

$\overline{\overline{f}}_Z$ = Tmin algoritmasının bulduğu çözüm değeri,

$\hat{f}_Z$ = İyileştirilmiş çözüm değeridir.

Görüldüğü gibi $\overline{\overline{R}}_Z$ tümleyen probleminin $\overline{\overline{f}}_Z$ yaklaşık çözümü bulunarak R_Z probleminin f_Z yaklaşık çözümünü iyileştiren bir $\hat{f}_Z$ çözümünün bulunması amaçlanmıştır.

KAYNAKLAR DİZİNİ

Aarts E. and Lenstra J. K., 1997, Local Search in Combinatorial Optimization, John Wiley&Sons, England, 512p.

Ausiello G., Cerscenzi P., Kann V., Marchetti-Spaccamela A. and Protasi M., 1999, Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties, Springer, Berlin, 524p.

Bakır M. A. ve Altunkaynak B., T., 2003, Tamsayılı Programlama, Nobel Yayınları, Ankara, 620s.

Cormen T., Leiserson C. E., Rivest R. L. and Stein C., 2001, Inroduction to Algorithms, MIT Press, USA, 1202p.

Dantzig G. B., 1963, Linear Programming and Extensions, Princeton University Press, New Jersey, 648p.

Fisher M. L., 1980, Worst-case Analysis of Heuristic Algorithms, *Management Science*, 26(1):1-17.

Garey M. R. and Johnson D. S., 1979, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 338p.

Güntzer M. M. and Jungnickel D., 2000, Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem, *Operations Research Letters*, 26:55-66.

Hochbaum D. S., 1997, Approximation Algorithms for NP-Hard Problems, PWS Publishing, Boston, 596p.

KAYNAKLAR DİZİNİ (devam)

Ibarra O. H and Kim C. E., 1975, Fast approximation algorithms for the knapsack and sum of subset problems, *Journal of ACM*, 22(4):463-468.

Kara İ., 1991, Doğrusal Programlama, Bilim Teknik Yayınevi, Eskişehir, 270s.

Kellerer H., Pferschy U. and Pisinger D., 2004, Knapsack Problems, Springer, Berlin, 546p.

Martello S. and Toth P., 1990, Knapsack Problems, John Wiley&Sons, England, 296p.

Nikitin A. I. ve Nuriyev U. G., 1983, On a method of the solution of the knapsack problem, *Kibernetika*, 2:108-110.

Nuriyev U. G., 1986, On the solution of the knapsack problem with guaranteed estimate, *Problems of Computing Mathematics and Theoretical Cybernetics,* 66-70.

Nuriyev U. G. ve Dündar P., 1998, Knapsack probleminin greedy algoritma ile bulunmuş çözümünün incelenmesi, *Atatürk Üniversitesi 40. Kuruluş Yılı Matematik Sempozyumu Bildirileri,* Erzurum, 179-185.

Nuriyev U. G., Çalışkan E. ve Sadıgova H. G., 2004, Knapsack minimizasyon probleminin bir greedy algoritması ile çözümü, *Dördüncü İstatistik Günleri Sempozyumu Bildirileri*, İzmir, 387-392.

Papadimitriou C. H. And Steiglitz K., 1982, Combinatorial Optimization: Algorithms and Complexity, Prentice Hall, New Jersey, 496p.

KAYNAKLAR DİZİNİ (devam)

Sahni S., 1975, Approximate algorithms for the 0/1 knapsack problems, *Journal of ACM*, 1:115-124.

Vazirani V. V., 2001, Approximation Algorithms, Springer, Berlin, 380p.

Winston W. L., 1991, Operations Research Application and Algorithms, PWS Publishing, Boston, 1262p.

ÖZGEÇMİŞ

Aslı GÜLER, 29.10.1983 tarihinde İzmir'de doğdu. İlkokulu Turgut Reis İlkokulu'nda tamamlayarak Alsancak Orta Okulu'na devam etti. Lise öğrenimini Şirinyer Lisesi'nde (YDAL) tamamladıktan sonra 2001 yılında Eskişehir Anadolu Üniversitesi Fen Fakültesi Matematik Bölümü'nü kazandı. Hazırlık ve birinci sınıf öğrenimini burada tamamlayarak 2003 yılında Ege Üniversitesi Fen Fakültesi Matematik Bölümü'ne yatay geçiş yaptı. 2006 yılında Matematik Bölümü Bilgisayar Opsiyonu'ndan mezun olarak aynı yıl Ege Üniversitesi Fen Bilimleri Enstitüsü Matematik Anabilim Dalı'nda yüksek lisans öğrenimine başladı.