Karaelmas Fen ve Mühendislik Dergisi
Dergi web sayfası: http://fbd.beun.edu.tr
DOI: 10.7212%2Fzkufbd.v8i1.830

Araştırma Makalesi
Geliş tarihi / Received : 04.01.2017
Kabul tarihi / Accepted : 27.11.2017

Tam Sayı Programlamada Açgözlü ve Sezgisel Aramalar ile 0/1 Sırt Çantası Problemine Yeni Bir Bakış

A New Approach to 0/1 Knapsack Problem with Greedy and Heuristic Searches in Integer Programming

Faruk Bulut[1*], İbrahim Furkan Ince[2]

[1]Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği Bölümü, İstanbul Rumeli Üniversitesi, İstanbul, Türkiye
[2]Department of Electrical and Electronics Engineering, Kyungsung University, Busan – South Korea

Öz

Tam sayılı programlama, bir çeşit optimize edilmiş Lineer Programlama olarak adlandırılan doğrusal programlama yöntemidir. Amaç doğrusal programlamada meydana gelebilecek gerçekçi olmayan sonuçları ortadan kaldırmaktır. Tam sayılı Programlama birçok mühendislik alanına uygulanmaktadır. Bu çalışmada, Tam sayılı Programlama yöntemine sezgisel ve açgözlü bir yaklaşım eklenerek çok bilinen ve birçok mühendislik uygulamasının kaynağı olan "sırt çantası" problemine uygulanmıştır. Ayrıca Yığın veri yapısı kullanılarak alan karmaşıklığı en aza indirilmiş ve daha hızlı sonuçlara ulaşılmıştır. Yapılan deneysel uygulamalarda önerilen yöntemin özyinelemeli, kesmeli ve optimize edilmiş yöntemlere göre daha az hesaplama zamanı içerisinde sonuç verdiği gözlemlenmiştir.

Anahtar Kelimeler: Dal & Sınır algoritması, Sezgisel arama, Yığın

Abstract

Integer programming is a type of optimized Linear Programming method. The goal is to get rid of unrealistic results that might occur in the linear programming process. Integer Programming is applied to many engineering fields. In this study, a new heuristic and greedy approach to the conventional Integer Programming method is proposed to the well-known knapsack problem, which is accepted as the source of many engineering problems. Heap data structure, additionally, is used in order to decrease both space complexity and implementation time. In the experimental applications, it is observed that the proposed method yields less computational time than the recursive, pruned, and optimized methods.

Keywords: Branch & Bound Algorithm, Heuristic search, Heap

1. Giriş

Bilindiği üzere sırt çantası problemi, genellikle hırsızlıkta hırsızın mal kaçırma senaryosunda anlatılan bir sorudur. Amaç, belirli bir kapasiteye sahip sırt çantasına yükte hafif, pahada ağır eşyaların tam kapasitede ya da kapasiteyi aşmayacak şekilde yerleştirilerek en kazançlı kombinasyonun oluşturulmasıdır. 0/1 durumu ise çantaya alınabilecek belirli bir eşyanın ya alınması ya da alınmaması anlamına gelir. Kesirli sırt çantası uygulamalarında ise öğeler parçalara ayrılabilmektedir. Bilişim dünyasını yakından ilgilendiren bu konu birçok temel mühendislik problemlerinin temelini oluşturmaktadır. Sırt çantası probleminin farklı türlerinin mühendislik alnında uygulamaları vardır. Yani birçok gerçek dünya sorununun temel felsefesini içermektedir.

Lineer yani Doğrusal Programlama ise (*Linear Programming*), bir fonksiyonun yardımıyla verilen kısıtlamalara göre *global minimum* veya *global maksimumu* bulmak için kullanılan bir yöntemdir. *Global maksimum* ya da *minimum* hesaplamada elde edilmesi gereken sonuçtur. İstenilen sonuç olarak elde edilen çıktının küsuratlı ya da tam sayı olması önemli değildir. Sonucun net olması yeterlidir. Fakat bazı doğrusal programlama uygulamalarında elde edilen sonuçların tam sayı çıkmaması problemin gerçek hayatta uyarlanmasını engellemektedir (Vanderbei 2015). Örneğin bir üretim merkezinde belirli bir zaman diliminde kişi başı yaptırılacak üretim miktarı tam sayılarla ifade edilmelidir. Burada küsuratlı sayılar planlamayı bozmakta ve gerçeği yansıtmamaktadır. Bu gibi özel durumların çözülebilmesi için tam sayı programlama (*Integer Programming*)

*Sorumlu yazarın e-posta adresi: faruk.bulut@rumeli.edu.tr

Faruk Bulut orcid.org/0000-0003-1646-0266
İbrahim Furkan Ince orcid.org/0000-0003-1570-875X

yöntemleri önerilmiştir. Tam sayılı programlamada yaygın olarak kullanılan bazı algoritmalar şunlardır (Taha 2014):

1. Gevşetme - Sınır Algoritması (*Relaxations & Bound Algorithm*)
2. Dal - Sınır Algoritması (*Branch & Bound Algorithm*)
3. Kesme Düzlemi Yöntemi (*Cutting-Planes Method*)
4. Böl-Yönet Algoritması (*Divide & Conquer Algorithm*)
5. Dinamik Programlama (*Dynamic Programming*)
6. Sütun Oluşturma Yaklaşımı (*Column Generations Approach)*
7. Sezgisel Programlama (*Heuristic Programming*)
8. Karar Ağaçlarının Budanması (*Pruning Decision Trees*)

Tam sayılı programlama, doğrusal programlama yönteminin yetersiz kaldığı durumlarda kullanılan bir yöntemdir. Daha geniş bir açıklama ile *matematiksel optimizasyon* yöntemi ile sonucun tam sayılı çıkmasına odaklı bir programlama yöntemidir. Bu durumda probleme özgü girdi değişkenlerinin de tam sayı olması beklenmektedir (Bosch ve Michael 2014).

Doğrusal programlama sürekli fonksiyonlar ile ilgili olurken tam sayılı programlama kesikli fonksiyonlarla ilgilidir. Doğrusal Programlama problemlerinin çözüm değerleri çoğu kez tam sayılı olmayan kesirli pozitif sayılardır. Güncel hayatta çözüm değerlerinin tamsayı olmasını gerektiren birçok problem bulunmaktadır. Beyaz eşya, otomobil ve makine üretimi, bunların depolanması ve sevkiyatı, personel gereksinimi gibi problemler tam sayılı doğrusal programlamaya birer örnek teşkil ederler. Özetle Tam Sayılı Programlama, çıktı değerlerinin ve karar değişkenlerinin tam sayılı olması durumlarında, mühendislik problemlerin modellenmesi ve çözümüne ilişkin kavram ve tekniklerin geliştirilmesidir denilebilir (Conforti vd. 2014). Tam sayı programlama, karmaşıklık analizinde (*Complexity Analysis*) çözüm yöntemine bağlı olarak NP-Hard (*Non Polynomial - Hard*) grubuna dâhil edilmektedir. NP-Hard problemlerin karmaşıklığı polinomsal olmayan bir yapıdadır yani tam sayı olan girdi değerleri arttıkça problemin çözümü üstsel olarak da artmaktadır. Örneğin Büyük O notasyonuna göre ve NP-Hard grubuna girmektedir (Kellerer vd. 2004). Sırt çantası problemi de lineer yöntemlerle çözüldüğünde zaman karmaşıklığı olmaktadır.

Bu çalışmada bilinen tam sayı programlama yöntemi olarak kabul edilen Dal-Sınır algoritmasının daha iyi ve hızlı sonuçlar verebilmesi için Açgözlü arama metodu içeren sezgisel bir fonksiyonun eklenmesi amaçlanmıştır. Ayrıca hesaplama zamanının düşürülmesi için en uygun veri yapısı ilave edilerek eniyileme çalışması yapılmıştır. Çalışmamızın bundan sonraki bölümlerinde beş ana bölüm daha içermektedir. Takip eden ikinci bölümde konu ile ilgili yapılan akademik çalışmalara, üçüncü bölümde önerilen sezgisel aramalı Dal-Sınır yöntemine, dördüncü bölümde örnek bir senaryo üzerinden algoritmanın uygulanmasına, test edilmesine ve diğer benzer yöntemlerle karşılaştırılmasına yer verilmiştir. Son iki bölümde ise değerlendirmelere ve uygulamaların kaynak kodları ile test dosyalarına nasıl ulaşılabileceğini anlatan erişebilirlik bölümlerine yer verilmiştir.

2. Literatür Taraması

Sırt çantası probleminin çözümü için çok sayıda algoritma ve çözüm yaklaşımı önerilmiştir. Çözüm tekniklerini doğrusal olmayan (*non-linear*) ve doğrusal (*linear*) yöntemler olmak üzere iki gruba ayrılmaktadır. Çünkü sırt çantasına alınabilecek öğe sayısı yaklaşık olarak 50'yi geçtiği durumlarda doğrusal olmayan yani optimizasyon yöntemleri en iyi sonuca yakın çözümleri daha hızlı verebilmektedir. Fakat öğe sayısının küçük olması durumlarında ise doğrusal yöntemler yani tam sayı programlama teknikleri hem daha hızlı hem de daha doğru çıktılar üretebilmektedir. Belirtmek isteriz ki sırt çantası problemleri, tam sayılı programlama problemleri içinde üzerinde en çok çalışılmış olanlardan birisidir. Yapılan literatür taramasını iki farklı sınıfta incelemekte fayda vardır. Birincisi doğrusal olmayan yöntemler, ikincisi ise doğrusal olan yöntemlerdir.

2.1. Doğrusal Olmayan Yöntemler

Bu alanda yapılan çalışmalar genellikle sezgisel ve meta sezgisel arama mantığı içermektedir. En iyileştirme olarak Türkçeye çevrilen optimizasyon yöntemleri birçok çalışmada bu probleme uyarlanmıştır. Literatürde önerilen oldukça fazla optimizasyon yöntemi vardır. Bazıları birbirlerine oldukça fazla benzemektedir. Bu yöntemlerin sırt çantasına nasıl uyarlanacağı ve nasıl bir çözüm önerisi sunulacağı çalışmayı yapan kişinin önerdiği modellemeye bağlıdır.

Ayrıca bilinmelidir ki optimizasyon yöntemlerinin bu tür problemler karşısında bazı avantajları ve dezavantajları vardır. Dezavantajlar şu şekilde sıralanabilir:

1. Her ne kadar optimizasyon teknikleri diğer doğrusal yöntemlerle karşılaştırıldığında genel olarak başarılı sonuçlar verse de bazı mühendislik problemlerine uyarlanması aşamasında inşa edilen modelin yetersiz oluşu nedeniyle elde edilen sonuçlar hiç de beklenen düzeyde olmamaktadır.

2. Öncelikle çözüme ulaşma süresinin yani hesaplama zamanının ne kadar süreceği önceden kestirilemeyebilir. Standart bir donanıma sahip bilgisayar ile çok kısa bir zamanda sonuca ulaşılabileceği gibi hızlı ve çok çekirdekli işlemcilere sahip bir ortamda bile basit bir problem için dakikalar hatta saatler harcanabilmektedir.
3. Metotların en iyi performansı verebilmesi için en uygun parametrelerin belirlenmesi gerekebilir. Bu da kullanıcın bir problem üzerinde deneme yanılma yöntemi ile en iyi parametreleri seçmesi ile gerçekleşebilir. Bu durum esasen istenmeyen bir durumdur. Parametreye bağlı başarı gösteren bir sistemin inşa edilmeye çalışılması mühendislik uygulamalarında uygulanabilirlik ve basitlik ilkesini zedelemektedir.
4. Elde edilen sonuç her zaman mümkün olan en iyi sonuç değildir. Alternatif çözümler içerisinde en iyiye yakın her hangi bir çözüm çıktı olarak karşımıza çıkabilir. Her ne kadar bazı mühendislik uygulamalarında doğru olan her hangi bir sonuç elde edilmek istense de çoğu durumda zaman kısıtlamasına bağlı kalmaksızın en iyi sonuca ulaşılmak istenebilir.
5. Parametrelerin çok olması ise karmaşayı artırdığı gibi en iyi sonucu veren parametre kombinasyonun oluşturulması ayrı bir handikap olarak karşımıza çıkmaktadır.
6. Uygulama aşamasında bulunan süreçler çoğu zaman analiz edilemez ve incelenemez. İç kısımda olan olayların incelenmesi zordur. Örneğin Çok Katmanlı Algılayıcılar ile oluşturulan bir Yapay Sinir Ağı mekanizmasında katmanlar içindeki algılayıcılar analiz edilemez. Genetik Algoritmalarda gen havuzunda bulunan kromozomlar ile eşleştirilmesindeki başarı, rastgeleliliğe doğrudan bağlıdır.

Avantajlar ise şu şekilde sıralanabilir:

1. Problemde öğe sayısının fazla olması durumunda bile hesaplama zamanı çok kısa bir sürede tamamlanabilir.
2. Yine aynı şekilde normalden fazla öğenin bulunduğu girdi listesinde mümkün olan her hangi bir sonuca ulaşılabilir.
3. Bu sayede bilgisayara ait sistem kaynaklarının daha az kullanılması söz konusudur.

Sır Çantası problemi optimizasyon olarak bilinen eniyileştirme yöntemleri ile de çözülmeye çalışılmıştır. Bu alanda birçok çalışma incelenmiş ve sadece birkaç tanesi makalede anlatıma uygun bulunmuştur. Doğrusal olmayan bu yöntem grubunda meta sezgisel bir yaklaşımla Karınca Kolonisi algoritması tek ve çok boyutlu problemlere uyarlanmıştır (Kong vd. 2008). Kong ve arkadaşları tarafından yapılan bu çalışmada klasik ve bilinen Karınca Kolonisi Optimasyon tekniğinden farklı olarak eniyileştirme sürecinin her aşamasında özel bir operatör ile iyi olmayan çözüm alternatiflerini tamir etmişlerdir. Yaptıkları deneysel uygulamalarda önerdikleri yöntemin bilinen yöntemden daha iyi olduğunu kanıtlanmışlardır.

Khuri vd. (1994) sırt çantası problemine Genetik Algoritmalar tekniği ile alternatif çözümlerden birini en hızlı şekilde bulmayı amaçlayan bir çözüm önerisi getirmiştir. Oluşturulan bir gen havuzunda kromozomlar yani çözüm bireyleri çapraz eşlemelerle yeni kromozomların basit bir uygunluk fonksiyonu ile türetilmesine çalışılmıştır ve başarılı sonuçlar elde edilmiştir.

Zou vd. (2011) ise NGHS (*Novel Global Harmony Search*) yöntemi ile Genetik Algoritmaların benzeri bir çalışma ortaya koymuşlardır. Optimize edilen bu yöntemde 0/1 sırt çantası için daha az hesaplama süresinde (*computational time*) sonuca ulaşabilmesi için iki farklı yöntem kullanmıştır. Birincisi pozisyonun güncellenmesi, ikincisi genetik mutasyonun daha küçük bir olasılık ile yapılması. NGHS'de Harmoni arama algoritması sayesinde lokal en iyi sonuca takılıp kalmamakta global maksimuma daha hızlı ulaşabilmektedir.

Ohlsson vd. (2008), sırt çantası problemine yapay sinir ağları kullanarak bir çözüm getirmeye çalışmışlardır. Yaklaşık ve olası bir sonuç bulmayı hedefleyen çalışmaları optimizasyon kategorisinde kalmıştır. Önerdikleri çözüm modelini 103 öğeli sırt çantası uygulamasına kadar deneyebilmişler. Bulunan çözümlerin tam kapsamlı arama ile elde edilebilen olası en iyi çözüme ortalama olarak %95 oranında benzerlik gösterdiği de vurgulanmıştır. Ayrıca çalışmalarını *Simulated Annealing* ve birkaç doğrusal model ile karşılaştırmışlar ve daha başarılı sonuçlar elde ettiklerini vurgulamışlardır. Her ne kadar olası gerçek ve en iyi çözümden ortalama %5 kadar geride sonuçlar elde etseler de fazla öğeli sırt çantası problemlerinde her hangi bir doğru sonucun elde edildiğini ön plana çıkarmaya çalışmışlardır.

2.2. Doğrusal Yöntemler

Bu problem türüne uygulanabilecek doğrusal olan yöntemler, olmayanlara göre sınırlı sayıdadır. Özyineleme, Açgözlü arama, Dal-Sınır algoritması, Backtracking, Dinamik programlama ve Lagrangian Gevşetme Metodu çözüm modelleri kategorisinde önerilmektedir (Taha 2014,

Martello vd. 2000, Fisher 2004). Bu tür yöntemler öğe sayısının düşük olduğu durumlarda doğru ve net sonuçlar verdiği için işlevsellik kazanmaktadır.

Hochbaum (1995) bir çalışmasında, doğrusal yöntemlerle bu problemin nasıl çözülebileceği konusunda araştırma yapmıştır ve yöntemler arasındaki zaman karmaşıklıklarını karşılaştırmıştır.

Yine Chekuri vd. (2005), polinomsal bir zaman karmaşıklığına sahip yöntemi çok boyutlu sırt çantası için önermişlerdir. Belirli sayıdaki öğe ile birden çok sırt çantası olması durumunda bu problemin Genelleştirilmiş Atama Problemi (*Generalized Assignment Problem*) olduğunu belirtmiştir ve çözüm önerileri sunmuştur.

Puchinger vd. (2010) bir çalışmalarında çok boyutlu sırt çantası problemlerinin yapısı ve algoritmaları konusunda hem teorik hem de uygulamalı bir çalışma yapmışlardır. Sezgisel yaklaşımların, lineer tabanlı modellerin, işbirlikçi yöntemlerin incelendiği çalışmada, yöntemler karşılaştırılmıştır.

Lineer programlama teknikerinin bu problem türünde kullanılmasının bazı avantajları ve dezavantajları vardır. Öncelikle avantajlar şu şekilde sıralanabilir:

1. Elde edilen sonuç her zaman için olası çözümler kümesinde bulunanların en iyidir. Bu durum, mühendislik ve bilimsel uygulamalarda çoğu zaman beklenen bir durumdur.
2. Doğrusal yöntemlerin bu tür problemlere uyarlanması yani kodlarının yazılması oldukça kolaydır.
3. Parametre gereksinimi yok gibidir. Doğal olarak sistem parametreden bağımsız olduğu için farklı sonuçlar verme durumu söz konusu değildir.
4. Doğrusal yöntemlerin uygulama süreci içerisinde bulunan bütün aşamaları oldukça şeffaftır. Her bir adım incelenebilir ve gözlemlenebilir.

Tam sayı programlamanın bu alana uygulanmasının olumsuz yönü öğe sayısının normalden fazla olması durumunda geçerlidir. Çoğu durumda öğe sayısının artması ile gereken işlem sayısı asimtotik olarak da artmaktadır. Bu da hesaplama zamanının uzaması ve sistem kaynaklarının gereğinden fazla kullanılması anlamına gelir. Eğer tam doğru veya en iyi sonuç çıktı olarak beklenmiyorsa bu problem türleri optimizasyon teknikleri ile çözülmelidir. NP-Hard sınıfında olan problemler için doğrusal olmayan yöntemlerin gerçek hayatta tercih edildiğini vurgulamakta fayda vardır.

3. Gereç ve Yöntem

Dal-Sınır algoritması genel olarak *minimizasyon* ya da *maksimizasyon* problemlerine uygulanır. Kombinasyonel veri analizinde (*combinatorial data analysis*) yapılan uygulamalar oldukça geniştir. Bu konu ile ilgili Brusco vd. (2006) tarafından hazırlanan kitapta uygulamaların genel olarak şu çerçeve içerisinde olduğu görülmektedir:

1. Sırt çantası problemi (*maksimizasyon problemi*)
2. Gezgin satıcı (*Travelling Salesman Problem*) problemi (*minimizasyon problemi*)
3. İş dağılımı ve atama (*Job Assignment and Workflow*) problemi (*minimizasyon problemi*)

Tam sayılı programlama problemlerini çözmek için, literatürde bir dizi çözüm yöntemleri geliştirilmiştir. Geliştirilen bu çözüm yaklaşımlarının birinin diğerine üstünlüğü problemin yapısına göre değişebilmektedir. Tam sayılı programlama metotları kesme yöntemleri ve arama yöntemleri olarak genelde iki ayrı kategoriye ayrılmaktadır. Kesme yöntemlerinde *Gomory* kesme düzlemi algoritması; arama yöntemlerinde ise Dal-Sınır algoritması sıklıkla kullanılır (Cornuéjols 2007).

Dal-Sınır algoritması tamsayı ve karışık tamsayı programlama problemlerinin çözümünde sık kullanılan bir metottur. Bu algoritma özyineleme (*recursion*) tekniğinin hızlandırılmış bir çeşididir. Normalde özyinelemeli fonksiyonlar olası tüm durumları deneyerek sonuca ulaşmaya çalışırlar. Diğer bir deyişle çözüm kümesine ulaşabilmek için çözümü oluşturabilecek veya oluşturamayacak tüm durumları teker teker ve çoğu zaman tekrar tekrar denerler. Bu durum hesaplama süresini artıran ve istenilmeyen bir faktördür (Brucker vd. 1994). *N* elemanlı bir sırt çantası probleminde 1'den başlayarak N'e kadar 1'li, 2'li, … ,(N-1)'li ve N'li kombinasyonların tamamı incelenerek tam kapsamlı bir arama (*exhaustive search*) yapılır (Zhang 1999). Toplam arama işlemi sayısı *N* elemanlı bir kümenin alt kümeleri sayısı kadardır ve bu açıklama aşağıdaki Eşitlik 1'de gösterilmektedir.

$$C(N,1) + C(N,2) + \ldots + C(N,N-1) + C(N,N) = 2^N - 1 \qquad (1)$$

Burada $2^N - 1$ adet durum tek tek kontrol edilir ve sırt çantasının kapasitesini aşmayan en iyi çözüm sonuç olarak kabul edilir. Bu çözüm özyinelemeli bir fonksiyon ile yazılıp çalıştırılabilir. Özyinelemeli programlama uygulamalarında genellikle hesaplanan bir sonuç ilerleyen aşamalarda tekrar tekrar hesaplanabilmektedir. Bu da hesaplama zamanını artıran bir faktördür. Burada zaman karmaşıklığı doğal

olarak polinomsal olmayan bir sonuç çıkmaktadır: $O(2^N)$. Bu durum N değerinin büyük olduğu durumlarda istenmeyen büyük bir zaman karmaşıklığına sahiptir (Pisinger 2005).

Sırt çantasına ait ağırlık kapasitenin V, toplam eşya sayısının N, i. eşyanın ağırlığının , değerinin de olduğunu düşünelim. ise, i. elemandan kaçar adet olduğunu gösteren tam sayı, X_i ise i. eşyadan sırt çantasına alınacak adet sayısını belirtiyor olsun. Bu durumda koşullar ve amaç eşitlik 2 ve 3'de verilmektedir.

Ön koşullar 2 numaralı eşitlikte verildiği gibidir.

$$\sum_{i=1}^{N} V_i X_i \leq V \text{ ve } 1 \leq Q_i \leq \infty \quad (2)$$

Amaç 3 numaralı eşitlikte gösterildiği gibi en yüksek değeri elde etmektir.

$$\sum_{i=1}^{N} B_i X_i \quad (3)$$

3.1. A-Star Sezgisel Fonksiyonu

Belirtilmesi gereken diğer bir nokta ise, Dal-Sınır algoritmasında alt fonksiyonlar özyinelemeli olarak dallanırken sınır değerler belirlenirken sezgisel yöntemlerden (*Heuristic Methods*) faydalanılması gerektiğidir. Burada sınır değeri hesaplanırken *A-Star* sezgisel yaklaşımı kullanılır (Li vd. 2008). A* olarak da gösterilen bu yöntemin genel çalışma mantığı Eşitlik 4'deki fonksiyonda verilmektedir (Delling vd. 2009).

$$f(x) = g(x) + h(x) \quad (4)$$

Burada $g(x)$, o aşamaya kadar yapılan somut giderleri, masrafları belirten fonksiyondur. $h(x)$, bulunulan aşamadan sonra yapılması gereken tahmini masrafları belirleyen sezgisel (*heuristic*) fonksiyondur. $f(x)$ ise tahmini toplam maliyet fonksiyonudur. Burada $h(x)$ fonksiyonun nasıl tasarlanacağı, neye göre hesaplanacağı, hangi parametrelere dikkat edileceği önemli bir konudur ve tamamıyla tasarımcının kontrolüne bırakılmıştır. Unutulmamalıdır ki belirlenen $h(x)$ modeli, önerilen sistemin toplam performansını etkileyen en önemli faktördür. Burada $h(x)$ için en önemli parametre sınır (*bound*) değerleridir.

Dal-Sınır algoritması, üst sınır (*upper bound, UB*) veya alt sınır (*lower bound, LB*) kısıtlarıyla özyinelemeli fonksiyonunun gereksiz bazı hesaplamalar yapmasının önüne geçerek işlem hızını artırmayı hedefler. Dal-Sınır algoritması *minimizasyon* ya da *maksimizasyon* problemlerine uygulanır. *Minimizasyon* işlemlerinde bir alt sınır; *maksimizasyon* işlemlerinde de bir üst sınır değeri vardır. Üst sınır ve alt sınır değerleri daha önceden çeşitli yaklaşımlar kullanılarak programcı tarafından hesaplanmıştır. Üst sınır kontrolü *maksimizasyon* problemlerinde; alt sınır kontrolü de *minimizasyon* problemlerinde kullanılan bir kesme yöntemidir.

3.2. Açgözlü ve Sezgisel Yaklaşımlı Dal-Sınır Yöntemi

Açgözlü arama tekniğinin mantığı oldukça basittir. İlgili aşamada alternatif seçeneklerden "en iyisi" tercih edilir. "En iyisi" kavramı bazen en büyük değer olabileceği gibi bazen de en küçük değer olabilir. Açgözlü yöntemi her zaman doğru ve optimum sonucu veremeyebilir. Hatta bazı uygulamalarda yanlış sonuca ulaşılmasına neden olur (Mitzenmacher vd. 2017). Fakat burada Açgözlü yöntemi ile sadece sırt çantasına alınacak eşyaların değerlikleri büyükten küçüğe doğru sıralandığı için çantanın öncelikli olarak değerli eşyalar ile doldurulması amaçlanmıştır.

Açgözlü ve Sezgisel Aramalı Dal-Sınır yöntemde çantaya alınacak öğeler birim değerine göre tersten sıralanır. Birim değeri malın değerinin hacmine bölünmesi ile bulunur. En değerli obje çantaya ilk olarak alınacağı için. Maksimizasyon işlemi için bunun öncelikle yapılması gerekir. Sıralama algoritması olarak $O(N \log N)$ karmaşıklığına sahip Çabuk sıralama (*quick sort*) kullanılmıştır. Eğer bu sıralama yapılmayıp gelişigüzel öğelerin çantaya alınması ve farklı kombinasyonların denenmesi durumu tercih edilseydi yine aynı sonuca ulaşılacaktı. Bu durumda sadece hesaplama zamanını uzatacaktı. Açgözlü arama yapılmadan gelişigüzel eklemeler ile de Dal-Sınır algoritmasında aynı sonuçlara ulaşılabilmektedir.

Özyineleme fonksiyonu alt fonksiyonlara dallanmadan yani *branch* işlemi yapmadan önce bu sınır değeri kontrol edilir. Sınır değerler aşılıyorsa o aşamadaki özyinelemeli fonksiyon kapatılır ve kendinden sonraki alt fonksiyonlara da gidilmesi engellenir; aşılmıyorsa özyinelemeli fonksiyonu değişik parametreler ile kendini tekrar çağırarak işleme devam eder. Amaç her olası doğru çözümün sınır değerleriyle her aşamada kontrol edilerek en doğru çözümün hızlı bir şekilde bulunmasıdır.

Burada özetle özyinelemeli fonksiyon kendini tekrar çağırmadan yani alt fonksiyonlara dallanmadan önce alt fonksiyonlarda elde edilecek sonucun üst sınırı aşıp aşmadığı ya da alt sınırın altına inip inmediğini kontrol eder ve buna bağlı olarak fonksiyon kendini tekrar çağırır ya da çağırmaz. Kısaca üst sınır ve alt sınırlar yardımıyla makul çözümü vermeyen alt çözüm kümeleri elenir. Optimum çözüm bulunana kadar alt kümeler oluşturulmaya ve oluştururken elenmelere devam edilerek sonuca ulaşılmaya çalışılır. Bu

Çizelge 1. Zaman ve alan karmaşıklığı

Yöntem	Zaman Karmaşıklığı	Alan Karmaşıklığı
Özyinelemeli Yöntemi	$O(2^N)$	$O(2^N)$
Dal-Sınır Yöntemi	$O(2^N)$'den daha az	$O(N)$
Sezgisel Dal-Sınır Yöntemi	$O(N\log N + N)$	$O(N)$
Yığın Veri Yapılı Açgözlü Sezgisel Dal-Sınır Yöntemi	$O(N\log N + N + N\log N)$	$O(N + N)$

Çizelge 2. Örnek senaryo.

Mal	Ağırlığı (kg)	Değeri (TL)
A	8	56
B	7	63
C	10	100
D	4	12

sayede gereksiz alt fonksiyonların çağırılmasını engelleyerek yöntemin hem zaman karmaşıklığının azaltılması hem de hesaplama süresinin düşürülmesi sağlanmış olur.

3.3. Yığın Veri Yapısı (Heap Data Structure)

Algoritmaların kodlanması ve uygulaması sürecinde en iyi performans için en uygun veri yapısına ihtiyaç duyulur. Probleme uygun olarak seçilmiş bir veri yapısı hem alan karmaşıklığını (*space complexity*) hem de hesaplama zamanını azaltır. Bu uygulamada Yığın (*heap*) veri yapısı tercih edilmiştir. Heap veri yapısı iki şekilde kullanılabilir. Birincisi Min-Heap, ikincisi ise Max-Heap. Bu çalışmada Max-Heap veri yapısı tercih edilmiştir. Her iki yöntemde de veri yapısına eleman eklemenin zaman karmaşıklığı $O(\log N)$'dir. Eleman alma $O(1)$, yapıyı tekrar güncelleme $O(\log N)$'dir. Doğal olarak N elemanlı bir yapıda bu veri yapısının kullanımı $O(N\log N)$'lik bir karmaşıklığı ortaya çıkarmaktadır (Cormen vd. 2009).

Önerilen yöntemde özyinelemeli Dal-Sınır algoritması kendini alt fonksiyonlara en iyi çözümü bulmak için tekrar tekrar göndermektedir. Küçük öğe sayısına sahip uygulamalarda bile binlerce alt fonksiyon oluşabilmektedir. Örneğin N değerinin 20 olduğu durumda 2^{20} yaklaşık olarak 1 milyon çıkacaktır. Kısaca ana fonksiyonun kendini tekrar tekrar çağırmasıyla 1 milyon durum tek tek değerlendirilecektir. Gereksiz işlem tekrarlarını engellemek olası çözüm alternatifleri içerisinde Açgözlü arama ile en iyi olanı tercih edilecektir. Peki, bahsi geçen bu olası çözüm alternatifleri içerisinde her defasında tekrar tekrar arama mı yapılacaktır? Elbette ki hayır. Bu alternatif çözümler Max-Heap yapısı içerisinde tutulacağı için en uygun çözüm kök'te (*root*) duran olacaktır. Burada "en uygun çözüm" parametresi nasıl belirlenmelidir. Bu önemli bir konudur ve sistemin doğru çaşılmasını sağlayan önemli bir parametredir. "En uygun çözüm", 4. Bölümde detaylı olarak anlatılan Dal-Sınır yönteminin *Upper Bound* (UB) değeridir.

3.4. Toplam Zaman ve Alan Karmaşıklığı

Sezgisel fonksiyonlu, Açgözlü aramalı ve Yığın veri yapısı entegre edilmiş bu yöntemin zaman karmaşıklığı toplam $O(N\log N + N + N\log N)$'dir. Bu notasyon içerisinde toplanan 3 karmaşıklık değerleri sırasıyla Çabuk sıramala yöntemine, sırasıyla ele alınan N adet eşyaya ve Yığın veri yapısına aittir. Gerekli sadeleştirmeler yapıldığında ise yine $O(N\log N)$ karmaşıklığı karşımıza çıkar.

Alan karmaşıklığı ise $O(N + N)$'dir. İlk N değeri sıralanan eşya sayısına, diğeri ise Yığın veri yapısına aittir.

Özyinelemeli Yöntemi, Dal-Sınır Yöntemi, Sezgisel Dal-Sınır Yöntemi ve önerilen Yığın Veri Yapılı Açgözlü Sezgisel Dal-Sınır Yöntemi ile ilgili zaman ve alan karmaşıklıkları Çizelge 1'de görüldüğü gibidir.

4. Bulgular ve Karşılaştırma

Sezgisel yaklaşımlı ve Açgözlü arama yöntemli Dal-Sınır algoritması, bir *maksimizasyon problemi* olan sırt çantası problemine uygulanmaktadır. Bilindiği üzere bu soru yangın esnasında, *kg* cinsinden V kapasitesindeki bir çuvala yükte hafif, pahada ağır şeylerin konulup bir an önce yangın mahallinden dışarı çıkarılmasıdır. Bu soruda hedeflenen şey, fazla değerlikteki (*maksimizasyon problemi*) malların yangından kurtarılmasıdır. N adet eşyanın isimleri, ağırlıkları ve TL cinsinden değerleri Çizelge 2'de verildiği gibi olsun. Mallardan birer tane vardır ve çuvalın kapasitesi V=16 kg şeklinde kabul edilsin.

Bu soruyu en basit şekliyle 4'ün 1'li, 2'li, 3'lü ve 4'lü kombinasyonlarını tek tek deneyip en iyi çözümü içlerinden seçerek çözebiliriz. Yani N elemanlı bir kümenin alt kümelerinin sayısı, C(4,1) + C(4,2) + C(4,3) + C(4,4) = $2^N - 1$ olur. Burada $2^N - 1$ adet durum tek tek kontrol edilir ve çuvalın V kapasitesini aşmayan en iyi çözüm sonuç olarak kabul edilir. Bu çözüm özyinelemeli bir fonksiyon

Çizelge 3. Malların değerlerine göre sıralanışı.

Mal	Ağırlığı (kg)	Değeri (TL)	Birim Değeri (TL/kg)
C	10	100	10
B	7	63	9
A	8	56	7
D	4	12	3

ile yazılıp çalıştırılabilir ve zaman karmaşıklığı $O(2^N)$ olur [11]. Bu çözüm yöntemi hesaplama süresi bakımından hızlı çalışmaktadır. Fakat zaman karmaşıklığı, *Upper Bound* (UB) kesmelerinin programın koşturulması esnasında işletilip işletilmemesi garanti edilmediği için $O(2^N)$ olarak kalacaktır.

Burada önerilen ve kullanılacak olan *A-Star* sezgisel $f(x)$ fonksiyonunu $f(x) = g(x) + h(x)$ şeklindedir. $g(x)$, çuvalda somut olarak var olan malların gerçek ağırlığı ve fiyatıdır. $h(x)$, çuvala alınacak olanların birim fiyatı ile poşetin kapasitesinin çarpımıdır. $f(x)$ ise UB değeri, alt fonksiyonlarda karşımıza çıkacak olan toplam tahmini masraflardır. İşlem adımları şu şekildedir:

Adım 1. Önce birim değerlerine göre eşyalar büyükten küçüğe doğru Çizelge 3'de görüldüğü gibi sıralanır. Burada Açgözlü (*Greedy*) bir yaklaşım olduğunu belirtmek isteriz. Çünkü en değerli mallar ile çantanın doldurulması hedeflenmektedir. Bazı uygulamalarda bu sıralama yapılmadan da sonuca ulaşılabilmektedir. Zaman karmaşıklığı açısından bir şey fark etmese de hesaplama zamanı açısından çok fazla bir kazanç olduğunu açıktır. Bu durum Deneysel Uygulamalar bölümünde de ispat edilmektedir.

Adım 2. Bu aşamada üst sınır (UB) değeri bulunur. Basit mantıkla sadece en değerlikli eşya olan *C* malı ile çuval tam doldurulursa en kârlı seçim yapılmış olur. V=16 kg olduğuna göre teorik olarak çuvala yerleştirebilecek en değerlikteki malların TL karşılığı 16x10=160 TL olur. Bu durumda sezgisel fonksiyon:

$$f(x) = g(x) + h(x) = 0 + 160$$

İlerleyen aşamalardaki hesaplamalarda bulunabilecek sonuç kesinlikle 160'ı geçmeyecektir ve bu bilgi özyinelemeli alt fonksiyonlarda kullanılacaktır. Her alt fonksiyonda yeni bir $f(x)$ fonksiyonu yani *UB* değeri hesaplanacak, alt fonksiyonlar bu sınır değerine göre çağırılacak ya da çağırılmayacaktır.

Adım 3. Birim değeri en iyi olan sıradaki mal *C* malıdır. *C*'nin alınması ya da alınamaması durumunda yeni üst sınır değeri $f(x)$ fonksiyonu ile bulunsun:

a. *C*'nin çuvala alınması durumu:

C malı çuvala konulduğu vakit çuvalın kapasitesi 16-10=6 kg'ye düşmektedir. Sıradaki en değerli mal 2 numaralı maldır ve birim fiyatı 99'dur. Alınabilecek en fazla miktar 6x9=54'dür. Bu durumda

$$f(x) = g(x) + h(x) = 100 + 9\times6 = 154$$

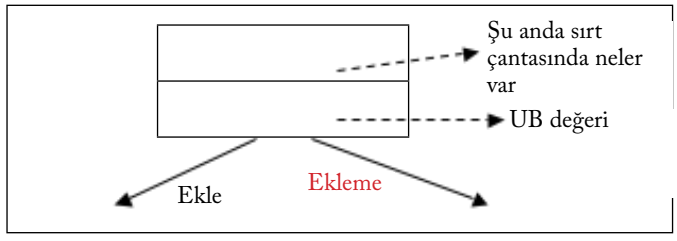
C'nin çuvala eklenmesi durumunda elde edilen çözüm en fazla 154 olabilmektedir. Alt fonksiyonlarda 154'den daha iyi bir çözüm bulmak imkânsızdır. Eğer alt fonksiyonlarda 154'den daha iyi bir çözüm bulunabileceği vaat ediliyorsa bu çözüm UB değerini aştığı için geçersizdir.

b. C'nin çuvala alınmaması durumu:

$$f(x) = g(x) + h(x) = 0 + 9\times16 = 144$$

C alınamıyorsa alınacak sıradaki eleman *B*'dir. V=16 olduğu için *B* alınırsa bu yolla elde edilen çözüm en fazla 144 olacaktır.

Adım 4. Bir önceki adımda elde edilen *a* ve *b* şıklarındaki çözümler özyinelemeli olarak tekrar tekrar hesaplanır. Çözüm yöntemini şekillerle ifade etmek için Şekil 1'de gösterilen kutucuk kullanılabilir.

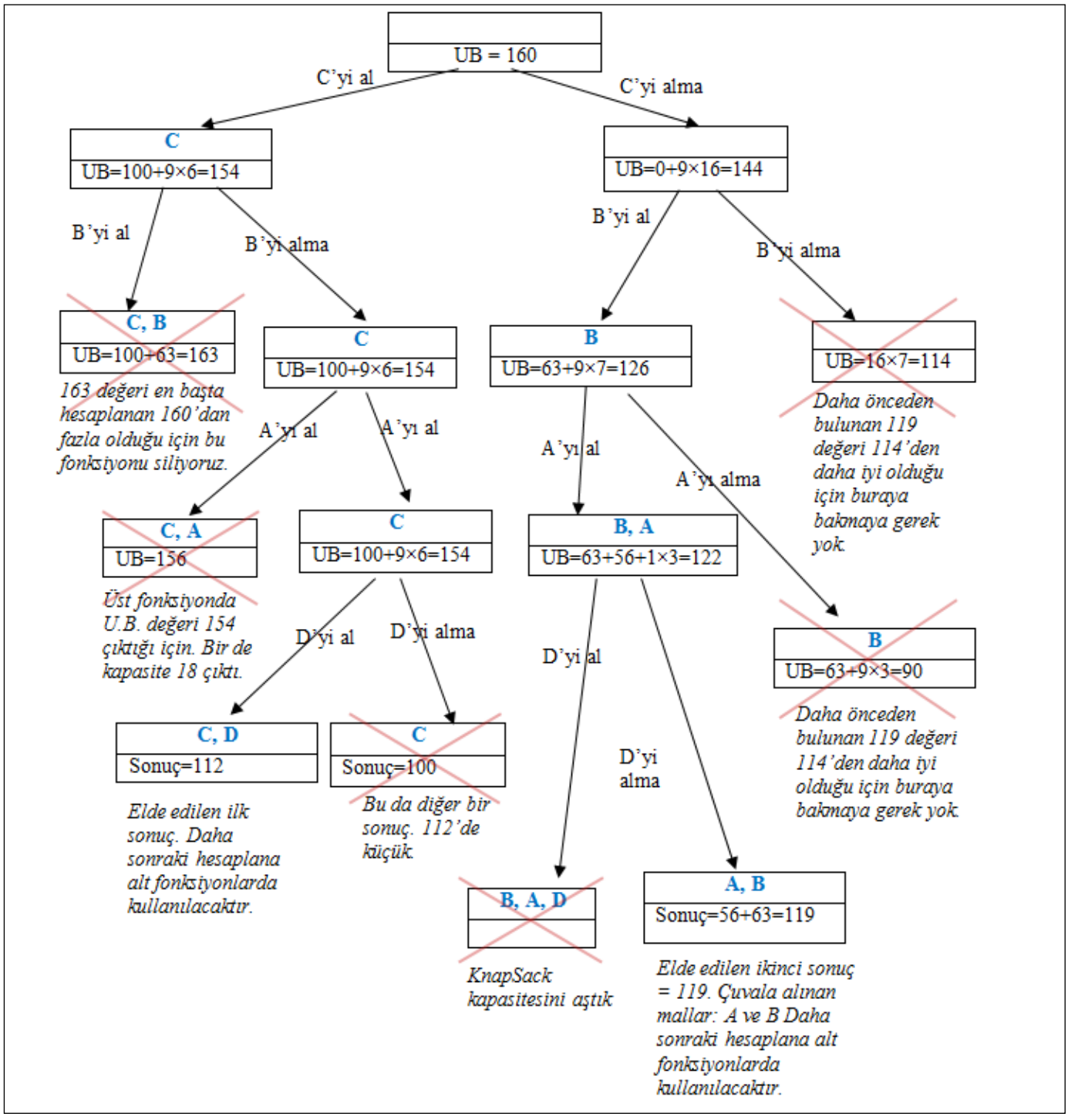


Şekil 1. Çözüm şekli.

Yukarıda açıklanan adımlar Şekil 2'de görülmektedir.

Özyinelemeli fonksiyonun bir sonraki iterasyona geçmesi diğer bir deyişle kendini çağırma şartları şu üç duruma bağlıdır:

1. Çuvalın kapasitesi alınan mallarla aşılmıyorsa,
2. Üst (*parent*) fonksiyonlarda elde edilen *UB* değerinden daha küçük sonuçlar bulunduysa,



Şekil 2. Çözüm aşamaları.

3. Yapraklardan birine ulaşıldıysa, fonksiyon kendini daha fazla çağıramayacaksa, yani bir sonuç değeri bulunduysa ve de daha sonraki fonksiyonlarda *UB* değeri bulunan bu sonuçtan daha küçük çıkarsa, alt fonksiyonlar yardımıyla hesaplama yapılmasına hiç gerek yoktur.

Şekil 2'de anlatılan çözümde de gösterildiği gibi her bir iterasyon sonucu elde edilen sonuçların sonraki aşamalarda kullanılmak üzere bellekte tutulması gerekmektedir. Rastgele bir şekilde elde edilen sonuçların bellekte tutulması ve gerektiğinde en düşük *UB* değerli sonucun işleme alınması için en uygun veri yapısının kullanılması zorunluluğu ortaya

Çizelge 4. Standart ve önerilen yöntemin hesaplama süresi karşılaştırılması.

N	Özyinelemeli (Recursion) Yöntem	Dal-Sınır Yöntemi	Sezgisel Dal-Sınır Yöntemi	Yığın Veri Yapılı Açgözlü Sezgisel Dal-Sınır Yöntemi
5	0 sn	0 sn	0 sn	0 sn
10	0,01 sn	0 sn	0 sn	0 sn
20	0,1 sn	0,09 sn	0,01 sn	0 sn
30	37 sn	2,8 sn	1,04 sn	0,2 sn
40	12 dk 38 sn	28 sn	6,1 sn	4,2 sn

çıkmıştır. Yığın veri yapısı (*Heap data structure*) bu tür sorunların çözümü için en uygun modeldir.

Bu yapıda en düşük *UB* değerine sahip alt çözüm modeli ikili ağaç yapısının en üst kısmında durur. Kendinden büyük *UB* değerlikli çözümler alt düğüm şeklinde ağaca yerleştirilir. *N* elemanlı bir veri kümesinde ağacın oluşturulmasının zaman karmaşıklığı $O(N\log N)$, ağaçtan bir elamanın alınması ya da eklenmesi durumunda ise ağacın yeniden düzenlenmesi $O(\log N)$ maliyetinde olacağı için oldukça avantajlıdır (Cormen vd. 2009). Zaman karmaşıklığında ifade edilen bu düşük maliyet, hesaplama zamanının normalden daha düşük olmasını sağlamaktadır ve oldukça avantajlı bir durum oluşturmaktadır.

Burada belirtmek isteriz ki, dinamik programlama yöntemi ile sırt çantası probleminin oldukça hızlı ve basit bir çözümü vardır (Martello vd. 1999). Dinamik algoritmalar daha çok alt problemlere bölünebilen problemlerde işe yarar ve zaman karmaşıklığını minimuma indirger. Dinamik çözüm demek; bir problemin çözümünü hesaplarken durumları sadece ve sadece bir kez hesaplayıp, sonucu hafızaya alıp, tekrar lazım olduğunda önceden kaydedilmiş sonucu kullanarak tekrar hesaplamaktan kaçınmaktır. Böylece sonucu bulmak için alt problemlere bölerek ilerlerken farklı büyük problemlerden aynı küçük alt problemlere gelindiği zaman küçük problemin sonucu tekrar hesaplanmayarak zamandan tasarruf sağlanacaktır (Ross 2014). Böylece en fazla yapılacak işlem sayısı her alt problem için yalnız bir hesaplama yapılacağından elde edilebilecek alt problem sayısı kadardır. Üzerinde çalışılan konu Dal-Sınır yönteminin değişik bir yöntemle entegre edilip optimize edilmesi üzerinedir, iki farklı kategorideki algoritmaların karşılaştırılması değildir.

Özyinelemeli Yöntem, Dal-Sınır Yöntemi, Sezgisel Dal-Sınır Yöntemi, Yığın Veri Yapılı Açgözlü Sezgisel Dal-Sınır Yöntemi ile ilgili çalışma zamanı karşılaştırmaları Çizelge 4'de verilmektedir. Eşya sayısı N'in 5, 10, 20, 30 ve 40 olması durumunda sn cinsinden 2.4 GHz hızına sahip Intel i7 işlemcili ve 16 GB RAM'li bir bilgisayarda elde edilen çalışma zamanı hesaplamaları çizelgede verilmektedir. Bu durumda üzerinde çalışılan ve önerilen Yığın Veri Yapılı Açgözlü Sezgisel Dal-Sınır Yönteminin diğerleri ile kıyaslandığında oldukça avantajlı olduğu görülmektedir.

5. Sonuç ve Tartışma

Bu çalışmada tam sayılı programlama problemlerinin çözümünde kullanılan dal-sınır algoritmasının daha hızlı çalışabilmesi için bir çözüm yaklaşımı sunulmuştur. Bir çok mühendislik problemin prototip hali olarak kabul edilen ve bir çok çeşidi olan sırt çantası problemine farklı bir yaklaşım ile çözüm önerisi getirilmiştir. Tam sayı programlama modellerinden biri olarak kabul edilen Dal-Sınır algoritmasının özyinelemeli yöntemin optimize edilmiş bir hali olarak değerlendirilse de bu modele sezgisel bir yaklaşım ile Açgözlü arama eklenerek özyinelemeli modelin en iyi şekilde optimizasyonu gerçekleştirilmeye çalışılmıştır. Ayrıca bu problem tipine uygun bir veri yapısı ile de hesaplama zamanı düşürülmüştür. Bu çalışmada salt amaç, yaygın bir şekilde bilinen ve Dinamik Programlama gibi daha hızlı çözüm yöntemleri ile çözülen bir problemi Dal-Sınır yöntemi ile çözmek değildir, bilinen klasik bir modele sezgisel ve açgözlü yaklaşım ile farklı bir bakış açısı getirerek optimizasyon yapmak ve uygun bir veri yapısı ile hesaplama zamanını en aza indirmektir.

6. Kullanılabilirlik

Bu çalışmada önerilen yöntemin C/C++ programlama dili ile derlenmiş hali açık kaynak olarak şu web sitesinden (Bulut 2018) elde edilebilir. İlgili yerde verilen kaynak kodlarının ve içinde kullanılan fonksiyonların açıklamaları kod içerisinde verilmiştir. Ayrıca örnek olması açısından *10 adet* örnek senaryo ve her biri için elde edilen sonuçlar test amaçlı olarak bu siteden indirilebilir.

7. Kaynaklar

Bosch, R., Michael, T. 2014. *Integer programming. Search methodologies*. Springer USA, 67-92.

Brucker, P., Jurisch, B., Sievers, B. 1994. A branch and bound algorithm for the job-shop scheduling problem. *Disc. App. Math.* 49(1):107-127.

Brusco, MJ., Stahl, S. 2006. Branch-and-bound applications in combinatorial data analysis. Springer Science & Business Media.

Bulut, F. 2018. Study of BB, Branch&Bound, https://sites.google.com/site/bulutfaruk/study-of-bb Erişim tarihi: Ocak 30, 2018.

Chekuri, C., Khanna, S. 2005. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM J.. Comp.*, 35(3): 713-728.

Conforti, M., Cornuéjols, G., Zambelli, G. 2014. *Integer Programming Models. Integer Programming*. Springer International Publishing, pp. 45-84.

Cormen, TH., Leiserso, CE. 2009. Introduction to algorithms. MIT press. ISBN-13: 978-0262033848, Cambridge.

Cornuéjols, G. 2007. Revival of the Gomory cuts in the 1990's. *An. Op. Res.,* 149(1), 63-66.

Delling, D., Sanders, P., Schultes, D., Wagner, D. 2009. Algorithmics of large and complex networks: design, analysis, and simulation. Springer, New York.

Fisher, ML. 2004. The Lagrangian relaxation method for solving integer programming problems. *Man. Sci.*, 50(12): 1861-1871.

Hochbaum, DS. 1995. A nonlinear knapsack problem. *Opr. Res. Let.*, 17(3): 103-110.

Kellerer, H., Pferschy, U., Pisinger, D. 2004. Introduction to NP-Completeness of knapsack problems. *In Knapsack problems*. Springer Berlin Heidelberg. pp. 483-493.

Khuri, S., Bäck, T., Heitkötter, J. 1994. The zero/one multiple knapsack problem and genetic algorithms. *In Proceedings of the 1994 ACM symposium on Applied computing.* pp. 188-193. Arizona.

Kong, M., Tian, P., Kao, Y. 2008. A new ant colony optimization algorithm for the multidimensional knapsack problem. *Comp. Op. Res.*, 35(8): 2672-2683.

Li, J., Xiu-xia S. 2008. A Route Planning's Method for Unmanned Aerial Vehicles Based on Improved A-Star Algorithm [J]. *Acta Armamentarii,* 7: 788-792.

Martello, S., Pisinger, D., Toth P. 2000. New trends in exact algorithms for the 0–1 knapsack problem. *European J. Op. Res.*123(2):325-332.

Martello, S., Pisinger, D., Toth, P. 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Mang. Sci.*, 45(3): 414-424.

Mitzenmacher, M., Upfal, E. 2017. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Cambridge university press.

Ohlsson, M., Peterson, C., Söderberg, B. 2008. Neural networks for optimization problems with inequality constraints: the knapsack problem. *Nrl. Ntw.*, 5(2).

Pisinger, D. 2005. Where are the hard knapsack problems?. *Comp. Op. Res.* 32(9):2271-2284.

Puchinger, J., Raidl, G. R., Pferschy, U. 2010. The multidimensional knapsack problem: Structure and algorithms. *INFORMS J. Comp.*, 22(2): 250-265.

Ross, SM. 2014. *Introduction to stochastic dynamic programming*. Academic Press, New York, USA.

Taha, HA. 2014. *Integer Programming: Theory, Applications, and Computations.* Academic Press, New York.

Vanderbei, RJ. 2015. *Linear Programming Book,* Publisher: Springer, New York.

Zhang, W. 1999. State-space search: Algorithms, complexity, extensions, and applications. Springer Science & Business Media.

Zou, D., Gao, L., Li, S., Wu, J. 2011. Solving 0–1 knapsack problem by a novel global harmony search algorithm. *App. So. Comp.*, 11(2): 1556-1564.