Eczane Nöbet Çizelgeleme Problemleri

Program Kodu: 3501

Proje No: 111M007

Proje Yürütücüsü:
Yrd. Doç. Dr. Özgür ÖZPEYNİRCİ

Bursiyerler:
Gökhan CEYHAN
Fatih KOCATÜRK
Cansu KANDEMİR
Aybike ÖZDEMİREL

MAYIS 2014
İZMİR

ÖNSÖZ

Bu projenin ana fikri bir tatil günü hamile eşime ilaç almak için çok uzun yol kat etmemden sonra ortaya çıktı. Bu olaydan sonraki ilk iş günü, eczacılar ile görüştüm ve nöbet çizelgelerinin nasıl hazırlandığı hakkında bilgi aldım. Daha sonra, bu problemi yüksek lisans öğrencim olan Ebru AĞLAMAZ'a tez konusu olarak önerdim ve beraber çalışmaya başladık. Yüksek lisans tezinin son aşamalarında proje önerisini TÜBİTAK'a sundum.

Projenin ilk yılında Cansu KANDEMİR (Aralık 2011 - Temmuz 2012) ve Aybike ÖZDEMİREL (Aralık 2011 - Mayıs 2012) bursiyer olarak ancak kısa süreli yer aldılar. Fatih KOCATÜRK (Eylül 2012 - Mayıs 2014) ve Gökhan CEYHAN (Ekim 2012 - Mayıs 2014) projede bursiyer olarak uzun süreli görev aldılar. Ebru AĞLAMAZ kariyerine Mutfak Sanatları Bölümünde devam ettiği için ne yazık ki projede görev alamadı.

Projede, Eczane Nöbet Çizelgeleme problemine, dal-sınır ve dal-fiyat algoritmaları olmak üzere, iki farklı kesin çözüm algoritması önerdik. İkinci algoritma (dal-fiyat), ilkinden (dal-sınır) ve genel amaçlı çözücüden (IBM ILOG CPLEX) daha hızlı çalışıyor ancak bu algoritma bile büyük boyutlu problemlerde en iyi sonuca ulaşamıyor. Büyük boyutlu problemler için Tabu Arama ve Değişken Komşuluk Arama sezgiselleri geliştirdik. Her iki sezgisel de gerçek hayat problemlerine mevcut çözümlerden daha iyi çözümler üretti. Özellikle değişken komşuluk arama ile çok iyi (alt sınırdan %0,5 uzak) sonuçlar elde etmeyi başardık.

Ayrıca problemi iki amaçlı olarak modelledik ve kesin bir yöntem kullanarak tüm etkin sonuçlarını bulduk. Büyük boyutlu problemlerde kesin yöntem ile etkin sonuç kümesini elde edemediğimiz için İki-Amaçlı Değişken Komşuluk Arama sezgiseli önerdik ve test sonuçları raporladık.

Bu projeye vermiş olduğu destekten dolayı TÜBİTAK'a, bilgi ve belge paylaşımındaki desteklerinden dolayı İzmir Eczacılar Odası'na, kıymetli geri bildirimleri için proje danışmanına, Bora KAT ve Ercan AYAZ başta olmak üzere TÜBİTAK Mühendislik Araştırma Grubu ve Mali Denetleme ve Sözleşmeler Müdürlüğü çalışanlarına teşekkürlerimizi sunuyoruz.

İÇİNDEKİLER

TABLO LİSTESİ

ŞEKİL LİSTESİ

ALGORİTMA LİSTESİ

ÖZET

Bu proje kapsamında Eczane Nöbet Çizelgeleme problemleri üzerinde çalıştık. Problemin çeşitli türevlerini tanımladık ve her birisi için matematiksel programlama modellerini ortaya koyduk. Gerçek hayat probleminin hesaplama karmaşıklığını NP-Zor olarak belirledik. Probleme özgü kesin ve sezgisel çözüm yöntemleri geliştirdik. Kesin çözüm yöntemi olarak dal-sınır ve dal-fiyat algoritmaları, sezgisel yöntem olarak tabu arama ve değişken komşuluk arama algoritmaları önerdik. Probleme özgü alt sınır algoritmaları, alt problemlere özgü sezgiseller tasarladık. Problem boyutunu azaltmak için çeşitli yöntemler önerdik.

Geliştirdiğimiz algoritmaların performanslarını test etmek amacıyla küçük ve büyük boyutlu rassal örnekler oluşturduk. Ayrıca İzmir iline ait gerçek verileri derleyerek, iki büyük boyutlu gerçek hayat örneği elde ettik.

IBM ILOG CPLEX genel çözücüsü küçük boyutlu örneklerin bir kısmını çözebilmektedir. Dal-sınır ve dal-fiyat algoritmalarının temel hallerinin ve çeşitli türevlerinin performanslarını rassal örnekler üzerinde test ettik. Dal-sınır algoritması genel çözücü ile benzer performans göstermektedir. Dal-fiyat algoritması ise genel çözücüden daha iyi performans göstermekte ve onun çözemediği örnekleri çözebilmektedir.

Büyük boyutlu örnekler ve gerçek hayat örnekleri kesin çözüm yöntemleri için fazla büyüktür. Bu sebeple, sezgisel yöntemler önerdik. Tabu arama ve değişken komşuluk arama yöntemleri ile kısa sürede olurlu çözümler elde ettik. Gerçekçi problemler için değişken komşuluk arama yöntemi ile alt sınır değerine %0,5'den daha yakın sonuçlar elde etmeyi başardık.

Problemin ilk amacı olan talep ağırlıklı kat edilen yol miktarını en azlamaya ek olarak en az iş yükü oranına sahip eczanenin iş yükünü en çoklama amacını tanımladık. Bu iki amaçlı problemin tüm etkin sonuçlarını bulabilmek için kesin ve sezgisel yöntemler önerdik. Küçük boyutlu örneklerde matematiksel programlama ile tüm etkin sonuçları bulduk. Büyük boyutlu örnekler için değişken komşuluk arama yöntemi geliştirdik. Sezgisel yöntemin performansını süreye ek olarak etkin sonuçlar kümesini bulma başarısı ile değerlendirdik.

Anahtar kelimeler: Eczane nöbet çizelgeleme problemi, matematiksel programlama, alt sınır, üst sınır, dal-sınır algoritması, dal-fiyat algoritması, tabu arama, değişken komşuluk arama, çok amaçlı karar verme, etkin sonuç.

ABSTRACT

In this project, we studied Pharmacy Duty Scheduling problems. We defined different types of the problem and formulated mathematical programming models for each type. We showed that the real life problem is NP-Hard and developed problem specific exact and heuristic solution approaches. We proposed branch-and-bound (B&B) and branch-and-price (B&P) algorithms as exact methods. Besides, we presented tabu search (TS) and variable neighborhood search (VNS) heuristics. We designed lower bound algorithms for the original problem and some heuristics for the subproblems. We addressed problem size reduction algorithms. We acquired real life data of İzmir and compiled two large instances in addition to the randomly generated instances.

IBM ILOG CPLEX solver can solve some of the small size instances. We tested the performance of B&B and B&P algorithms on the random instances. Although B&B shows similar performance to the generic solver, B&P performs better and can solve more problems.

The real life problem is too large to be able to solve with exact approaches. Therefore, we recommend the use of heuristics. We reported feasible solutions in small computation times with TS and VNS algorithms. For the real size instances, VNS is able to find feasible solutions with objective function value within the 0.5% of the lower bound.

We introduced second objective as the maximization of the workload of the pharmacy having the minimum workload ratio. We developed exact and heuristic methods to generate all efficient solutions. At small instances, we found all efficient solutions by mathematical programming. Furthermore, we developed VNS algorithm for the large size problems. We tested the performance of the heuristic method by the ability of finding efficient solutions in addition to the CPU time.

Keywords: Pharmacy duty scheduling problem, mathematical programming, lower bound, upper bound, branch and bound algorithm, branch and price algorithm, tabu search, variable neighborhood search, multiple objective decision making, efficient solution.

1. GİRİŞ

Türkiye'de her eczane bağımsız bir işletmedir ve bir eczacı tarafından işletilir. Mesai saatleri içerisinde tüm eczaneler halka hizmet verirler. Mesai saatleri haricinde ise, eczaneler belirli bir çizelgeye göre nöbet tutar ve bu saatlerde halkın ilaç taleplerini nöbetçi eczaneler sağlar. Nüfusun yüksek olduğu yerlerde aynı günde birden fazla eczane nöbetçi olabilir. Aynı gün nöbetçi olan eczanelerin şehre düzgün bir şekilde yayılmış olması halka sunulan bu hizmetin seviyesinin yüksek olmasını sağlar.

Bu proje kapsamında Eczane Nöbet Çizelgeleme Problemleri (ENÇP) üzerinde çalıştık. Öncelikle literatür taraması yaptık ve benzer problemler ile bu problemlerin çözümü için kullanılan yöntemleri inceledik. İzmir Eczacılar Odası yöneticileri ile görüşmeler yaparak problem hakkında bilgi edindik. Problemin tek ve iki amaçlı çeşitli türevlerini tanımladık ve bu problemleri çözmek için kesin ve sezgisel yöntemler geliştirdik. Ayrıca alt sınır algoritmaları ürettik.

Geliştirilen yöntemlerin performanslarını test etmek için rassal örnekler oluşturduk. İzmir Eczacılar Odası desteği ile eczane listelerini ve geçmiş dönem nöbet çizelgelerini elde ettik. Türkiye İstatistik Kurumu'ndan satın aldığımız verileri de kullanarak gerçek hayat örnekleri hazırladık.

Dal-fiyat algoritması küçük boyutlu problemlerin %96'sını (86/90) süre limiti içinde çözmeyi başardı. Büyük boyutlu problemlerde ise değişken komşuluk arama algoritması alt sınırdan en fazla %1,2 farklı sonuçlar bulmuştur. Gerçek hayat problemlerinde önemli iyileştirmeler elde ettik. Uygulanan çözüm ile alt sınır arasındaki fark %5,87 seviyesinde iken, bu fark önerilen çözüm için %0,54'den azdır.

Rapor şu şekilde organize edilmiştir: İkinci Bölüm'de literatür özeti sunduk. Kullandığımız gereç ve yöntemleri Bölüm 3'de, deney sonuçlarını Bölüm 4'de aktardık. Beşinci Bölüm'de projenin sonuçlarını ve önerilerimizi tartıştık. Ek'de proje kapsamındaki çalışmaların bir kısmını içeren ve Computer & Operations Research dergisinde yayınlanmak üzere kabul edilen makalemizi sunduk.

2. LİTERATÜR ÖZETİ

Bu bölümde proje kapsamında incelenecek problem ve kullanılacak yöntemler ile ilgili yapılan literatür taraması özetlenmiştir. Bu bölüm, tesis yerleşim problemleri, kesin çözüm yöntemleri, sezgisel yöntemler ve çok amaçlı karar verme alt başlıklarından oluşmuştur

2.1 Tesis Yerleşim Problemleri

Hale ve Moberg (2003) tesis yerleşim problemini belli kısıtları sağlayan müşteri düğümleri kümesinin maliyetini en azlamak için tesisler kümesindeki tesislerin fiziksel olarak nereye yerleştirileceğini araştıran bir yöntem olarak tanımlamışlardır. Onlar ayrıca tesis yerleşim modellerinin modellerin amaç fonksiyonuna, yerleştirilecek tesislerin sayısına ve boyutuna ve diğer karar belirleyicilerine göre farklılık gösterebileceğini vurgulamışlardır. ReVelle v.d. (2008) yerleşim modellemedeki çalışmaları dört gruba ayırmışlardır; çok sayıda basitleştirme varsayımlarına dayanan analitik modeller; müşteriler genellikle ayrık yerleştirilirken tesislerin her yere yerleştirilebildiğini varsayan sürekli modeller; yerleşim probleminin düğümlerden ve bağlantılardan oluşan bir ağın içinde gömülü olduğunu varsayan ağ modelleri; müşterilerin ve tesislerin ayrık olduğu ayrık yerleşim modelleri. ENÇP ayrık yerleşim modelleri içinde yer almaktadır.

P-ortanca problemi yerleşim biliminde yaygın olarak kullanılmaktadır. Kariv ve Hakimi (1969) *p*-ortanca probleminin değişken *p* için NP-Zor bir problem olduğunu göstermişlerdir. Araştırmacılar *p*-ortanca problemini endüstri tesisleri yerleşimi, depolar ve halk tesisleri yerleşimi gibi birçok gerçek hayat problemini modellemede kullanırlar. Araştırmacılar aynı zamanda *p*-ortanca problemini sınıflandırma analizi (Hansen ve Jaumard, 1997) ve veri madenciliği (Ng ve Han, 1994) gibi diğer araştırma alanlarına uyarlamışlardır.

İtfaiyeler, ambulanslar, polis karakolları ve hastaneler önemli acil servis istasyonlarına örnektir. Acil Servis İstasyonu (ASİ) yerleşim problemi literatürde büyük ilgi görmüş çok önemli bir problemdir. Başar v.d. (2011) ASİ yerleşim problemleri için bir taksonomi önermişlerdir. Adenso-Diaz ve Rodriguez (1997), İspanya'nın Leon ilindeki ambulansları yerleştirmek için TA sezgiselini önermişlerdir. Çatay v.d. (2008), İstanbul, Türkiye için tek ve çok dönemli acil istasyon yerleşim problemini incelemişler ve büyük boyutlu problemler için sezgisel yöntemler önermişlerdir. Harewood (2002) Barbados adasındaki ambulansları yerleştirmek için çok amaçlı programlama

problemini önermiştir. Carreras ve Serra (1999), servis eşik değerlerini göz önüne alarak İspanya'nın kırsal kesimlerindeki yeni eczanelerin yerleştirilmesi üzerine çalışmışlar ve bir TA sezgiseli önermişlerdir. ENÇP aynı zamanda bir ASİ problemidir. Buna rağmen, ENÇP için eczanelerin yerleri bilinir ve bu problemde toplumun acil ilaç ihtiyaçlarına iyi kalitede servis sağlamak amacıyla eczanelere nöbet atamayla ilgilenilir.

2.2 Kesin Çözüm Yöntemleri

Bu alt başlıkta probleme özgü olarak geliştirilmiş dal-sınır ve dal-fiyat algoritmaları ile ilgili literatür özeti sunulmuştur.

Özpeynirci ve Azizoğlu (2008) esnek imalat sistemlerinde operasyon atama ve kapasite tahsis problemini dal-sınır algoritması yardımıyla çözmüşlerdir. Makine ve zaman kısıtları altında maksimum ağırlığı verecek operasyon altkümesini seçmeyi amaçlamışlardır. Dal-sınır algoritmasında kullanmak üzere beş farklı üst sınır algoritması geliştirmişlerdir. Çözüm süresinin önemli olduğu durumlarda hızlı çalışan algoritmaları diğer durumlarda ise daha çok süre alan bununla birlikte daha yüksek kalitede sonuç veren algoritmaları kullanmışlardır.

Karsu ve Azizoğlu (2012) darboğaz çok kaynaklı genelleştirilmiş atama problemi için bir dal-sınır algoritması geliştirmişlerdir. Problemin amacını en yoğun ajanın yükünü enazlamak olarak belirlemişlerdir. Geliştirmiş oldukları iki alt sınır algoritmasını ve optimal doğrusal programlama gevşetmesi çözümlerini dal-sınır algoritmalarında birlikte kullanmışlardır. Optimal doğrusal programlama gevşetmesi çözümünde en kesirli değer alan değişken üzerinden dallandırmaya gitmişlerdir ve o değişkenin temsil ettiği işi her bir ajana atayarak tüm olası dalları oluşturmuşlardır. Sonrasında en düşük alt sınır değerine sahip değişken üzerinden yeni dallar oluşturmuşlardır. Alt sınır çözümünü olurlu bir çözüme çevirerek ve belirli sayıda döngü süresince iyileştirmeye çalışarak elde ettikleri çözümleri aday çözüm olarak kullanmışlardır. Geliştirmiş oldukları algoritma sayesinde problem büyüklüğü ajan sayısı 5 iken 60, ajan sayısı 10 iken 30 işe kadar olan problemleri 20 dakikadan az bir sürede optimal olarak çözebilmektedirler.

Dal-sınır algoritmasının kullanıldığı bir diğer problem türü ise tesis yerleşim problemidir. Tesis yerleşim problemleri genellikle tamsayı ya da karışık tamsayı modellendiğinden dolayı dal-sınır algoritmaları sıkça başvurulan yöntem olmaktadır. Buna örnek olarak Dupont (2007) çalışmasını

örnek gösterebiliriz. Bu çalışmadaki amaç yatırım, üretim ve dağıtım maliyetlerini tüm müşteri talebini karşılayacak şekilde enazlayacak tesis konumlarını potansiyeller arasından seçmek olarak tanımlanmıştır. Çalışmasında sayma planını (enumeration scheme) raporlamış ve dallanılacak değişken olarak maksimum konumdan servis alabilen müşteri seçilmiştir. Müşteri sayısı, potansiyel konum sayısı gibi problem parametrelerinin algoritmanın performansı üzerindeki etkisini araştırmışlarıdır. Geliştirdiği algoritma 20-25 konumlu ve 250'den fazla müşterili problemleri uygun zamanlarda çözebilmektedir.

Fayed ve Atiya (2012) *k*-merkez problemi için dal-sınır algoritması geliştirmişlerdir. Bu problemde amaç belirli bir noktalar kümesini olabilecek en küçük boyutta halkalar ile kapsayabilmek olarak tanımlanmıştır. Algoritmalarında önce-derinlik (depth first) ve önce-genişlik (breadth first) karışımı bir yaklaşım uygulamışlardır. Önce-derinlik tekniğinde hafıza problemi en aza indirilirken bir yandan da çok fazla sayıda alt ağacın değerlendirildiği ve bunun da fazla maliyete yol açtığı belirtilmiştir. Optimal çözüme daha hızlı ulaşmak için aynı seviyedeki potansiyeli yüksek düğümlerin her biri için en alt seviyeye kadar inerek birer aday çözüm elde etmişlerdir. Buna temel olarak ise farklı yapıdaki olurlu çözümlerin iyi birer aday çözüm olacağını öne sürmüşlerdir. Bu tekniklerinin sade dal-sınır algoritmasından daha iyi performans gösterdiğini belirtmişlerdir.

Kapasiteli tesis yerleşim problemlerine kesin çözüm yaklaşımı olarak lagrange gevşetmesi tabanlı dal-sınır algoritması öneren Holmberg v.d. (1999) sonuçlarını CPLEX 3.0 çözümleri ile karşılaştırmışlardır. Yaratılan düğüm sayısı ve çözüm süresine kısıtlar koyarak yaptıkları testler sonucunda algoritmalarının CPLEX 3.0'a göre daha iyi bir performans ortaya koyduğunu öne sürmüşlerdir.

Nadirler ve Karasakal (2007), tek tesis yerleşim problemi için karışık tamsayı programlama tabanlı bir çözüm yaklaşımı geliştirmişlerdir. Çalıştıkları problemde amaç, istenmeyen bir tesisin yaşam alanlarından olabildiğince uzağa yerleştirilmesidir. Problemin çözüm kısmında CPLEX'in varsayılan parametrelerini değiştirerek deneyler yapmışlardır. Değişken seçimi, düğüm seçimi gibi kararları değiştirerek en iyi çözümü veren parametreleri bulmaya çalışmışlardır.

Dantzig-Wolfe ayrıştırmasının mümkün olduğu problemlerde, ayrıştırılmış problemin doğrusal gevşetilmiş en iyi sonucu orijinal formülasyondan elde edilecek doğrusal gevşetme sınırına göre

daha iyi sonuç vermektedir (Barnhart vd.,1996). Bu özellik, dal-fiyat algoritmasında yaratılan düğümlerde daha kaliteli alt sınırlar elde edilmesini sağlamaktadır.

Dal-fiyat algoritmaları yapısal olarak dal-sınır algoritmalarına benzer olmak ile birlikte, düğümlerde alt sınırlar elde etmek için kullanılan problemin doğrusal gevşetmesini çözerken farklılıklar içermektedir. Dal-fiyat algoritmaları düğümlerdeki doğrusal gevşetme problemini kolon türetme algoritması ile çözmektedir.

Kolon türetme algoritmasının temel özelliği, problemi tek seferde çözmek yerine, karar değişkenlerinin küçük bir alt kümesi ile problemi çözmek ve sonrasında kademeli bir şekilde karar değişkenlerini probleme ekleyip tekrar çözmektir. Bu özellik, çok fazla sayıda karar değişkeni içeren problemleri çözmede büyük bir avantaj sağlamaktadır. Bu problemler için doğrusal gevşetme problemi çözüldüğünde, karar değişkenlerinin büyük bir kısmı optimal çözümde sıfır değeri almaktadır. Bu yöntem büyük problemler için verimsiz olmaktadır. Kolon türetme yöntemi kullanıldığında ise, her seferinde daha küçük bir problem çözülmüş olacaktır. Karar değişkenlerinin büyük bir kısmı dışarıda tutularak, sadece en karlı azaltılmış maliyete (reduced cost) sahip değişkenler probleme eklenecektir. Bu amaçla fiyatlandırma alt problemi kullanılmaktadır. Bu şekilde ilerlenerek, eklenecek değişkenin azaltılmış maliyeti karlı olmaktan çıktığı an algoritma sona erecektir.

Kolon türetme algoritmasında, her döngüde çözülen fiyatlandırma alt problemini en iyi çözmek yerine sezgisel yaklaşımların kullanımına rastlanmaktadır (Barnhart vd.,1996). Sezgisel olarak bulunan azaltılmış maliyet karlı olmaktan çıktığı anda fiyatlandırma alt problemi tekrar optimal çözümü elde edecek şekilde çözülmektedir.

Barnhart vd. (1998)'nin önerdiği bir diğer algoritmayı hızlandırma yaklaşımı ise her düğümde doğrusal gevşetme problemini en iyi şekilde çözmek yerine bu değer için bir sınır elde ederek devam etmektir. Bu yaklaşım, doğrusal gevşetme probleminde en iyiliği kanıtlamanın çok fazla döngü gerektirdiği durumlarda etkili olmaktadır.

Solyalı ve Özpeynirci (2009), yayılmış zaman kısıtı altında operasyonel sabit iş çizelgeleme problemi için bir dal-fiyat algoritması geliştirmişlerdir. Bu problemde amaç, sabit başlangıç ve bitiş zamanları olan iş setinden bir alt kümeyi seçerek birbirinin aynısı paralel makineler üzerinde

işlemek ve bu kararı seçilen işlerin ağırlıkları toplamını ençoklayacak şekilde yapmaktır. Bu problemlerde makine sayısı problem parametresidir ve bu özelliğiyle taktiksel sabit iş çizelgeleme problemlerinden ayrılırlar. Yayılmış zaman kısıtı bir makinenin çalışmaya başladıktan sonra ne kadar süre boyunca çalışır vaziyette olacağını ifade eder.

Problemin matematiksel modelinde karar değişkenleri iş makine ataması olarak belirlenmiştir. Dantzig-Wolfe ayrıştırmasından sonra ise karar değişkenleri bir makine için olurlu bir çizelgenin oluşturulmak istenen asıl çizelgede kullanılıp kullanılmaması kararıdır. Ana problemde, tek makinelik çizelgelerden toplam makine sayısı kadar olanı her işin birden fazla makineye atanmaması kısıtını sağlayacak şekilde seçilmektedir. Fiyatlandırma alt probleminde ise tek makinelik olurlu çizelgelerden azaltılmış maliyeti en çok olan seçilmektedir.

Kolon türetme sayesinde dal-fiyat algoritması test örneklerinin %86,8'ine kök düğümde en iyi çözümü bulabilmektedir. Bulunamadığı zamanlarda ise iş atama değişkenleri üzerinden dallandırma yapılarak çözüme gidilmektedir. Dal-fiyat algoritmasının çözüm süreleri dal-sınır algoritmasının sonuçları ile karşılaştırılmış ve dal-fiyat algoritmasının çok daha üstün bir performans sergilediği görülmüştür. Yazarlar, bu üstünlükteki temel etkenin dal-fiyat algoritmasının sağladığı kaliteli sınırlar olarak yorumlamaktadırlar.

2.3 Sezgisel Çözüm Yöntemleri

Bu alt başlıkta Tabu Arama (TA) ve Değişken Komşuluk Arama (DKA) ileri sezgisel yöntemleri ile ilgili literatür özetlenmiştir.

Bir ileri-sezgisel yöntem olan TA, yerleşim problemlerinde ve acil tesis yerleşim problemlerinde sıkça kullanılmaktadır. Glover (1989), TA algoritmasını ve tabu yıkma kriteri, tabu listesi ve bellek fonksiyonları gibi genel özelliklerini tanıtmıştır. Glover (1990), TA algoritması için değişik komşu arama yöntemlerini tanıtmıştır. Tabu Arama algoritmasının temelleri, değişik örnekler ile birlikte Glover (1990)'da anlatılmıştır. Önceki araştırmalara ek olarak, Glover ve Laguna (1997) TA sezgiselinin temelleri ve varyasyonlarını detaylı bir şekilde kitaplarında anlatmaktadırlar.

Gendreau vd. (1997) ambulans yerleşim problemini çözmek için TA algoritmasını kullanmışlardır. Statik bir kapsama modelini çifte kapsama problemi olarak tanımlamışlar ve belirli bir zaman limiti içerisinde en az iki ambulans tarafından kapsanan nüfusu ençoklamaya

çalışmışlardır. Algoritmaya başlangıç çözümü için ilk önce problemin doğrusal gevşetilmiş çözümünü elde etmişlerdir. Daha sonra bu çözüm olursuz ise olurlu bir çözüm haline getirmek için kapsama kısıtlarını dikkate alan çeşitli algoritmalar kullanmışlardır. Bir lokasyonda bir veya birden fazla ambulans varsa, bu ambulans bu lokasyona en yakın diğer beş olası lokasyona kaydırılarak daha iyi çözümler elde edilmeye çalışılmıştır. Bir kaydırma işleminin yapılması halinde bu işlem rassal bir değişken olarak belirlenen tabu süresi kadar süre tekrar değerlendirmeye alınmamıştır. Daha farklı çözümler elde edebilmek için ise (diversification) en yakın beş lokasyon yerine tüm olası lokasyonlar denenmiştir ve amaç fonksiyonunda iyileştirme sağlayan çözüme gidilmiştir. İleriki çalışmalarında Gendreau vd. (2001), dinamik bir problem olan ambulans yerlerinin gerçek zamanda yeniden düzenlenmesi üzerine çalışmışlar ve çözümlerini ambulans yönetim sistemine entegre etmeyi amaçlamışlardır. Olası bir çağrıya bir ambulansın görevlendirilmesinden sonra en kısa sürede ambulansları yeniden düzenlemeye çalışmışlardır. Bu sebeple, yeniden yerleştirme senaryolarını hesaplamak için paralel TA algoritmasını kullanmışlardır. Algoritmanın çözüm süresini kısaltmak için paralel hesaplama tekniğinden faydalanmışlardır.

Doerner vd. (2005), Avusturya'daki bir ambulans yerleşim problemini çifte kapsama modeli olarak tanımlamışlar ve Gendreau vd. (1997) tarafından geliştirilen modelin bir uzantısını problemi modellemek için kullanmışlardır. Problemi çözmek için Gendreau vd. (1997) tarafından geliştirilen TA algoritmasını ve Ant Colony Optimization (ACO) algoritmasını kullanmışlardır. TA algoritmasında başlangıç çözümünü doğrusal gevşetme kullanarak değil rassal bir şekilde oluşturmuşlardır. Küçük problemlerde her iki sezgisel de en iyi çözüme ulaşmışlardır. TA sezgiseli en büyük ölçekli problemlerde daha iyi performans gösterse dahi hangi algoritmanın çözüm kalitesi bakımından daha iyi olduğu sonucuna varmak zordur. Ancak, TA algoritması çözüm süresi bakımından ACO'ya üstünlük sağlamaktadır.

Başar vd. (2011) iki farklı zaman kısıtını dikkate alan çok dönemli problem için TA algoritması geliştirmiştir. Bu algoritmayı rassal oluşturulmuş problemler ve bir gerçek hayat verisi üzerinde test etmişlerdir.

Rajagopalan vd. (2008), ambulansların dinamik bir şekilde yeniden yerleştirilmesi üzerine çalışmıştır. Ambulanslara olan talebin hafta içerisinde ve hatta gün içerisinde değiştiğini belirtmişlerdir. Böylece, ambulansları yeniden yerleştirip konuşlandırarak servis performansını

arttırmayı hedeflemişlerdir. İlk önce, yerleştirilecek en az ambulans sayısını tanımlayabilmek için bir arama algoritması önermişlerdir. Daha sonra ise, eldeki ambulansları yerleştirmek için bir TA algoritması kullanmışlarıdır.

Ağlamaz (2011) eczane nöbetlerinin planlanması problemi üzerinde çalışmıştır. Bu çalışmada problemin farklı versiyonları tanımlanmıştır. Ayrıca, problem için bir TA algoritması geliştirilmiş ve deneyler yapılmıştır.

Rolland vd. (1996), *p*-ortanca problemi için TA yöntemlerini baz alarak yeni bir çözüm sezgiseli geliştirmiştir. Sonuçlar iyi bilinen iki sezgisel ile karşılaştırılmış ve daha iyi sonuçlar bulunmuştur.

Wang vd. (2003), içerisinde yeni tesislerin açılıp var olanların kapandığı bütçe kısıtlı problem üzerinde çalışmıştır. Modellerini, büyük bir şehirdeki banka şubelerinin yerleşimi ve tekrar yerleşimi üzerinde uygulamışlardır. İlk önce matematiksel programlama modeli daha sonra ise üç sezgisel yaklaşım geliştirmişlerdir. Al-Sultan vd. (1999) TA sezgiselini kullanarak kapasite kısıtsız tesis yerleşim problemi üzerine çalışmıştır.

Bilgin ve Azizoğlu (2009) makine çizelgeleme problemi üzerine çalışmışlardır. Bu çalışmada, operasyonların değişik makinelere atanması ve kapasite ile aletlerin tahsis edilmesi amaçlanmıştır. Bu problem için ilk önce matematiksel bir model geliştirmişlerdir. Sonrasında, alt sınırlar elde edebilmek için değişik sezgisel metotlar ve lagrange gevşetme algoritması önermişleridir. Nihai olarak, problemi çözmek için bir TA algoritması kullanmışlardır.

P-ortanca probleminin hesaplama karmaşıklığı araştırmacıları sezgisel metotlar geliştirmeye yönlendirmiştir. Mladenoviç v.d. (2007) *p*-ortanca problemi için kullanılan sezgiselleri klasik sezgiseller ve akıllı sezgiseller olarak iki gruba ayırmışlardır. Klasik sezgiseller açgözlü, cimri, çiftli yükselme, bileşik, değişimli, değiştirme, dinamik programlama, langrange gevşetme ve toplama sezgisellerini içermektedir. Akıllı sezgiseller ise tabu arama (TA), DKA, genetik algoritma (GA), dağılım arama, tavlama benzetimi (TB), sezgisel konsantrasyonu, karınca koloni optimizasyonu (KKO), sinir ağları, ayrıştırma sezgiselleri ve melez sezgisellerini (MS) içermektedir.

Alp v.d. (2003) *p*-ortanca problemine etkili bir GA geliştirmişler ve bu algoritmayı 100 düğüm ile 1000 düğüm arasında değişen 80 problem üzerinde test etmişlerdir. Levanova ve Loresh (2004) TB algoritmasını önermişler ve OR kütüphanesindeki (Beasley,1985) test örneklerinin ilk 20 tanesinin (40 problem arasından) 17 tanesini tam olarak çözmüşlerdir. Resende ve Werneck (2003) ve Kochetov v.d. (2005) KKO algoritmasının iki farklı iyileştirmesini önermişler ve OR kütüphanesindeki 20 örneğin hepsini çözmeyi başarmışlardır.

Mladenoviç ve Hansen (1997) yerel en iyi çözüme takılmayı önleyen yeni bir akıllı sezgisel, DKA, tanımlamışlardır. Bu basit ve etkili akıllı sezgiseli yerel arama algoritması içinde sistematik komşuluk değişimi olarak tanımlamışlardır. DKA normal bir yön takip etmek yerine aramayı mevcut çözümün artan uzak komşuluklarında yapar ve yeni bir çözüme ancak ve ancak iyileştirme elde ederse geçer.

Hansen ve Mladenoviç (1997) *p*-ortanca problemini çözmek için DKA sezgiselini önermişler ve DKA sezgiselini TA sezgiseli ile karşılaştırmışlardır. TA sezgiseline göre daha iyi sonuçlar almışlar ancak büyük boyutlu problemler için DKA sezgiseli çok zaman almıştır. DKA sezgiselinin çözüm süresini azaltmak için Hansen ve Mladenoviç (2001a) Değişken Komşuluk Ayrıştırma Arama (DKAA) sezgiselini geliştirmişlerdir. DKAA sezgiseli arama uzayını ayrıştırır ve yerel aramayı oldukça küçük uzaylarda yapar. DKAA sezgiselini *p*-ortanca problemi üzerinde göstermişler ve TSP kütüphanesinin (Reinelt, 1991) 1400, 3038 ve 5934 kullanıcı örneklerinde test etmişlerdir. DKAA sezgiselinin az çözüm süresinde DKA sezgiseline göre ciddi derecede iyileştirme sağladığını göstermişlerdir.

Hansen ve Mladenoviç (2001b) büyük boyutlu problemler için DKA sezgiselinin birçok türevlerini önermişlerdir: yerel aramanın uzun zaman aldığı büyük problemlerde çözüm süresini azaltmak için temel DKA sezgiselinin yerel arama adımını uygulamayan Azaltılmış DKA (ADKA); mevcut çözümden daha uzak bölgeleri kolayca aramayı sağlamak için bazı olasılıklarla mevcut çözümden daha kötü bir çözüme geçmeye izin veren etkili Yamuk DKA (YDKA). Aynı zamanda bu DKA türevlerini bazı klasik optimizasyon problemlerine uygulamışlardır. Daha sonra araştırmacılar DKA türevlerini melez algoritmalarla 0-1 karışık tamsayı programlama (KTP) problemlerini çözmek için kullanmışlardır (bakınız örneğin Laziç v.d., 2010, Hanafi v.d. (2010), Zhao v.d., 2012).

DKA sezgiselini diğer sezgisel metotlarla birleştirerek DKA sezgiselinin yeni türleri oluşturulur. Perez v.d. (2003) DKA sezgiselini TA sezgiseliyle birleştirerek Değişken Komşuluk Tabu Arama (DKTA) metodunu geliştirdiler. Garcia-Lopez v.d. (2002) çözüm uzayını etkili şekilde aramak için DKA algoritmasının adımlarını mevcut işlemciler arasında dağıtarak Paralel DKA (PDKA) sezgiselini geliştirmişler ve bu sezgiseli TSP kütüphanesinin büyük boyutlu *p*-ortanca problemlerinde test etmişlerdir. Crainic v.d. (2004) birbirinden bağımsız birçok DKA sezgiselini aralarında elde edilen en iyi çözüm hakkında farklı zamanlarda bilgi alışverişi sağlayarak birlikte çalıştıran Kooperatif Paralel DKA (KPDKA) sezgiselini tanımlamışlardır. Bu metodu TSP kütüphanesinin 11948 düğümlü ve 1000 tesisli problem örneklerinde test etmişlerdir. Elde edilen sonuçlar kooperatif metodun önceki metotlara göre çözüm kalitesinden ödün vermeden daha az çözüm süresinde sonuçlar verdiğini göstermiştir.

Literatürde DKA sezgiseli ve türevleri birçok optimizasyon problemlerine uygulanmıştır. Şevkli ve Aydın (2006) iş çizelgeleme problemini çözmek için DKA sezgiselini önermişler ve önerilen diğer metotlardan daha iyi sonuçlar elde etmişlerdir. DKA sezgiselinin kullanıldığı diğer optimizasyon problemleri şunlardır: minimum liman yerleşim problemi (Hansen v.d., 2008), yerleşim yönlendirme problemi (Jarboui v.d., 2013), bant genişliği azaltma problemi (Mladenoviç v.d., 2010), kısıtlı ve kısıtsız sürekli optimizasyon problemleri (Mladenoviç v.d., 2008), uyumlu ortalama sınıflandırma problemi (Alguwaizani v.d., 2011), yerel dallanma problemi (Hansen v.d., 2006), *k*-elemanlı ağaç problemleri (Brimberg v.d., 2006; Uroseviç v.d., 2004) ve kapasiteli *p*-ortanca problemi (Fleszar ve Hindi, 2008).

2.4 Çok Amaçlı Karar Verme

Bu alt başlıkta çok amaçlı karar verme ilgili bir önbilgi sunulmuş ve daha sonra literatür özeti aktarılmıştır.

Çok Kriterli Karar Verme (ÇKKV) probleminin amacı, birçok kriterden oluşan belirli ya da belirsiz olarak tanımlanmış alternatifler kümesi içindeki en çok tercih edilen alternatifi bulmak ya da bu alternatifleri sıralamaktır. Kısıtlar kullanılarak alternatiflerin belirsiz olarak tanımlandığı problemler çok kriterli tasarlama problemleri, alternatiflerin belirli olarak verildiği problemler çok kriterli hesaplama problemleri olarak adlandırılır (Dehnokhalaji v.d., 2011).

p kriterli Çok Kriterli Karışık Tamsayı Programı (ÇKKTP) şu şekilde tanımlanır (Özpeynirci ve Köksalan, 2010) :

$$\text{"min"} \ Cx = \left(f_1(x), f_2(x), \dots, f_p(x)\right)$$
$$s.t \ \ x \in X,$$

Burada $X = \{Ax \leq b, x \geq 0, x = (x', x''), x' \in \mathbb{R}^{n'}, x'' \in \mathbb{Z}^{n''}\}$ bütün mümkün çözümlerin kümesi ve $n = n' + n''$. $A, m \times n$ boyutunda bir matris, $A \in \mathbb{R}^{m \times n}$, ve $b \in \mathbb{R}^m$. Reel değer alan n' adet ve tamsayı değer alan n'' adet karar değişkeni vardır. En azlanması gereken p kriter vardır. C, $p \times n$ boyutunda bir matris ve C matrisinin q. satırı q. amaç fonksiyonuna karşılık gelmektedir, $f_q(x)$. Vektör en azlama iyi tanımlı bir matematik operatörü olmadığı için tırnak işareti kullanılmıştır.

$y = Cx$ koşulunu sağlayan $y = \left(y_1, y_2, \dots, y_p\right) \in \mathbb{R}^p$ noktası, $x \in X$ çözümünün çıktısı olarak adlandırılır. X ve $Y = \{y \in \mathbb{R}^p : y = Cx, \ x \in X\}$ sırasıyla karar uzayı ve amaç (kriter) uzayı olarak adlandırılır. Bütün amaçları aynı anda en azlayan bir $y \in Y$ noktasının olmadığı varsayılmaktadır. Ayrıca, Y kümesinin sınırlı (bounded) olduğu ve $f_q(x) > 0, q = 1, \dots, p; \ \forall \ x \in X$ olduğu varsayılmaktadır.

$y, y' \in Y$, olsun. y' noktası y noktasını baskılar (dominate) ancak ve ancak $y'_q \leq y_q, \forall \ q$ ve en az bir q değeri için $y'_q < y_q$ ise. Eğer $y'_q < y_q, \forall q$ ise, y' kesinlikle y çözümünü baskılar (strictly dominate) denir. Eğer y çözmünü baskılayan bir $y' \in Y$ çözümü yoksa y çözümü baskın (nondominated) olarak adlandırılır. y noktası az baskın (weakly nondominated) olarak adlandırılır ancak ve ancak $y'_q < y_q, \forall q$ olacak şekilde bir $y' \in Y$ noktası yoksa. Az baskın noktalar kümesi, bütün baskın noktaları ve bazı özel baskılanan (dominated) noktaları içerir. Vektör gösteriminde $y' \leq y, \ y'_q \leq y_q, \forall \ q$ 'ya; $y' < y, \ y'_q < y_q, \forall q$ 'ya ve $y' = y, \ \ y'_q = y_q, \forall q$ 'ya karşılık gelmektedir. Bu tanımlara dayanarak, eğer $y' \leq y, y' \neq y$ ise y' noktası y noktasını baskılar ve eğer $y' < y$ ise y' noktası y noktasını kesinlikle baskılar.

Y_{ND}, baskın noktaların kümesi olsun. Verilen bir $y \in Y_{ND}$ noktası için, y^{conv} noktası y haricindeki baskın noktaların konveks birleşimi olsun. Özpeynirci ve Köksalan (2010) üç farklı baskın nokta türü tanımlamışlardır. Bir $y \in Y_{ND}$ noktası

- uç destekli baskın (*extreme supported nondominated* (ESN)) nokta olarak adlandırılır ancak ve ancak $y^{conv} \leq y$ olacak şekilde bir y^{conv} noktası yoksa,
- normal destekli baskın (*nonextreme supported nondominated* (NSN)) nokta olarak adlandırılır ancak ve ancak $y^{conv} < y$ olacak şekilde bir y^{conv} noktası yoksa, fakat $y^{conv} = y$ olacak şekilde bir y^{conv} noktası varsa,
- desteksiz baskın (*unsupported nondominated)* nokta olarak adlandırılır ancak ve ancak $y^{conv} < y$ olacak şekilde bir y^{conv} noktası varsa.

Baskınlık (dominance) ve etkinlik (efficiency) sırasıyla amaç ve karar uzaylarında birbirinin karşılığıdır. $x \in X$ bir çözüm ve $y = Cx$ bu çözümün amaç uzayındaki karşılığı olsun. x çözümü etkindir ancak ve ancak y çözümü baskınsa ve x çözümü etkisizdir ancak ve ancak y çözümü baskılanansa. x çözümü az etkindir ancak ve ancak y çözümü az baskınsa. Uç destekli etkin, normal destekli etkin ve desteksiz etkin çözümler benzer şekilde tanımlanır.

Literatürde, DKA sezgiseli birkaç ÇKKV problemlerine uygulanmıştır. Liang ve Lo (2010) üç farklı çok amaçlı artıklık yerleştirme problemi (redundancy allocation problem) için çok amaçlı DKA sezgiselini önermişlerdir. Bu çalışmada önerilen DKA sezgiseli tek bir alternatifle başlar ve bu alternatif çözümün etrafında rastgele seçilecek bir komşulukta yeni alternatifler bularak yerel en uygun alternatifi bulur. Yerel en uygun alternatifi ararken mevcut alternatiften daha iyi bir alternatif bulursa mevcut alternatifi değiştirir ve bu alternatifin baskıladığı iyi olmayan alternatifleri eleyerek Pareto yüzeyini (Pareto front) günceller. Eğer seçilen komşulukta baskın bir alternatif bulunamazsa tekrar rastgele bir komşuluk seçilerek aramaya devam edilir. Bu sezgisel, belirlenen bir durdurma koşulu sağlanıncaya kadar devam eder ve durduğunda baskın olan çözümlerin oluşturduğu Pareto yüzeyi raporlanır.

Gagne v.d. (2005) çok amaçlı çizelgeleme problemleri için akıllı sezgiselleri kullanarak uzlaşık çözümleri (compromise solutions) bulmak amacıyla genel bir yaklaşım önermişlerdir. Bu yaklaşımı iki amaçlı bir çizelgeleme problemine yeni bir melez tabu arama/DKA sezgiseli kullanarak uygulamışlardır. Bu tabu/DKA sezgiseliyle oluşturdukları çözümlerin bilinen referans kümelerine yaklaştığını göstermişlerdir. Arroyo v.d. (2011) DKA sezgiseline dayanan üç adet çok amaçlı algoritmayı karşılaştırmışlardır. Bu algoritmalar genel olarak Liang ve Lo (2010) 'un çalışmalarında kullandığı yönteme benzemektedir, ancak bu algoritmaların iki tanesinin sallama (shaking) adımında rastgele bir alternatif seçerken amaçlara ağırlık vererek aramanın bir

noktaya yoğunlaşması (intensification) ya da bir noktadan uzaklaşması (diversification) sağlanmıştır. Bu algoritmaları tek makine çizelgeleme problemi türü üzerinde uygulamışlar ve yoğunlaşma yöntemiyle daha iyi kalitede çözümler elde etmişlerdir.

Liang v.d. (2013) iki amaçlı özdeş paralel makine çizelgeleme problemi için DKA ve çoklu karınca kolonisi optimizasyonu algoritmalarını geliştirmişlerdir. Bu çalışmada kullanılan DKA sezgiseli Liang ve Lo (2010) 'da önerilen sezgisele dayanmaktadır ve test sonuçlarına göre DKA sezgiseli karşılaştırılan diğer bütün sezgisellerden daha iyi sonuç vermiştir. Liang ve Chuang (2013) çok-amaçlı kaynak yerleştirme problemleri için Liang ve Lo (2010) 'da önerilen DKA sezgiseline dayanan üç tane DKA sezgiseli önermişlerdir. Geliştirilen sezgiseller, temel çözüm seçme yöntemleri ve baskın çözümleri sıfırlama yöntemleri bakımından farklılık göstermektedir. Elde edilen sonuçlar, geliştirilen DKA sezgisellerinin literatürde önerilen metotlardan daha iyi sonuçlar verdiğini göstermiştir.

DKA sezgiseli diğer akıllı sezgisellerle birlikte kullanılarak çok amaçlı çeşitli problemlerin çözümü için kullanılmıştır. Li v.d. (2010), genetik arama (GA) sezgiselini DKA sezgiseli ile birleştirerek çok amaçlı esnek makine çizelgeleme problemleri için yeni bir melez DKA algoritması önermişlerdir. Onety v.d. (2013) çalışmalarında, IP rotalama problemi için DKA sezgiselinin yerel arama adımını çok amaçlı GA sezgiseliyle birlikte kullanarak Değişken Komşuluk Çok Amaçlı GA sezgiselini geliştirmişlerdir. Ripon v.d. (2013), düzensiz alanlı çok amaçlı tesis yerleşim problemi için DKA sezgiseline dayanan evrimsel algoritma geliştirmişlerdir.

ÇKKV literatüründe yapılan çalışmaların bir kısmı, amaç fonksiyonlarını her bir amaca bir ağırlık vererek tek bir amaç fonksiyonunda toplamıştır. Daha sonra elde edilen problemler, tek amaçlı problem gibi çözülmüş ve en iyi çözüme ulaşmak amaçlanmıştır. DKA sezgiselinin uygulandığı bu tür problemlere Adibi v.d. (2010) (çok amaçlı dinamik makine çizelgeleme), Eskandarpour v.d. (2013) (satış sonrası ağ tasarım problemi), Abedzadeh v.d. (2013) (çok amaçlı dinamik tesis yerleşim problemi) ve Eskandarpour v.d. (2013a) (satış sonrası ağ tasarım problemi) örnek olarak gösterilebilir.

3. GEREÇ VE YÖNTEM

Bu bölümde proje kapsamında yaptığımız çalışmaları açıkladık. Bu çalışmalar; (*i*) eczane nöbet çizelgeleme problemlerinin tanımlanması, (*ii*) nöbet bölgelerinin güncellenmesi, (*iii*) alt sınır, (*iv*) dal-sınır, (*v*) dal-fiyat, (*vi*) Tabu arama, (*vii*) değişken komşuluk arama algoritmalarının geliştirilmesi, (*viii*) iki amaçlı problemin tanımlanması ve (*xi*) iki amaçlı değişken komşuluk arama algoritmalarının geliştirilmesini kapsar.

3.1 Eczane Nöbet Çizelgeleme Problemleri

Bu alt başlıkta Eczane Nöbet Çizelgeleme (ENÇ) problemlerini tanımladık. Öncelikle İzmir'de kullanılmakta olan mevcut sistemi aktardık. Daha sonra basit ve gerçekçi modelleri tanımladık.

ENÇ problemi için geliştirilen farklı matematik programlama modelleri sunduk. Bu modelleri basit ve gerçekçi olarak iki başlık altında inceledik. Basit modellerde her eczane planlama periyodu boyunca bir defa nöbet tutar. Gerçekçi modellerde ise eczaneler birden fazla nöbet tutabilir. Sunulan modellerin tamamı karışık tam sayılı matematiksel programlama modelleridir.

Planlama ufku geliştirilen modellerin çoğu için bir girdi parametresidir ve bu parametrenin karar verici tarafından belirlendiğini varsaydık. İkinci basit modelde, planlama ufkunu modelin karar değişkenlerinden bir tanesi olarak belirledik. Bu model planlama ufkunu en büyüklemeyi amaçlar.

3.1.1 Mevcut Sistem

Bu bölümde Türkiye'deki ve İzmir'deki eczane nöbet sistemleri hakkında bilgiler verdik. Bu bölümün hazırlanması için İzmir Eczacılar Odası ile çeşitli görüşmeler yaptık.

Türkiye'de her eczane Türk Eczacıları Birliği'ne kayıtlıdır. Bu birliğin 53 bölge odası bulunmaktadır ve İzmir 3. Bölge içindedir. İzmir Büyükşehir Belediyesi içerisinde yaklaşık 1300 eczane vardır ve 3. Bölge Eczacılar Odası bu eczanelerin nöbet çizelgelerinden sorumludur.

Mevcut planlama sisteminde 3 farklı tip nöbet vardır. Bunlar ulusal tatillerde, hafta sonları ve hafta içi gecelerinde tutulan nöbetlerdir. İzmir'de Eylül ile Mayıs ayları arasında (talebin yılın

diğer zamanlarına göre daha fazla olması sebebi ile) Cumartesi günü hafta içi gün sayılmakta ve eczaneler bu gün çalışma saatlerinde açık olmaktadır. Talebin az olduğu yaz aylarında Cumartesi günü hafta sonu olarak değerlendirilir ve bugün sadece nöbetçi eczaneler açık olur. Her tip nöbet için eczanelerin nöbet çizelgeleme kuralı aynı olmasına karşı nöbetlerin sıralanması farklılık gösterir. Ulusal tatillerde tutulan nöbetler eczacıların en az tercih ettiği nöbet olduğundan, bu tip için özel bir sıralama yöntemi kullanılır.

İzmir kent merkezinde 11 ilçe vardır: Balçova, Bayraklı, Bornova, Buca, Çiğli, Gaziemir, Güzelbahçe, Karabağlar, Karşıyaka, Konak ve Narlıdere. Bu ilçeler nöbetçi eczane çizelgeleme sürecinin kontrol ve planlanmasını basitleştirmek için bölgelere bölünmüştür. Kent merkezinde toplam 45 bölge vardır.

Nöbet planlamada görev yapmak için her bölgeden bir temsilci eczacı seçilir. Bu temsilci bir sonraki planlama periyodu için kendi bölgesinin nöbet çizelgesini hazırlar ve Eczacılar Odası'na önerir. Eczacılar Odası'nın nöbetçi eczane komisyonu tüm nöbet bölgelerinden gelen çizelgeleri kontrol eder ve gerekli değişiklikleri yapar. İzmir İl Sağlık Müdürlüğü çizelgeye son onayı verir.

İzmir'de her bir planlama periyodunun uzunluğu 4 ay olarak belirlenmiştir. Dolayısıyla yılda 3 periyod vardır. Ancak bu uzunluk iller arasında farklılık gösterebilir. Örneğin İstanbul'da 1. Bölge Eczacı Odası planlama periyodunu 1 yıl olarak belirlemiştir. Ankara'da da aynı durum söz konusudur. Bir yıl içinde olabilecek değişiklikleri kontrol edebilmek zor olduğundan kısa planlama periyotları daha esnektir. Bir yıl içerisinde yeni eczanelerin açılması, bazılarının kapanması, yer değişimleri ya da öngörülmeyen hastalıkların olması gibi birçok değişiklik söz konusu olabilir. Bu sebeple kısa planlama periyotları Odanın çizelge üstündeki kontrolünü arttırmaktadır. Bununla birlikte nöbet çizelgeleme sürecinin Oda için zaman alıcı ve zor bir iş olmasından dolayı çok kısa periyotlar da tercih edilmemektedir.

İzmir Eczacı Odası nöbet planlamasını halen elle yapmaktadır. Kullanılan yöntem şöyle açıklanabilir: Her bölgede nöbet tutabilecek eczaneler lisans numaralarına göre bir listede sıralanır. Aynı liste 3 tip nöbet (hafta içi gece, hafta sonu ve ulusal tatil) için de kullanılır. Gelecek nöbetler bu nöbetçi eczane listesindeki sıraya göre atanır. Periyodun ilk nöbeti, bir önceki periyotta son nöbetçi eczaneyi takip eden eczaneye verilir. Böylelikle gelecekteki tüm nöbetler listedeki sırayı takip eden eczanelere atanır. Listenin sonuna gelindiğinde atamaya listenin

başından devam edilir. Bu bir planlama periyodundaki tüm nöbetlerin her bölgeden bir eczaneye atanmasına kadar devam eder. En son nöbet atanan eczaneyi takip eden eczane bir sonraki periyotta ilk nöbeti alır. Bu yöntem her bölge ve her nöbet tipi için ayrı ayrı uygulanır.

İzmir kent merkezindeki eczanelerin sayısı, nöbet tutan eczanelerin sayısından farklıdır. Çünkü bazı eczaneler nöbet tutmamaktadır. Bunun iki sebebi vardır; ilki bazı eczanelerin iş merkezlerinde yer alması ve yerleşim bölgeleri ile sağlık bölgelerine uzak olmasıdır. İkinci sebep ise bazı eczacıların özel durumlarıdır. Bu durumlar eczacının nöbet tutamayacak kadar yaşlı veya hasta olmasıdır. Bu eczacılar kalıcı veya geçici olarak nöbet tutmaktan muaf sayılabilirler.
Yukarıda bahsedilen sebeplerden dolayı nöbet tutabilecek eczane sayısı ve bu sayının bölgelere dağılımı bir periyottan diğerine değişebilmektedir. Her planlama periyodunun başlamasına 1 ay kala o periyotta nöbet tutacak eczaneler belirlenir.

İzmir kent merkezinde yaklaşık 1300 eczane vardır fakat 2010 yılının 3. periyodunda bahsedilen sebeplerden dolayı yalnızca 1046 eczane nöbet tutmuştur. Bu sayı 2011 yılının ilk periyodunda 1053 eczaneye ulaşmıştır.

Tablo 1. Örnek eczane listesi

Eczane		**Aylar**			
Lisans Numarası	**İsmi**	**Eylül**	**Ekim**	**Kasım**	**Aralık**
230	Güçlü	30		13	5H 31
253	M.şehir Sağlık		1	15	12H
262	Bostanlı Pınar		2	20	19H
305	Damla		7	22	26H
310	Bostanlı Saadet		5	23	
...	...				
507	Site		16 29U		4
Listenin Sonu					
10	Çaylı	1	18		6

Tablo 1'de, 2010 yılının 3. periyodunda Bostanlı bölgesi için bir nöbetçi eczane çizelge örneği görülmektedir. Tablodaki sayılar günleri temsil etmektedir. *H* hafta sonu, *U* ulusal tatillerdeki nöbetleri göstermektedir. Geri kalan tüm nöbetler hafta içi gece nöbetleridir. 2010 yılının 3. periyodu için Pazar günü talep fazlalığından dolayı hafta içi bir gün olarak düşünülmüştür. 3. periyodun son günü 31 Aralık'tır ve son nöbet Güçlü Eczanesi'ne atanmıştır. Bu da 2011 yılının 1. periyodunun ilk gece nöbetinin Mavişehir Sağlık Eczanesi'ne atanacağı anlamına gelir. Aynı şekilde, son Pazar nöbeti olan 26 Aralık'ta Damla Eczanesi nöbet tutmuştur. Bu sebeple gelecek periyodun ilk Pazar nöbeti Bostanlı Saadet Eczanesi'ne atanacaktır. Aynı kural ulusal tatil nöbetleri için de geçerlidir. Son ulusal tatil nöbeti olan 29 Ekim günü Site Eczanesi nöbet tuttuğundan gelecek periyotta ilk ulusal tatil nöbeti (1 Ocak) Çaylı Eczanesi'ne atanacaktır.

Tabloda görüldüğü gibi, Ekim ayında Damla ve Bostanlı Saadet eczanelerinin nöbetleri, eczanelerin nöbet tarihlerini karşılıklı olarak değiştirmelerinden dolayı verilen sıraya göre atanmamıştır. Bu durum Oda tarafından onaylanmak zorundadır.

Eczacı Odası yetkilileri, hafta sonu ve resmi tatiller için hali hazırda kullanılmakta olan yöntemin devam etmesinin uygun olacağını, hafta içi nöbetler için ise farklı bir çizelgeleme yönteminin kabul görebileceğini belirttiler.

Nöbet çizelgelemenin asıl amacı ilaca gereksinimi olan müşterilerin planlama periyotlarının bütün günlerinde eczaneye kolaylıkla ulaşabileceği şekilde eczane nöbetlerinin yerleştirilmesidir. Mevcut sistemin en büyük problemi atamaların merkezi olarak değil, belirli bölgeler için yapılmasıdır. Bu atama yönteminde kimi zaman farklı bölgelerde bulunan fakat birbirlerine çok yakın veya çok uzak eczanelerin aynı gün nöbet tutması söz konusu olabilmektedir. Her bölgede her gün yalnızca bir eczane nöbet tutabildiğinden bu tip bir atama müşterilerin eczane bulabilmek için çok uzun mesafeler kat etmelerine yol açmaktadır.

3.1.2 Basit Matematiksel Programlama Modelleri

İki farklı basit model geliştirdik. Bu modeller gerçek hayat problemin özelliklerinin bir kısmını yansıtırlar, bu sebeple basit olarak adlandırdık. Bu modellerdeki eczaneler planlama ufku içerisinde bir kez nöbet tutar. Modelleri tek nöbet modelleri olarak da adlandırabiliriz. Bu bölümdeki ilk model müşterilerin talep ağırlıklı kat ettikleri mesafeyi en azlamayı amaçlar. İkinci

model ise belirlenmiş bir servis seviyesini göz önünde bulundurarak planlama ufkunu en çoklamayı amaçlar.

Basit Model 1

Basit Model 1 (BM1) müşterilerin talep ağırlıklı kat ettikleri mesafeyi en azlamayı amaçlamaktadır. Bu modelde, her *t* gününde, her *i* müşterisi en az bir *j* eczanesine atanır. Bu modelde *T* gün için planlama yapılmaktadır ve *I* adet müşteri ve *J* adet eczane vardır.

BM1 modelinde kullandığımız kümeleri, girdileri, karar değişkenlerini, amaç fonksiyonunu ve kısıtları aşağıda sunduk:

Kümeler ve girdiler:

i : talep düğümleri (müşteriler), $i \in \{1,...,I\}$

j : tesis düğümleri (eczaneler), $j \in \{1,...,J\}$

t : planlama periyotları (günler), $t \in \{1,...,T\}$

h_i : *i* düğümünün talep miktarı

d_{ij} : *i* talep düğümü ve *j* tesis düğümü arasındaki mesafe,

Karar değişkenleri:

$$y_{jt} = \begin{cases} 1 & \text{eğer } j \text{ tesisi } t \text{ gününde açılırsa} \\ 0 & diğer \end{cases}$$

$$x_{ijt} = \begin{cases} 1 & \text{eğer } i \text{ müşterisi } j \text{ tesisine } t \text{ gününde atanmış ise} \\ 0 & diğer \end{cases}$$

Amaç fonksiyonu:

$$Min\ F = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{t=1}^{T} h_i d_{ij} x_{ijt} \qquad \text{(BM1-1)}$$

Bu fonksiyon toplam kat edilen mesafeyi talep ağırlıklı olarak en azlamayı amaçlar.

Kısıtlar:

$$x_{ijt} \leq y_{jt}\ \forall i,j,t \qquad \text{(BM1-2)}$$

Müşteri *i* eczane *j*'ye *t* gününde ancak *j* eczanesi *t* gününde açılmış ise atanabilir.

$$\sum_{t=1}^{T} y_{jt} = 1 \; \forall j \qquad \text{(BM1-3)}$$

Her *j* eczanesi planlama süresince 1 kez nöbet tutmalıdır.

$$\sum_{j=1}^{J} x_{ijt} = 1 \; \forall i, t \qquad \text{(BM1-4)}$$

Her müşteri her gün bir eczaneye atanmalıdır.

$$y_{jt} \in \{0,1\} \; \forall j, t \qquad \text{(BM1-5)}$$

Eczanelere nöbet atama kararı ikili değişken ile gösterilir.

$$x_{ijt} \in \{0,1\} \; \forall i, j, t \qquad \text{(BM1-6)}$$

Müşterilerin eczanelere atanması kararı ikili değişken ile gösterilir.

Modelin ve amaç fonksiyonunun yapısı dikkate alındığında (BM1-6)'da ikili olarak tanımlanan x_{ijt} karar değişkeninin sürekli ve pozitif tanımlanmasının yeterli olacağı görülür. Çünkü en iyi çözümde, (eczanelerin hizmet verecekleri müşteri sayısında bir kısıt olmadığı için) her müşteri kendisine en yakın nöbetçi eczaneye atanacaktır.

Ayrıca (BM1-3) kısıtındaki eşitlik küçük eşit olarak değiştirilebilir. Çünkü optimal sonuçta bir eczaneye nöbet atanmamış ise bu eczaneye nöbet atanarak amaç fonksiyon değeri iyileştirilemez. Eğer bu eczane açık iken bir müşteriye en yakın eczane oluyor ise bu müşteri o eczaneye atanır ve amaç fonksiyon değeri iyileşir ama bu durumda mevcut çözümün optimal çözüm olmaması gerekir.

Benzer şekilde (BM1-4) kısıtı büyük eşit olarak değiştirilebilir. Çünkü bir müşterinin bir günde birden fazla eczaneye atanması amaç fonksiyon değerini azaltmaz.

Basit Model 2

İkinci basit model (BM2) müşterilere sağlanan servis seviyesini kısıt olarak alıp planlama ufkunu en uzun yapmayı amaçlar. Bu amaçla BM1 modelindeki değişkenlere ek olarak yeni bir ikili karar değişkeni olan g_t tanımlanmıştır:

$$g_t = \begin{cases} 1 & \text{eğer } t \text{ gününde hizmet veriliyor ise} \\ 0 & diğer \end{cases}$$

BM2 modeli BM1 modelindeki girdi, parametre ve karar değişkenlerini kullanır. BM2 modeli müşterilere belirli bir servis seviyesinde hizmet etmeyi amaçlar. Bu sebeple bir müşterinin kendisine kritik uzaklıktan (*DT*) daha uzak eczanelerden hizmet alamayacağını varsaydık. Kritik uzaklık parametresi *DT* model için bir girdidir. Bu girdi kullanılarak aşağıdaki parametreyi tanımladık:

$$\alpha_{ij} = \begin{cases} 1 & \text{eğer } d_{ij} \leq DT \\ 0 & diğer \end{cases}$$

BM2 modelinin amaç fonksiyonu:

$$Max\ G = \sum_{t=1}^{T} g_t \qquad \text{(BM2-1)}$$

Bu fonksiyon hizmet verilen günlerin adedini en çoklamayı amaçlar.

Kısıtlar:

$$g_{t+1} \leq g_t\ \forall t/\{T\} \qquad \text{(BM2-2)}$$

Bu kısıt, *t*+1 gününde hizmet verilebilmesi için *t* gününde de hizmet veriliyor olmasını sağlar.

$$g_t \leq \sum_{j=1}^{J} y_{jt}\ \forall t \qquad \text{(BM2-3)}$$

Eğer *t* gününde hizmet veriliyor ise en az bir eczanenin nöbetçi olarak atanması gerekir.

$$\sum_{j=1}^{J} \alpha_{ij} x_{ijt} \geq g_t \quad \forall i,t \qquad \text{(BM2-4)}$$

Eğer *t* gününde hizmet veriliyor ise her *i* müşterisi *t* gününde kendisine *DT* birimden uzak olmayan bir eczaneye atanmalıdır.

$$g_t \in \{0,1\} \; \forall t \qquad \text{(BM2-5)}$$

Bir günde hizmet verme kararı ikili değişken ile gösterilir.

Bu modelde ayrıca (BM1-2), (BM1-3), (BM1-5) ve (BM1-6) kısıtları vardır.

Bu bölümde geliştirilen matematiksel programlama modelleri problemi daha iyi anlamak ve alternatif gerçekçi modeller geliştirmek için faydalı oldular. Her eczanenin planlama süresince bir defa nöbet tutması kısıtı bu modellerin önemli bir kabulüdür. Gerçekçi problemler için bu kısıt oldukça bağlayıcıdır ve bir sonraki bölümde bahsedilecek gerçekçi modellerde bu kabulü gevşettik.

3.1.3 Gerçekçi Matematiksel Programlama Modelleri

Bu bölümde geliştirilen modellerde gerçek hayatın daha iyi yansıtılmasını amaçladık. Eczacı Odası yetkilileri ile yaptığımız görüşmelerle elde ettiğimiz bilgileri, bu modelleri geliştirirken göz önünde bulundurduk. Gerçekçi modellerde eczaneler birden fazla defa nöbet tutabilirler. Bu sebeple bu modelleri çok nöbetli modeller olarak da adlandırabiliriz.

İlk gerçekçi modelde, Mevcut Sistem bölümünde bahsettiğimiz nöbet bölgeleri yaklaşımını kullandık. İkinci gerçekçi model ise bu yaklaşımı kullanmamakta, ancak her eczaneye çevresindeki eczaneler ile benzer sayıda nöbet atamaktadır. Her iki modelde de amaç müşterilerin planlama ufku boyunca kat etmeleri gereken talep ağırlıklı yolu en azlamaktır.

Gerçekçi Model 1

Gerçekçi Model 1 (GM1) mevcut sistemde de kullanılmakta olan nöbet bölgesi yaklaşımını kullanır. Bir eczanenin tutacağı nöbet sayısı içinde bulunduğu bölgedeki eczanelerin sayısı ile ilişkilidir. Çok fazla (az) sayıda eczane olan nöbet bölgelerinde her eczane az (çok) sayıda nöbet tutar. Bu sebeple farklı bölgelerdeki eczaneler farklı sayıda nöbet tutabilirler. Bir nöbet bölgesindeki ortalama nöbet sayısı planlama periyodundaki gün sayısının o bölgedeki eczane

sayısına bölünmesi ile bulunur. Bu sayının alta ve üste yuvarlanması ile o bölgedeki her eczanenin tutması gereken nöbet sayısının alt ve üst sınırları bulunur.

Basit modellerde kullanılan küme, girdi, ve karar değişkenlerine ek olarak GM1 aşağıdaki küme ve girdileri kullanır:

k : bölgeler, $k \in \{1,...,K\}$

J_k : k bölgesindeki eczaneler kümesi

r_j : j eczanesinin bölgesi,

Her eczanenin sadece bir bölgesi vardır ve nöbet bölgeleri aşağıdaki özelliği sağlar:

$$J = \bigcup_{k=1}^{K} J_k$$

Amaç fonksiyonu:

$$Min\ F = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{t=1}^{T} h_i d_{ij} x_{ijt} \qquad \text{(GM1-1)}$$

Bu fonksiyon toplam kat edilen mesafeyi talep ağırlıklı olarak en azlamayı amaçlar.

Kısıtlar:

$$x_{ijt} \leq y_{jt}\ \forall i,j,t \qquad \text{(GM1-2)}$$

Müşteri i eczane j'ye t gününde ancak j eczanesi t gününde açılmış ise atanabilir.

$$\sum_{t=1}^{T} y_{jt} \leq \left\lceil \frac{T}{|J_k|} \right\rceil\ \forall j, k = r_j \qquad \text{(GM1-3)}$$

Her j eczanesi planlama süresince en fazla kendi nöbet bölgesinin nöbet üst limiti kadar nöbet tutabilir.

$$\sum_{t=1}^{T} y_{jt} \geq \left\lfloor \frac{T}{|J_k|} \right\rfloor\ \forall j, k = r_j \qquad \text{(GM1-4)}$$

Her j eczanesi planlama süresince en az kendi nöbet bölgesinin nöbet alt limiti kadar nöbet tutmalıdır.

$$\sum_{\{j:r_j=k\}} y_{jt} = 1 \ \forall k,t \qquad \text{(GM1-5)}$$

Her gün her bölgeden bir eczaneye nöbet atanmalıdır.

$$\sum_{j=1}^{J} x_{ijt} = 1 \ \forall i,t \qquad \text{(GM1-6)}$$

Her müşteri her gün bir eczaneye atanmalıdır.

$$y_{jt} \in \{0,1\} \ \forall j,t \qquad \text{(GM1-7)}$$

Eczanelere nöbet atama kararı ikili değişken ile gösterilir.

$$x_{ijt} \in \{0,1\} \ \forall i,j,t \qquad \text{(GM1-8)}$$

Müşterilerin eczanelere atanması kararı ikili değişken ile gösterilir.

Gerçekçi Model 2

Eczacılar Odası ile yapılan görüşmelerde mevcut sistemde kullanılmakta olan nöbet bölgesi yaklaşımı üzerinde durduk. Bu yaklaşımın en önemli özelliklerinden birisi, birbirine yakın eczanelerin (aynı nöbet bölgesine yer alması ve) benzer sayıda nöbet tutmalarını sağlamasıdır. Oda yetkilileri, nöbet sayılarının adil dağılımını sağladığı için bu yaklaşımı benimsediklerini ifade ettiler.

Nüfus yoğunluğunun şehrin farklı bölgelerinde zaman içerisinde değişiklik göstermesi, eczanelerin kapanması, yenilerinin açılması veya yer değiştirmesi gibi sebepler ile mevcut nöbet bölgelerinin tekrar değerlendirilme ihtiyacı doğar. Ancak nöbet bölgelerinin güncellenmesi düzenli olarak yapılmamaktadır.

Gerçekçi Model 2 (GM2) nöbet bölgeleri olmaksızın nöbet bölgelerinin sağladığı adil nöbet dağılımını sağlamayı hedefler. GM2 her eczanenin nöbet sayısının kendisine yakın eczanelerin nöbet sayılarının ortalamasına benzer olmasını hedefler.

GM2 ve GM1 modelinin kullandığı girdiler, kümeler ve karar değişkenleri çok benzerdir. GM2 GM1'de tanımlanan nöbet bölgeleri ile ilgili verileri kullanmaz. GM2'nin ihtiyaç duyduğu ek girdi ve kümeler şunlardır:

j,m : eczaneler, $j,m \in \{1,...,J\}$

c_{jm} : j ve m eczaneleri arasındaki mesafe,

α : iki eczanenin komşu olabileceği en uzun mesafe

S_j : j eczanesinin α-komşuluğundaki eczaneler kümesi

$$S_j = \bigcup_{m:c_{jm} \leq \alpha, m \neq j} \{m\}$$

β : Bir eczanenin nöbet sayısı ile α-komşuluğundaki eczanelerin ortalama nöbet sayısı arasındaki farkın alt ve üst sınırı

GM2 modelinin amaç fonksiyonu:

$$Min\ F = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{t=1}^{T} h_i d_{ij} x_{ijt} \qquad \text{(GM2-1)}$$

Bu fonksiyon toplam kat edilen mesafeyi talep ağırlıklı olarak en azlamayı amaçlamaktadır.

Kısıtlar:

$$\sum_{t=1}^{T} y_{jt} \leq \beta + \frac{1}{|S_j|} \sum_{t=1}^{T} \sum_{m \in S_j} y_{mt} \ \forall j \qquad \text{(GM2-2)}$$

Her j eczanesi planlama süresince en fazla α-komşuluğundaki eczanelerin ortalama nöbet sayısından β adet fazla nöbet tutabilir.

$$\sum_{t=1}^{T} y_{jt} \geq -\beta + \frac{1}{|S_j|} \sum_{t=1}^{T} \sum_{m \in S_j} y_{mt} \ \forall j \qquad \text{(GM2-3)}$$

Her j eczanesi planlama süresince en az α-komşuluğundaki eczanelerin ortalama nöbet sayısından β adet az nöbet tutmalıdır.

GM2 modeli ayrıca GM1-2, GM1-6, GM1-7 ve GM1-8 kısıtlarını içerir.

Gerçekçi modeller mevcut halleri ile daha önceki planlama dönemlerini dikkate almazlar. Bu sebeple bazı eczanelere diğerlerinden daha fazla nöbet atanabilir. Uzun vadede bu bir sorun oluşturacaktır. Bu sorunu önlemek için nöbet alt ve üst sınırlarının güncellenmesi yeterlidir. Bir önceki nöbet döneminde daha fazla nöbet tutan eczanelerin üst sınırları alt sınırlarına eşitlenir. Bu şekilde GM1 için aynı bölgedeki eczaneler uzun vadede aynı miktarda nöbet tutarlar. Benzer

güncelleme GM2'de her eczanenin α-komşuluğundaki eczaneler ile aynı miktarda nöbet tutmasını sağlar.

Bu bölümde iki basit ve iki de gerçekçi olmak üzere dört model geliştirdik. Planlama ufkunu en uzun yapmayı amaçlayan BM2 modelini ve Nöbet bölgesi olmaksızın birden fazla nöbet ataması yapan GM2 modelini ilerleyen bölümlerde ele almayacağız. BM2 modelini Eczacı Odası ile yapılan görüşmelerde planlama ufkunun bir sabit olarak değerlendirilmesi gerektiğine karar verdiğimiz için eledik. GM2 modeli ise modelde yer alan α ve β değerlerinin belirlenmesinin zor olması ve Eczacı Odası'nın nöbet bölgelerinin devamı yönünde fikir bildirmesi sebebi ile eledik. BM1 modeli deneyler esnasında problem girdilerinin çözüm süresine etkisini gözlemlemek amacı ile kullanılmıştır.

3.1.4 Hesaplama Karmaşıklıkları

Bu bölümde GM1 probleminin değişken bölge sayısı için *NP*-Zor olduğunu ispatladık. Ayrıca verilen bir nöbet ataması için müşterilerin eczanelere optimal atanmasının polinom zamanda yapılabileceğini gösterdik.

Teorem 1: GM1 Problemi değişken nöbet sayısı için *NP*-Zordur.

İspat: Planlama ufkunun T=1 gün olduğunu, p adet bölge ve her bölgede n adet eczane ($n>>p$) olduğunu varsayalım. Toplamda $J=np$ adet eczane vardır. İlk n eczane 1. bölgede, ikinci n eczane 2. bölgede ve $(k-1)n+1,...,kn$ aralığındaki eczaneler k. bölgededir. Toplam I adet müşteri vardır.

Müşteri i ile eczane j arasındaki uzaklık d_{ij} ile gösterilmektedir. Varsayalım ki, $d_{ij}>0$ eşitsizliği her $i=1,...,I$ ve $j=1,...,J$ için sağlanmaktadır. Ayrıca varsayalım ki $d_{ij}= d_{i(j+kn)}$ her $k=1,...,K$, $i=1,...,I$ ve $j=1,...,J$ için sağlanmaktadır. Basit bir ifade ile j, $j+n$, $j+2n,...,j+(p-1)n$ numaralı eczanelerin aynı yerde olduğu varsayılmıştır. Böylece n farklı tesis noktasında eczaneler bulunmaktadır ve her tesis noktasında, her bölgeden bir tane olmak üzere, p adet eczane vardır. Aynı noktada yer alan eczaneler tüm müşterilere eşit uzaklıktadır. Planlama ufku T=1 olduğu için her bölgeden sadece bir eczane nöbetçi olarak seçilecektir. GM1 problemi n tesis noktasından p adedi seçerek toplamda talep ağırlıklı kat edilen yolu en azlamaya çalışmaktadır. GM1 problemi p-ortanca (p-median) problemine denktir ve değişken p için NP-Zordur. ■

GM1 probleminin bir özel durumu *p*-ortanca problemine karşılık geldiği için bu problem değişken bölge sayısı (*p* değeri) için NP-Zordur. Ancak sabit bölge sayısı için GM1 probleminin hesaplama karmaşıklığı halen bilinmemektedir. Ayrıca BM1 ve GM2 problemlerinin hesaplama karmaşıklıkları bilinmemektedir.

Olurlu bir nöbet atamasının verilmesi durumunda (nöbet atamasının olurlu olduğunu varsaydığımız için probleme özgü kısıtların da sağlandığını varsayıyoruz), problem müşterilerin nöbetçi eczanelere atanması halini alır. Bu durumda, her dört problem için de her müşterinin her gün hangi eczaneye atanması gerektiğine karar vermek gerekir. Bu karar polinom zamanda verilebilir. BM1, GM1 ve GM2 problemlerinin amaç fonksiyonları talep ağırlıklı kat edilen yolu en azlamayı amaçlar ve optimal sonuç elde etmek için her müşterinin en yakın eczaneye atanması gerekir. BM2 probleminde ise amaç en uzun süre hizmet vermektir ve müşterilerin en yakın nöbetçi eczaneye atanmasına gerek yoktur, tanımlanan servis seviyesinde hizmet verebilecek nöbetçi eczanelerden birisine atanmaları yeterli olacaktır.

Teorem 2: Olurlu bir nöbet ataması var ise BM1, BM2, GM1 ve GM2 problemlerinde müşterilerin eczanelere optimal atanması $O(IJT)$ karmaşıklığındadır.

İspat: Eğer olurlu bir nöbet ataması var ise, y_{jt} karar değişkenlerinin değeri her j ve t için bilinir. Her müşteri her gün bir nöbetçi eczaneye atanmalıdır. Bir t gününde bir i müşterisinin kendisine en yakın nöbetçi eczane yerine daha uzak bir nöbetçi eczaneye atanması amaç fonksiyon değerini daha iyiye götürmez. Bu sebeple optimal sonuçta, her müşteri her gün kendisine en yakın nöbetçi eczaneye atanacaktır. Aşağıdaki algoritma her t günü için, i müşterisine en yakın j^* eczanesini belirlemekte ve atamayı yapmaktadır.

Algoritma 1. Optimal Atama Algoritması (OAA)

Her i =1 → I müşterisi için tekrarla

Her t =1→ T günü için tekrarla

$$j^* = \underset{j|y_{jt}=1}{\operatorname{argmin}}\{d_{ij}\}$$

$$x_{ij^*t} = 1$$

Tekrarla sonu (*t*)

Tekrarla sonu (*i*)

Bu algoritma I müşteri için T gün boyunca J eczane arasından en yakını seçecek ve atayacaktır. Bu işlem $O(IJT)$ adımda yapılır.

■

3.2 Nöbet Bölgelerinin Güncellenmesi

Daha önce de bahsedildiği üzere, nöbet bölgelerinin belirlenmesinden sonra meydana gelen çeşitli değişiklikler (şehrin nüfus yoğunluğunun bölgeler arasında değişim göstermesi, mevcut eczanelerin kapanması, yer değiştirmesi veya yeni eczanelerin açılması gibi) mevcut nöbet bölgelerinin güncellenmesi (NBG) ihtiyacını doğurur. Bu problemin çözümü için bir matematik model geliştirdik. Bu model verilen bir nöbet bölgesi sayısı için her bölgeye en az bir eczane atar ve aynı bölgeye atanan en uzak eczane ikilisi arasındaki mesafeyi en aza indirmeyi amaçlar.

Bu model aşağıdaki küme ve girdileri kullanmaktadır:

k : bölgeler, $k \in \{1,...,K\}$

j,m : eczaneler, $j,m \in \{1,...,J\}$

c_{jm} : j ve m eczaneleri arasındaki mesafe,

Karar değişkenleri:

$$z_{jk} = \begin{cases} 1 & \text{eğer } j \text{ eczanesi } k \text{ bölgesine atanırsa} \\ 0 & diğer \end{cases}$$

$u_k = k$ bölgesine atanmış eczanelerden birbirine en uzak olan ikilinin arasındaki mesafe

$u =$ en büyük u_k değeri

Amaç fonksiyonu:

$Min\ u$ (NBG-1)

Bu fonksiyon en büyük u_k değerini en aza indirmeyi amaçlamaktadır.

Kısıtlar:

$u \geq u_k\ \forall k$ (NBG-2)

u değişkeninin değeri hiçbir u_k değişkenin değerinden küçük olamaz.

$$\sum_{k=1}^{K} z_{jk} = 1\ \forall j$$ (NBG-3)

Her eczane sadece bir bölgeye atanmalıdır.

$u_k \geq c_{jm}\left[z_{jk} + z_{mk} - 1\right]\ \forall j, m, k$ (NBG-4)

u_k değişkenin değeri *j* ve *m* eczanelerinin *k* bölgesine atanması durumunda c_{jm} değerinden daha az olamaz. Her iki eczanenin birden *k* bölgesine atanması dışındaki tüm durumlarda bu kısıt bağlayıcı olmayacaktır.

$z_{jk} \in \{0,1\}\ \forall j, k$ (NBG-5)

Eczaneleri bölgelere atama kararı ikili değişken ile gösterilir.

NBG modeli bu hali ile *K* adetten daha az nöbet bölgesi oluşturabilir. Bu durum her bölgeye en az bir eczane atanması kısıtının modele eklenmesi ile çözülebilir.

$$\sum_{j=1}^{J} z_{jk} \geq 1\ \forall k$$ (NBG-6)

Her bölgeye en az bir eczane atanmalıdır.

3.3 Alt Sınır Algoritmaları

Problemlere önerilecek olurlu çözümlerin kalitelerini ölçebilmek için alt sınır algoritmaları geliştirdik. Bu bölümde BM1 ve BM2 problemleri için geliştirdiğimiz alt sınırları aktardık

BM1 Problemi için Alt Sınır

BM1 problemi için geliştirilen alt sınır algoritmasını aşağıda verdik. Bu algoritmadaki *LB* değeri BM1 probleminin amaç fonksiyon değerinin alt sınırıdır. *M* problem için büyük bir değerdir (örneğin en büyük d_{ij} değerinden daha büyük bir değer).

Algoritma 2. BM1Alt Algoritması

LB=0
Her *i*=1→*I* için tekrarla
    Her *t*=1→*T* için tekrarla
        enyakın=*M*
        Her *j*=1→*J* için tekrarla
            Eğer d_{ij}<*enyakın*
                enyakın=d_{ij}
                enyakınj=*j*
            Eğer sonu
        Tekrarla sonu (*j*)
        LB=*LB*+ *enyakın* X h_i
        $d_{i,enyakınj}$=*M*
    Tekrarla sonu (*t*)
Tekrarla sonu(*i*)
LB raporla

BM1Alt algoritması her müşteri için ilk gün hizmet verebilecek en yakın eczaneyi belirler ve bu müşteriyi bu eczaneye bir gün atama maliyetini *LB* değerine ekler. Daha sonra bu en yakın eczanenin mesafesi çok büyük bir değere eşitlenir ve ikinci en yakın eczane için aynı işlem gerçekleştirilir. İşlem müşteriye en yakın *T* eczane için yapıldıktan sonra bir sonraki müşteriye geçilir. Algoritma tüm müşteriler sona erdiğinde *LB* değerini raporlar ve durur.

GM1 Problemi için Alt Sınırlar

GM1 problemi için iki alt sınır geliştirdik. GM1 probleminde eczaneler bölgelere ayrılır ve her bölgedeki eczaneye atanacak nöbet sayısı için alt ve üst değerler belirlenir. İlk alt sınır

eczanelere nöbet atanabilecek gün sayısını göz önüne alan bir algoritmadır. İkinci alt sınır ise bu bilgiye ek olarak bir bölgede nöbet üst değerinde nöbet tutabilecek eczane sayısını da dikkate alır.

NöbetAdediÜstSınırı$_j$: j eczanesine atanabilecek en fazla nöbet sayısını gösterir. Bu sayı *T'nin j* eczanesinin bulunduğu r_j bölgesindeki eczane sayısına oranının yukarı yuvarlanmış değerine eşittir.

$$NöbetAdediÜstSınırı_j = \left\lceil \frac{T}{|J_{r_j}|} \right\rceil$$

Algoritma 3. GM1Alt1 Algoritması

LB=0
Her *i*=1→*I* için tekrarla
    t=0
    (*t* ≤ T) doğru iken devam et
        EnYakın=*M*
        Her *j*=1→*J* için tekrarla
            Eğer d_{ij}<*EnYakın*
                EnYakın=d_{ij}
                EnYakınj=*j*
            Eğer sonu
            Tekrarla sonu (*j*)
        NöbetAdedi=*NöbetAdediÜstSınırı*$_{EnYakınj}$
        Eğer (*T-t* ≤ *NöbetAdedi*) ise *NöbetAdedi*=*T-t*
        LB=*LB* + *EnYakın* X h_i X *NöbetAdedi*
        $d_{i,enyakınj}$=*M*
        t=*t* + *NöbetAdedi*
    Devam et sonu (*t*)
Tekrarla sonu(*i*)
LB raporla

Bu algoritma her müşteriye en yakın eczaneyi bulur ve bu eczaneye tutabileceği en fazla miktarda nöbeti atar. Bu müşteri için kalan gün sayısı, *T-t,* nöbet adedinden az ise nöbet adedi azaltılır, bu azaltma işlemi sadece bir kez ve müşteriye hizmet verecek son eczane için olur. Alt sınır değerli *LB*, en yakın eczanenin mesafesi, müşteri talebi ve eczanenin nöbet tutacağı gün sayısının çarpımı kadar artırılır. Bu müşteri ile atanan eczane arası mesafe *M* değerine çıkartılır. Eğer bu müşteri için halen atanması gereken gün var ise kalan eczanelerin en yakını bulunarak işleme devam edilir. Müşterinin tüm günleri atanmış ise aynı işlem diğer müşterilere uygulanır. Algoritma sonunda *LB* değerini raporlar.

GM1Alt1 algoritması bir eczaneye atanabilecek en fazla nöbet adedini göz önüne alır. Ancak aynı bölgedeki eczanelerin sadece bir bölümü bu şekilde nöbet tutabilir. Kalan eczaneler alt sınır değerinde nöbet tutmak durumunda olabilir.

Örneğin *T*=10 gün ve bir bölgedeki eczane sayısı 7 olsun. Bu bölgedeki eczaneler için üst sınır 2 (*NöbetAdediÜstSınırı$_j$=2*), alt sınır 1'dir. Ancak sadece 3 eczane üst sınırda nöbet tutabilir. Diğer 4 eczane alt sınır değeri olan 1 kez nöbet tutmak zorundadır. Bu yaklaşımın algoritmaya aktarılması için her bölgede üst sınırda nöbet tutacak eczane sayısı tanımlanmıştır.

ÜSNTEA$_k$:(*ÜstSınırdaNöbetTutacakEczAdedi*) *k* bölgesinde üst sınırda nöbet tutabilecek eczane adedi.

$$\text{Ü}SNTEA_k = T - |J_k| \times \left\lceil \frac{T}{|J_k|} \right\rceil$$

GM1Alt2 algoritması GM1Alt1 algoritmasına benzer şekilde çalışır. Ancak NöbetAdedi belirleme işlemleri esnasında en yakın olarak belirlenen EnYakınj eczanesinin bölgesini tespit eder ve *k* değişkenine atar. GM1Alt1 algoritması bu eczaneye atayabileceği en fazla nöbet miktarını atar ancak GM1Alt2 algoritması bu bölgede üst sınırda atama yapılabilecek eczane sayısının *ÜSNTEA$_k$* 0 olması durumunda NöbetAdedi'ni 1 birim azaltarak alt sınıra çeker. Eğer üst sınırda atama yapılabilecek eczane sayısı pozitif ise NöbetAdedi değerini değiştirmez ancak *ÜSNTEA$_k$* değerini 1 birim azaltır.

Algoritma 4. GM1Alt2 Algoritması

LB=0

Her *i*=1→*I* için tekrarla

t=0

(*t* ≤ T) doğru iken devam et

EnYakın=M

Her *j*=1→*J* için tekrarla

Eğer d_{ij}<*EnYakın*

EnYakın=d_{ij}

EnYakınj=j

Eğer sonu

Tekrarla sonu (*j*)

NöbetAdedi=NöbetAdediÜstSınırı$_{EnYakınj}$

k= $r_{EnYakınj}$

Eğer (ÜSNTEA$_k$=0) ise *NöbetAdedi = NöbetAdedi* -1

Eğer (ÜSNTEA$_k$>0) *ÜSNTEA*$_k$ = *ÜSNTEA*$_k$ – 1

Eğer (*T-t ≤ NöbetAdedi*) ise *NöbetAdedi=T-t*

LB=LB + EnYakın X h_i X *NöbetAdedi*

$d_{i,EnYakınj}$=*M*

t=t + NöbetAdedi

Devam et sonu (*t*)

Tekrarla sonu(*i*)

LB raporla

3.4 İlkel Üst Sınırlar

Bu bölümde problemler için geliştirilen ilkel üst sınır algoritmalarından bahsedeceğiz. Bu algoritmalar olurlu çözümler bulmayı hedeflemektedir. Bu olurlu çözümler sezgisel yöntemler için başlangıç çözümleri olacak ve sezgisel yöntemler bu çözümleri iyileştirerek en iyi sonuca yaklaşmaya çalışacaklar.

Geliştirilen üst sınırlarda sadece eczanelere nöbet atama kararları veriyoruz. Çünkü verilen bir nöbet planına göre her bir müşterinin her gün hangi eczaneye atanacağı polinom zamanda çözülebilen bir problemdir.

BM1 Problemi için Üst Sınır

BM1 probleminde her eczaneye sadece bir kez nöbet atanabilir. Bu üst sınır algoritması amaç fonksiyonunun değerini göz önünde bulundurmaksızın olurlu bir çözüm elde etmeyi hedeflemektedir. BM1Üst Algoritması aşağıda verilmiştir. *UB* değeri bu algoritma sonucunda bulunan amaç fonksiyonu değerinin üst sınırıdır.

Bu algoritma birinci günden başlayarak her güne bir eczane atayarak planlama periyodunun sonuna kadar ilerler. Planlama periyodunun son gününe de atama yapıldıktan sonra ilk güne dönülerek işlem devam eder. İşlem atama yapılacak eczane kalmayınca kadar devam eder. Son adımda, atama kararları verilen çözümün maliyeti OAA (Optimal Atama Algoritması) kullanılarak hesaplanır.

Algoritma 5. BM1Üst Algoritması

UB=∞
t=0
Her j =1 → J eczanesi için tekrarla
Eğer $t \geq T$ ise t=0
t=t+1
y_{jt}=1
Tekrarla sonu
Atama maliyetini OAA kullanarak hesapla ve UB'yi güncelle

GM1 Problemi için Üst Sınır

GM1 Probleminde her eczaneye birden fazla kez nöbet atanabilir. Bu üst sınır algoritması BM1 problemi için geliştirilen algoritmaya benzemektedir. Ancak her bölgeden bir günde sadece tek

eczanenin nöbetçi olabileceği kısıtını göz önüne alacak şekilde geliştirilmiştir. Algoritma 6'da GM1Üst Algoritması verilmiştir.

Bu algoritma birinci bölgeden başlayarak bir bölge seçer. Her gün seçilen bölgedeki sıradaki eczane belirlenir ve bu eczaneye nöbet atanır. Eczane listesinin sonuna gelindiğinde tekrar başa dönülüp devam edilir. Günlerin sonuna gelindiğinde yeni bir bölge seçilip tüm bölgeler bitene kadar işleme devam edilir. Algoritmanın son adımında atamanın maliyeti OAA kullanılarak bulunur.

Algoritma 6. GM1Üst Algoritması

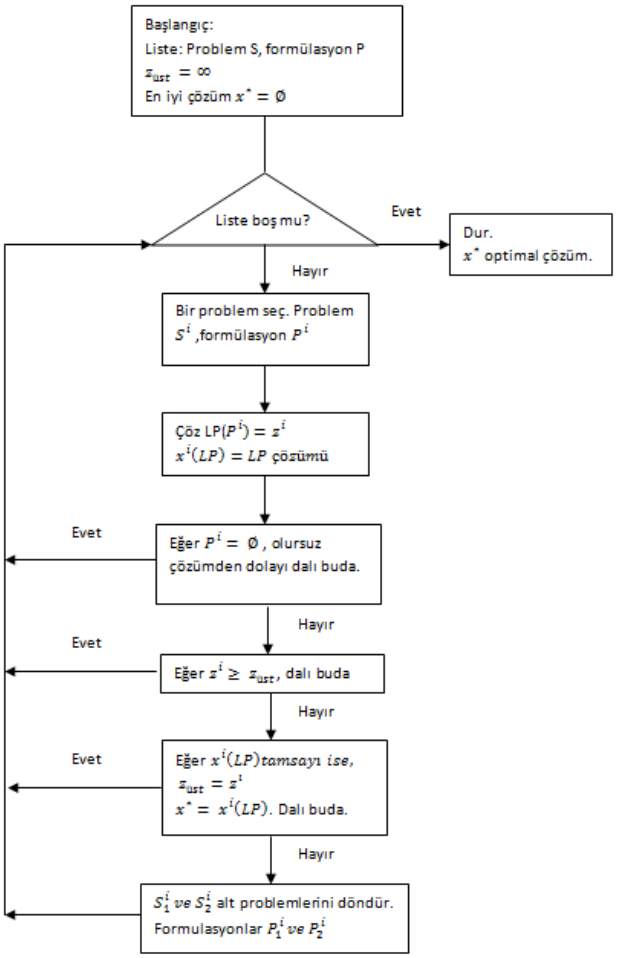
UB=∞
Her *k*=1 → *K* için tekrarla
    j=1
    Her *t*=1→*T* için tekrarla
        atamayapıldı=0
        (*atamayapıldı* == 0) doğru iken devam et
            Eğer $j \in J_k$ ise
                y_{jt}=1
                atamayapıldı=1
            Eğerin sonu
            j=*j*+1
            Eğer *j*>*J* ise *j*=1
        Devam et sonu
    Tekrarla sonu (*t*)
Tekrarla sonu (*k*)
Atama maliyetini OAA kullanarak hesapla, *UB*'yi güncelle

3.5 Dal-Sınır Algoritması

Bu bölümde Eczane Nöbet Çizelgeleme (ENÇ) problemi için geliştirilen dal-sınır algoritmasını (DSA) aktaracağız. DSA'nin genel yapısı ve işleyişini, ENÇ için geliştirilen DSA algoritmasını, alt

ve üst sınır algoritmalarını, problem büyüklüğünü azaltma algoritmalarını ve simetri kırma yönteminden bahsedeceğiz.

Bu bölümde ele aldığımız ENÇ problemi GM1'i temel alır. Ayrıca her *j* eczanesinin nöbet sayısının n_j olarak belirlenmiş olduğunu varsaydık. Bu n_j değerleri ilgili bölgenin nöbet alt ve üst sınırları arasında yer alır ve (GM1-3) ve (GM1-4) kısıt kümelerini sağlar.



Şekil 1. Genel dal sınır algoritması (Wolsey, 1998)

3.5.1 Genel Yapı ve İşleyiş

Dal-sınır algoritması, problem için en iyi çözümü bulma ihtimalinin yüksek olduğu alt çözüm uzaylarında aramaya dayalıdır. Bir alt uzayın en iyi çözümü bulundurmayacağı ispatlandığında ise o uzayda arama yapmaya devam edilmez. Böylece çözüm uzayının daha küçük bir kısmı aranarak en iyi çözüm bulunmuş olur. Dal-sınır algoritması kesikli programlama problemleri için

sıkça başvurulan bir yöntemdir. Genel bir dal-sınır algoritması kesikli bir enazlama problemi için Şekil 1'deki gibi özetlenebilir (Wolsey,1998).

Başlangıçta üst sınır değerine çok büyük bir değer atanır ve algoritma olurlu çözümler buldukça üst sınır değeri güncellenir. Kök düğümde problemin doğrusal gevşetmesi çözülür ve en iyi çözüm değeri kontrol edilir. Eğer çözüm tamsayı ise probleme en iyi çözüm bulunmuş olur. Eğer değilse, değeri kesirli olan bir değişken seçilir ve en yakın aşağı ve yukarı tamsayılara yuvarlanarak iki farklı alt problem oluşturulur. Bir alt problemin doğrusal gevşetilmiş en iyi çözümü o problemden türetilecek olurlu çözümler için bir alt sınırdır. Bu alt sınır değeri üst sınır değeri ile karşılaştırılır. Eğer üst sınıra eşit veya ondan daha büyükse bu düğüm budanır. Eğer doğrusal gevşetilmiş en iyi sonuç tamsayı ise üst sınır güncellenir ve düğüm budanır. Diğer durumlarda alt problemler türetilerek aramaya devam edilir.

3.5.2 ENÇ Problemi İçin Bir Dal-Sınır Algoritması (ENÇDSA)

Bu bölümde ENÇ problemine özgü geliştirilen dal-sınır algoritması anlatılmaktadır. Algoritmanın temel adımları Algoritma 7'de verilmiştir.

Algoritma 7. Basit dal-sınır Algoritması (BDSA)

Başlangıç: Olurlu bir çözüm bul ve bilinen en iyi çözüm olarak tut.
İlk gün için bir bölge seç.

Adım 1: Bu bölgeden daha önce seçilmemiş ve tutulacak nöbeti olan bir eczane varsa
Adım 2'ye git.

Adım 2: Eczaneye bu gün için nöbet ata ve nöbet bilgisini güncelle.
Üst sınır algoritması ile olurlu bir çözüm bul ve bilinen en iyi çözüm ile karşılaştır.
Eğer bilinen en iyi çözümden daha iyi ise güncelle. Adım 3'e git.

Adım 3: Alt sınır algoritması ile bir alt sınır bul.
Eğer bulunan alt sınır bilinen en iyi çözümden daha yüksek ise dur. Bu noktaya kadar belirlenen yarı çizelge en iyi çizelgede yer alamaz. Adım 1' e git. Eğer değilse, bir sonraki bölgeye geç. Eğer tüm bölgeler için o günün çizelgesi belirlenmişse ve planlama ufku bitmediyse bir sonraki güne geç. Adım 1'e git.

ENÇ problemini çözmek için eczane nöbet atamalarına karar vermek yeterlidir, çünkü; verilen bir çizelgede her mahalle her gün kendisine en yakın eczaneye atanacaktır. Bu özellikten yola çıkarak, dal-sınır algoritmasında kullanılan ağaç yapısı şu şekildedir: İlk günden başlayarak her bölgeden hangi eczanenin nöbetçi olacağına karar verilmelidir. Bu sebeple, kök düğümden ilk bölgedeki eczane sayısı kadar çocuk düğüm türetilir. Her çocuk düğüm, ilk gün için o bölgeden hangi eczanenin atamasının yapıldığı bilgisini taşımaktadır. Seçilen bir çocuk düğümden devam edilerek tüm bölgeler için ilk gün ataması yapılır. Birinci günden sonra ikinci gün ile devam edilerek, düğüm budanmadığı sürece planlama ufkunun sonuna kadar devam edilir. Daha sonra ikinci sırada incelenecek bölge ile devam edilir. Bu şekilde derin öncelikli (depth-first) yaklaşım uygulanmış olmaktadır.

Bu aşamada bölge seçim sırasının ve bir bölgedeki eczanelerin seçim sırasının belirlenmesi gerekmektedir. Bu sıralamaların alt sınır değerlerini etkileyeceği düşünülmüş olup, alt sınır değerlerinin kalitesini daha erken arttırabilmek amacıyla çeşitli sıralama ölçütleri düşünülmüş ve çözüm süresi üzerindeki etkileri iki problem büyüklüğündeki örnekler üzerinden ölçülmüştür.

- *Orta Bölge – Sınırdaki Eczane Önceliği(A)*: Problemde ele alınan coğrafik bölgelerde nüfusun orta kısımlarda toplanmış olacağı düşünülerek, öncelikli olarak coğrafik olarak ortalarda yer alan bölgelerde eczane atamasına karar vermenin daha fazla mahalleyi etkileyerek daha fazla atamanın yapılmasını sağlayacağı düşünülmüştür. Bu bölgede, sınırdaki bir eczanenin nöbetine öncelikli olarak karar vermenin ise alt sınırın kalitesini daha da arttıracağı düşünülmüştür.

Bu önceliğe karar verebilmek için orta bölge ve sınırdaki eczanelerin bir ölçüt yardımıyla yaklaşık olarak belirlenebilmesi gerekmektedir. Bu sebeple bölgeler aşağıdaki bölge belirleme ölçütleri (BBÖ) yardımıyla sıralanmaktadır:

$$reg\_dist_{k1,k2} = min\{d_{j1,j2}: r_{j1} = k_1, r_{j2} = k_2\} \quad \forall k_1, k_2 \qquad \text{(BBÖ1)}$$

Yukarıdaki ölçüt biri k_1, diğeri k_2 bölgesinde olan eczane çiftleri içerisinde birbirine en yakın olan çift arasındaki uzaklığı hesaplamaktadır.

$$reg\_metric_k = \sum_{\substack{k' \in K \\ k' \neq k}} reg\_dist_{kk'} \quad \forall k \qquad \text{(BBÖ2)}$$

BBÖ2 ise her bölge için BBÖ1 ölçütlerinin toplamıdır. BBÖ2 değeri küçük olan bir bölgenin coğrafik olarak diğer bölgelere göre ortada yer alacağı düşünülmüştür. BBÖ2 ölçütüne göre bölge sırası belirlenmiştir. Oluşturulan rassal örneklerin üzerinde denendiğinde de bu ölçütte küçük değere sahip olan bölgelerin kare şeklindeki yüzey üzerinde orta kısımlarda yer aldığı görülmüştür.

Sınırdaki eczanelere öncelik verebilmek için ise diğer bölgelerden olan eczaneler arasında eldeki eczaneye en yakın olan eczane ile arasındaki uzaklık (pr_dist_j) ele alınmıştır. Bir bölgede, bu ölçütte en küçük değeri alan eczaneye öncelik verilmiştir.

$$pr\_dist_j = \min\{d_{jj'} : r_j \neq r_{j'}\} \quad \forall j \qquad \text{(BBÖ3)}$$

Bölge ve eczane seçiminin önceliklendirilmesinin etkisini inceleyebilmek için aşağıdaki yaklaşımlar karşılaştırılmıştır.

- *Orta Bölge – Sınırdaki Eczane (A):* $k' = \text{argmin}_k\{reg\_metric_k\}$ ve $j' = \text{argmin}_{j \in J_{k'}}\{pr\_dist_j\}$
- *Orta Bölge – Orta Eczane (B):* $k' = \text{argmin}_k\{reg\_metric_k\}$ ve $j' = \text{argmax}_{j \in J_{k'}}\{pr\_dist_j\}$
- *Sınırdaki Bölge–Sınırdaki Eczane(C)* : $k' = \text{argmax}_k\{reg\_metric_k\}$ ve $j' = \text{argmin}_{j \in J_{k'}}\{pr\_dist_j\}$
- *Sınırdaki Bölge – Orta Eczane (D):* $k' = \text{argmax}_k\{reg\_metric_k\}$ ve $j' = \text{argmax}_{j \in J_{k'}}\{pr\_dist_j\}$

Küçük boyutlu iki problem üzerinde yukarıda belirtilen 4 farklı sıralama yöntemi ile çözümler elde edilmiş ve çözüm süreleri karşılaştırılmıştır.

Tablo 2. Sıralama yaklaşımlarının rassal iki problem boyutu üzerindeki performansları

Problem Büyüklüğü				**Örnek Adedi**	**En İyi Olduğu Örnek Adedi**			
I	**J**	**T**	**K**		**A**	**B**	**C**	**D**
20	10	10	4	10	3	**5**	1	1
60	30	5	9	7	**5**	1	2	1
Toplam				17	**8**	6	3	2

Tablo 2'de ilk problem 10 örnek üzerinden, ikinci problem ise 7 örnek üzerinden değerlendirilmiştir. Farklı bölge ve eczane seçim yaklaşımlarının aynı problemi çözme süreleri tutulmuş ve her örneği en kısa sürede çözen yaklaşalım belirlenmiştir. Her yaklaşımın altında yazan rakamlar, ilgili problem büyüklüğünde o yaklaşımın kaç defa en hızlı olduğunu göstermektedir. Satır toplamlarının örnek adetlerinden fazla olmasının sebebi bazı örneklerde birden fazla yöntemin aynı sürede çözüme ulaşmasıdır. Toplam değerlere bakıldığında orta bölge ve kenardaki eczane (A) seçiminin daha iyi sonuçlar verdiği gözlemlenmiş ve bu yöntemle devam edilmiştir. Ayrıca, bu yaklaşımın gerçek problem üzerinde de iyi sonuçlar vereceği düşünülmüştür ancak problemin büyüklüğünden dolayı bu test yapılamamıştır.

Dal-sınır algoritmasında kullanılmak üzere, bir alt sınır algoritması ve dört farklı üst sınır algoritması geliştirdik. Ayrıca çözüm uzayında simetrik çözümler olduğunu gözlemledik ve bu simetriyi kırmak için bir simetri kırma yöntemi geliştirdik.

3.5.3 Alt Sınır Algoritması

Alt sınır algoritması iki ana parçadan oluşmaktadır. Çizelgesi belirlenen günler için mahalle eczane atamaları Optimal Atama Algoritması (Algoritma 8) yardımıyla kesin bir şekilde yapılmakta, diğer günler için ise Başlangıç Alt Sınır Algoritması (Algoritma 9) çalıştırılmaktadır.

Algoritma 8. Optimal Atama Algoritması (OAA)

Her i =1 → I müşterisi için tekrarla

Her t =1→ T günü için tekrarla

$$j^* = \underset{j|y_{jt}=1}{\operatorname{argmin}}\{d_{ij}\}$$

$$x_{ij^*t} = 1$$

Tekrarla sonu (t)

Tekrarla sonu (i)

Algoritma 9. Başlangıç Alt Sınır Algoritması (BASA)

LB=0, t=0
Her *i*=1→*I* için tekrarla
    t<T doğru iken tekrarla
        enyakın=*M*
        Her *j*=1→*J* için tekrarla
            Eğer d_{ij}<*enyakın*
                enyakın=d_{ij}
                enyakınj=*j*
            Eğer sonu
        Tekrarla sonu (*j*)
        NobetAdedi=nobetsayısı[*enyakınj*]
        Eğer T-t < NobetAdedi
            NobetAdedi= T-t
        Eğer sonu
        LB=LB+ *enyakın* X h_i X NobetAdedi
        t = t + NobetAdedi
        $d_{i,enyakınj}$=*M*
    Tekrarla sonu (*t*)
Tekrarla sonu(*i*)
LB raporla

Dal–sınır algoritması çalıştırılırken, planlama ufkunda ilerledikçe o güne kadar belirlenen çizelge dikkate alınarak alt sınır değerinin güncellenmesi gerekmektedir. Bu amaçla Algoritma 10 olarak verilen Dal-Sınır Alt Sınır Algoritması (DSASA) kullanılmaktadır.

Algoritma 10. Dal-Sınır Alt Sınır Algoritması (DSASA)

Eğer *gün* için çizelge tamamlanmışsa time = *gün*

Değilse time = *gün* - 1

Eğer sonu

AtananToplam=0

Her *i* =1 → *I* müşterisi için tekrarla

Her *t* =1→ *time* günü için tekrarla

$j^* = \underset{j|y_{jt}=1}{\operatorname{argmin}}\{d_{ij}\}$

$x_{ij^*t} = 1$

AtananToplam = AtananToplam + d_{ij^*} X h_i

Tekrarla sonu (*t*)

Tekrarla sonu (*i*)

t= time

Her *i*=1→*I* için tekrarla

t<T doğru iken tekrarla

enyakın=*M*

Her *j*=1→*J* için tekrarla

Eğer d_{ij}<*enyakın*

enyakın=d_{ij}

enyakınj=*j*

Eğer sonu

Tekrarla sonu (*j*)

NobetAdedi=nobetsayısı[*enyakınj*]

Eğer T-t < NobetAdedi

NobetAdedi= T-t

Eğer sonu

LB=LB+ *enyakın* X h_i X NobetAdedi

t = t + NobetAdedi

$d_{i,enyakınj}$=*M*

Tekrarla sonu (*t*)

Tekrarla sonu(*i*)

3.5.4 Üst Sınır Algoritmaları

Dal sınır algoritmasında herhangi bir düğümde, verilmiş olan kararları göz önüne alarak olurlu çözümler bulmak amacı ile dört farklı üst sınır algoritması geliştirilmiştir.

Üst Sınır Algoritması 1 (ÜS1)

ÜS1 algoritması çizelgesi henüz belirlenmemiş gün ve bölgelere rassal bir şekilde eczane ataması yapar. Bir günün çizelgesinde eksik bölgeler varsa ilk olarak onları tamamlar, daha sonrasında kalan nöbetler ile eğer varsa kalan günlerin çizelgesini oluşturur.

Algoritma 11. Üst Sınır 1 (ÜS1)

```
Eğer bolge !=K (bolge K değil iken)
      Her k= bolge+1 → K
         Her i=1 → eczaneSayısı[k]
            Eğer nöbetSayısı[bolgeEczaneSeti[k][i]] > 0
               cozum[k][gün]= bolgeEczaneSeti [k][i];
               nöbetSayısı [cozum[k][gün]] --
               Çık;
            Eğer sonu
          Tekrarla sonu(i)
       Tekrarla sonu(k)
 Eğer sonu
Eğer gün != T
   Her k=1→ K
      t=gün+1;
         Her i=1 → eczaneSayısı[k]
            nöbetSayısı[bolgeEczaneSeti[k][i]] > 0 doğru iken devam et
               cozum[k][t]= bolgeEczaneSeti[k][i]
               nöbetSayısı[cozum[k][t]]--
               t++;
            Devam et sonu
          Tekrarla sonu(i)
   Tekrarla sonu(k)
Eğer sonu
```

Üst Sınır Algoritması 2 (ÜS2)

ÜS2 algoritmasının arkasındaki temel fikir, her gün için nöbet tutacak eczaneleri müşteri noktaları etrafında iyi yayılacak şekilde planlamaktır. Diğer bir deyişle, farklı bölgeden birbirine çok yakın eczanelere aynı günde nöbet atanmasına izin vermemektir. Bu amaçla, herhangi bir gün için bir bölgeden eczane seçerken o eczanenin diğer bölgelerden nöbette olan eczanelere en uzakta olması amaçlanmıştır. İki eczane arası yakınlık, eczanelerin servis verebildiği mahalle kümelerinin benzerliği şeklinde tanımlanmıştır.

Eczanelerin servis verebildiği mahalle kümeleri ise şu şekilde bulunmuştur. Eczaneleri bir mahalleye olan uzaklıklarına göre yakından uzağa doğru sıraladığımızda, herhangi bir bölgenin tüm eczaneleri kapsandığında, son eczane bu mahallenin olurlu bir çözümde atanabileceği en uzak eczanedir ve bu eczaneye kadar olan tüm eczaneler mahallenin servis alabileceği eczane kümesidir. Çünkü her olurlu çözümde bu bölgeden bir eczane mutlaka nöbette olacaktır. Bu şekilde eczanelerin servis verebildiği mahalle kümeleri oluşturulmuştur.

Algoritma 12. Üst Sınır 2 (ÜS2)

Notasyon:

Sj : j eczanesi tarafından servis verilebilecek mahalle sayısı
dsetj : j eczanesi tarafından servis verilebilecek mahalle kümesi
reg : bulunan günde çizelgesi belirlenmiş bölge sayısı
day : içinde bulunulan gün

Algoritma:

Eğer reg != K
    union_dist = ∪ $dset_j$ $\forall j$
    Çizelgesi belirlenmemiş bölge var iken devam et
        Her j = 1 → J
            Eğer nöbet$_j$ > 0 ve bölge$_j$ nin çizelgesi henüz belirli değilse
                benzerlik$_j$ = union_dist ∩ $dset_j$
            Eğer sonu
        Tekrarla sonu(j)
        p = min{ | benzerlik$_j$ | }

$y_{p,day}$ = 1

union_dist = ∪ $dset_p$

Devam et sonu

Eğer sonu

Rastgele bir bölge seç ve tüm çizelgesini rassal bir şekilde oluştur.

Her t = day + 1 → T

union_dist = $dset_j$, y_{jt} = 1

Çizelgesi belirlenmemiş bölge var iken devam et

Her j = 1 → J

Eğer nöbet$_j$ > 0 ve bölge$_j$ nin çizelgesi henüz belirli değilse

benzerlik$_j$ = union_dist ∩ $dset_j$

Eğer sonu

Tekrarla sonu(j)

p = min{ | benzerlik$_j$ | }

$y_{p,day}$ = 1

union_dist = ∪ $dset_p$

Devam et sonu

Tekrarla sonu(t)

OAA algoritmasını çalıştır

Üst Sınır Algoritması 3 (ÜS3)

Bu üst sınır algoritmasında alt sınır değerini veren mahalle atamaları bilgisi kullanılmıştır. Amaç alt sınır çizelgelerini olabildiğince kullanarak daha düşük maliyetli olurlu çözümler elde etmektir. Bu amaç için ilk önce kritik mahalle tanımlaması yapılmıştır. Olurlu çözümlerde her mahallenin alt sınır atamalarının gerçekleşme ihtimali düşük olduğu için, bir mahallenin alt sınır atamalarından farklı eczanelere atanması durumunda alt sınırın değerindeki artışa bakılmıştır. Hangi farklı eczanelere atanacağı konusu ise planlama ufkunu belirli bir miktar artırdığımızda oluşan durum ile eşdeğerdir. Bu sebeple, planlama ufku artırıldığı zaman atama maliyeti en çok artan mahalleler kritik mahalleler olarak tanımlanmıştır.

Algoritma bulunan noktaya kadar gerçekleşen atamaları dikkate alarak başlar. En kritik mahallenin alt sınır algoritmasında atandığı eczanelerin nöbette olduğu günlerde yine aynı

eczanelere nöbet atamaya çalışır. Daha sonra aynı işlemi bir sonraki en kritik mahalle ile yapar ve belirli bir mahalle sayısı kadar devam eder. Sonrasında olurlu çözümde boş kalan yerler rassal bir şekilde doldurulur.

Algoritma 13. Üst Sınır 3 (ÜS3)

Notasyon:

$LBSch_{i,t}$:alt sınır algoritması sonucu i mahallesinin t gününde atandığı eczane

crt_i : i. kritik mahalle

day : içinde bulunulan gün

$duty_j$: j eczanesinin kalan nöbet sayısı

$sol_{k,t}$: dal-sınır ağacında bu noktaya kadar olan yarı çizelge

$cozum_{k,t}$: ÜS3 çözümü

$region_j$: j eczanesinin bölgesi

$regionpset_{k,s}$: k bölgesinde s. Sıradaki eczane

$pnumber_k$: k bölgesinde bulunan eczane sayısı

Algoritma:

Her k = 1 → K

  Her t = 1 → T

    $Cozum_{k,t} = sol_{k,t}$

    Eğer $sol_{k,t}$ == 0

      $Flag_{k,t} = 0$

    değilse

      $flag_{k,t} = -1$

    Eğer sonu

  Tekrarla sonu(t)

Tekrarla sonu(k)

Her i = 1 → I

  Her t = 1 → T

Eğer $duty_{LBSch_{crt_i,t}} > 0$ ve $flag_{region_{LBSch_{crt_i,t}},t} == 0$

$çözüm_{region_{LBSch_{crt_i,t}},t} = LBSch_{crt_i,t}$

$flag_{region_{LBSch_{crt_i,t}},t} = -1$

$duty_{LBSch_{crt_i,t}}--$

Eğer sonu

Tekrarla sonu(t)

Tekrarla sonu(i)

Her k = 1 → K

Her t = 1 → T

Eğer $cozum_{k,t} == 0$

Her s = 1 → $pnumber_k$

Eğer $duty_{region\_pset_{k,s}} > 0$

$Cozum_{k,t} = regionpset_{k,s}$

$Flag_{k,t} = -1$

$duty_{region\_pset_{k,s}}$--

$s = pnumber_k + 1$

Eğer sonu

Tekrarla sonu(s)

Eğer sonu

Tekrarla sonu(t)

Tekrarla sonu(k)

Üst Sınır Algoritması 4 (ÜS4)

Yapılan ön deneylerde ÜS3 algoritmasının iyi sonuçlar verdiği ancak yavaş çalıştığı gözlenmiştir. ÜS1 algoritması sadece rassal atamalar yaptığı için hızlı çalışmakta ancak yeteri kadar iyi sonuçlar verememektedir. Bu sebeple ÜS1 ve ÜS3 algoritmalarının karma bir şekilde kullanıldığı bir üst sınır algoritması olan ÜS4 geliştirilmiştir. ÜS4 algoritması dal-sınır ağacında ilerlerken planlama ufkunun ilk %α kısmında ÜS3 daha sonrasında ise ÜS1 algoritmasını kullanmaktadır. Böylece ilk düğümlerde iyi üst sınırlar bulunarak birçok düğümün elenmesi amaçlanmıştır. Planlama ufkunun kalan bölümlerinde zaten çoğu nöbet kararı verilmiş olduğu için kalan kararları rassal şekilde vermek üst sınır bulmayı hızlandırmıştır.

Ayrıca ÜS3 algoritması tüm kritik mahalleleri kritiklik sırası ile ele almaktadır. Bu mahallelerin sayısı arttıkça nöbet çizelgesi belirmeye başlamakta ve yeni bir kritik mahallenin bu çizelgeye etkisi azalmaktadır, ayrıca mahallenin alt sınır ile bulunan çizelgesi ile mevcut çizelgeyi karşılaştırmak zaman alan bir işlemdir. Bu sebeple tüm kritik mahalleleri incelemek yerine, tüm mahalleler içinde en kritik %β mahalleyi incelemek kalan çizelgeyi ise rassal olarak olurlu bir şekilde hazırlamak bu algoritmanın temel yaklaşımı olmuştur.

3.5.5 Problem Büyüklüğünü Azaltma Algoritmaları

Problem büyüklüğü arttıkça dal-sınır algoritması en iyi sonucu verememesinden dolayı, algoritmayı uygulamadan önce ele alınan problemin büyüklüğünü azaltmaya yönelik yaklaşımlar aşağıda sunulmuştur. Bu yaklaşımlardan ilk ikisi mahalle sayısını azaltmaya yöneliktir ve gerçek problemin en iyi sonucunu değiştirmez. Eczane sayısını azaltmaya yönelik sunulan algoritma ise dal-sınır algoritması ile elde edilecek çözümlerin gerçek problem için en iyi olduğunu garantilemez ve ancak sezgisel çözümler olarak kullanılabilir. Bunlara ilave olarak problemi daha küçük problemlere ayırmak için de bir algoritma sunulmuştur. Bu yaklaşımlar 2010 yılının 3. dönemine ve 2011 yılının 1. dönemine ait gerçek veri seti üzerinde denenmiş ve sonuçları aşağıdaki bölümlerde raporlanmıştır.

Mahalle Atma Algoritması

Eğer bir mahallenin planlama ufku boyunca sadece tek bir bölgeden eczanelere atanacağını ispatlayabilirsek, o mahalleyi problemden atabiliriz. Bu durumda, mahalleyi o bölgedeki eczanelere nöbet sayıları kadar atar ve elde edilen amaç fonksiyonu değerini optimizasyon sonrasında elde edilen amaç fonksiyonu değerine ekleriz. Bir mahallenin sadece tek bir bölgeden hizmet aldığını ispatlamak için ise o mahalleye en yakın eczaneleri incelemek yeterlidir. Bir mahalle herhangi bir bölgeden kendisine en uzak olan eczane ile arasındaki uzaklıktan daha uzakta bulunan bir eczaneden hizmet alamaz. Bu değere, bu mahalle için kritik uzaklık diyebiliriz. Bu kritik uzaklıktan daha yakında bulunan eczanelerin hepsi aynı bölgeden ise mahalle sadece bu bölgeden hizmet alıyor demektir.

Yukarıda belirtilen gerçek probleme ait iki veri seti de aynı mahalle bilgisini içermektedir. Eczane bilgileri ise çok az miktarda farklılık göstermektedir. Her iki veri setinde de aynı sonuç elde edilmiş ve mahalle atma algoritması mahalle sayısını %19 azaltmıştır.

Algoritma 14. Mahalle Atma Algoritması

cr_dist_i : i mahallesi için kritik uzaklık

$rank_{it}$: i müşterisine t. en yakın eczane

Her *i* =1 → *I* müşterisi için tekrarla

$r_{rank_{i1}} = r'$;

bitir=0;

$d_{i,rank_{it}} \leq cr\_dist$ ve bitir = 0 doğru iken tekrarla

Eğer $r_{rank_{it}} \neq r'$

bitir=1;

Eğer sonu

t++;

Tekrarla sonu (t)

Eğer bitir =0

Mahalle i'yi çıkar;

Eğer sonu

Tekrarla sonu (*i*)

Mahalle Birleştirme Algoritması

Bu algoritmadaki temel fikir, iki mahallenin hangi koşullar altında tüm olurlu çözümlerde her gün aynı eczaneye atanacaklarını göstermektir. Eğer bir mahalle seti için bir önceki bölümde değinilen kritik uzaklıklar aynı ve bu değerden daha yakında bulunan eczane seti aynı ise, bu setteki mahalleler birleştirilebilir.

Elde edilen birleşik mahallenin talebi, kendisini oluşturan mahallelerin taleplerinin toplanması ile bulunur. Bu mahallenin bir *j* eczanesine uzaklığı ise kendisini oluşturan mahallelerin *j* eczanesine olan talep ağırlıklı uzaklıklarının toplamının birleşik mahallenin talebine bölünmesiyle bulunabilir.

Bu algoritmada aranan koşul çok kısıtlayıcı olduğu için gerçek veri setinde sadece 1 adet birleştirme yapabilmiştir. Bu yüzden sonraki aşamalarda dikkate alınmamıştır.

Algoritma 15. Mahalle Birleştirme Algoritması

$agg_{i,s}$: i mahallesi ile birleştirilmiş mahalle setindeki s. mahalle

$dist\_set_i = 0 \quad \forall\, i$

n_i: i mahallesine kritik uzaklıkta veya daha yakında bulunan eczane sayısı

$rank_{it}$: i müşterisine t. en yakın eczane

Her *i* =1 → *I-1* müşterisi için tekrarla

Eğer $dist\_set_i \neq 1$

$agg_{i,0} = 1$;

say=0;

Her *m* =i+1 → *I* müşterisi için tekrarla

Eğer $dist\_set_i \neq 1$ ve $n_i = n_m$

bayrak=1;

Her *c* =1 → n_i için tekrarla

Eğer $rank_{ic} \neq rank_{mc}$

bayrak=0;

çık;

Eğer sonu

Döngü sonu(c)

Eğer bayrak=1

say++;

$dist\_set_m = 1$

$agg_{i,say} = m$;

Eğer sonu

Eğer sonu

Döngü sonu(m)

Eğer sonu

Tekrarla sonu (*i*)

Eczane Birleştirme Algoritması

Gerçek problemde birbirine çok yakın eczaneler olabileceği düşüncesinden yola çıkarak problemdeki eczane sayısını azaltmaya yönelik bir algoritma geliştirilmiştir. Bir mahallenin birbirine çok yakın iki eczaneden herhangi birine atanmasının çok fark yaratmayacağı düşünülerek, ele alınan iki eczanenin tüm mahallelere olan uzaklıkları arasındaki farklar belirli bir eşik değerinin altında ise bu iki eczane birleştirilmiştir. Bu eşik değeri belirlemek için mahallelerin kritik uzaklıklarının ortalaması kullanılmıştır. Bu değer, gerçek problem veri setinde 1000 metre civarındadır. Bu değer göz önüne alınarak, eşik değerin 250 olarak belirlenmesinin gerçekçi olacağı düşünülmüştür.

Algoritmada eczaneler sadece ikişerli olarak birleştirilmemektedir. Yukarıda belirtilen koşulu sağlayan maksimum büyüklükte eczane setleri birleştirilmektedir. Elde edilen birleşik eczanelerin nöbet sayısı kendini oluşturan eczanelerin nöbet sayısına eşittir. Birleşik eczanelerin mahallere olan uzaklıkları ise kendisini oluşturan eczanelerin uzaklıklarının nöbet sayıları ile çarpımlarının birleşik eczanenin nöbet sayısına bölünmesiyle bulunmaktadır.

Eczane veri seti bu şekilde güncellendiği takdirde eczane sayısı %60 azaltılabilmektedir. Ancak, bu algoritmanın uygulanmasından sonra dal-sınır algoritması gerçek veri seti için en iyi çözümü garantilemez. Bu sebeple, nihai test sonuçlarını elde ederken bu algoritma kullanılmamıştır. Buna rağmen, bu algoritma kullanıldığında elde edilecek çözümler gerçek problem için sezgisel birer çözüm olarak kullanılabilir.

Algoritma 16. Eczane Birleştirme Algoritması

$$aggset_{j,p} = \begin{cases} 1 & j\ eczanesi\ ile\ p\ eczanesi\ birleştirilebilir\ ise \\ 0 & Diğer\ durumda \end{cases}$$

Her *j* =1 → *J-1* eczanesi için tekrarla

  Her *p*=j+1 → *J* eczanesi için tekrarla

    Eğer $r_j = r_p$

      $say_i = 0$;

      Her *i* =1 → *I-1* müşterisi için tekrarla

        Eğer $\left|d_{ij} - d_{ip}\right| \leq 250$

---

say_i++;

Değilse

Çık;

Döngü sonu(i)

Eğer say_i = I

$aggset_{j,p} = 1$;

Eğer sonu

Eğer sonu

Döngü sonu(p)

Döngü sonu(j)

// Birleştirilebilir olan eczane çiftlerini gruplama

$group_{j,p}$= j eczanesi ile birleştirilebilir eczaneler kümesinde p. eczane

Her *j* =1 → *J* eczanesi için tekrarla

$say_j = 0$;

$group_{j,0} = j$;

Her *p*=j+1 → *J* eczanesi için tekrarla

Eğer $aggset_{j,p} = 1$

say_j++;

$group_{j,say_j} = p$;

Eğer sonu

Döngü sonu(p)

Döngü sonu(j)

// Bu aşamadan sonra aşağıdaki rutinler çağrılmalıdır.

-Gruplardaki her eczane çiftinin arasındaki mesafeyi kontrol et.

-Mesafe limitini ihlal eden eczaneleri çıkar.

-Birden fazla grupta bulunan eczaneleri sadece bir gruba dahil et.

Problemi Bölme Algoritması

Problem büyüklüğünü düşürmeye yönelik bir diğer uygulama ise problemde ele alınan coğrafi bölgeyi parçalara bölerek daha küçük problemler elde etmek. Bu küçük problemleri üst bölge

olarak adlandırabiliriz. Her mahalle ve eczane bir üst bölgeye aittir. Eğer elde edilen üst bölgeler için en iyi çizelgeler diğer üst bölgelerden bağımsız olarak bulunabiliyorsa, elde edilen çizelgeler birleştirilerek gerçek problem için en iyi çizelge elde edilmiş olur. Bu durumun gerçekleşmesi için sağlanması gereken koşul ise herhangi bir üst bölgeden hiç bir eczanenin başka bir üst bölgeden bir mahalleye servis verememesi ya da hiç bir mahallenin başka bir üst bölgedeki eczaneden servis alamaması gerekir. Gerçek problem üzerinde bu algoritma uygulandığında sadece birkaç eczaneye sahip bir üst bölge ve geri kalan tüm eczaneler için bir üst bölge olmak üzere iki üst bölge elde edildi. Ancak ayrılan üst bölgelerden biri çok küçük olduğu için beklenen küçültme gerçekleşmemiştir. Bunun sebebi olarak, İzmir'deki mahalle ve eczanelerin coğrafik dağılımlarının istenilen koşulu sağlamaya elverişli olmamasını gösterebiliriz.

Algoritma 17. Problemi Bölme Algoritması

$dflag_i = 1 \ \forall i$

$pflag_j = 1 \ \forall j$

$superregion$: içerisinde bir yada daha fazla bölge içeren üst bölge kümesi

mah: algoritmadaki adıma kadar $dflag_i$ değeri 1 olan mahalle kümesi

ecz: algoritmadaki adıma kadar $pflag_j$ değeri 1 olan eczane kümesi

-Müşteri 1'i başlangıç olarak seç.

i=1, $dflag_1 = 1$

dur=1 iken devam et

    i müşterisine hizmet veren her j eczanesi için

        $pflag_j = 1$;

        j eczanesinin hizmet verdiği her k müşterisi için

            $dflag_k = 1$;

            $mah = mah \cup \{k\}$;

        Döngü sonu(k)

    Döngü sonu(j)

    mah kümesindeki her i mahallesi için

        i müşterisine hizmet veren her j eczanesi için

Eğer $pflag_j = 0$

$pflag_j = 1$;

ecz =ecz ∪{j};

Eğer sonu

Döngü sonu (j)

Döngü sonu (mah)

Eğer yeni bir mahalle ya da eczane bulunamıyorsa

ecz kümesi içerisinde tüm eczaneleri bulunan *r* bölgeleri için

$superregion$= $superregion$ ∪ r;

Döngü sonu (ecz)

Eğer sonu

$ecz = \emptyset$; $mah = \emptyset$;

// Herhangi bir alana dahil edilemeyen bir mahalle bulma

say=0;

Her *i* =1 → *I-1* müşterisi için tekrarla

Eğer $dflag_i = 0$

say=1;

müşteri i'yi yeni başlangıç noktası seç

i müşterisinin hizmet alabildiği her j eczanesi için

ecz =ecz ∪{j};

Döngü sonu(i)

Çık;

Eğer sonu

Döngü sonu(i)

Eğer say=0

dur=0;

Eğer sonu

Döngü sonu (dur)

3.5.6 Simetri Kırma

ENÇ probleminin önemli bir özelliği de çözüm uzayında simetrik çizelgeler bulunmasıdır. Hangi eczanelerin birlikte nöbette olduğu önem taşırken bu eczane kümesinin takvim üzerinde hangi

günde nöbette olduğu çözümü değiştirmemektedir. Dolayısı ile problemde $O(T!)$ simetrik çizelge bulunmaktadır. Çözüm uzayında simetrik çözümleri önlemek çözüm süresi bakımından önemli bir avantaj getirecektir. Buradan yola çıkarak geliştirilen simetri kırma algoritması (Algoritma 18) dal-sınır algoritmasından önce çalıştırılmaktadır. Algoritma en çok eczaneye sahip bölgeyi seçerek bu bölgenin çizelgesini rassal bir şekilde sabitler ve dal-sınır algoritması boyunca bu bölgeye dokunulmaz.

Algoritma 18. Simetri Kırma

```
Her k=1 → K
    Eğer pnumber[k] > max_p
      max_p=pnumber[k]
      max_r=k
    Eğer sonu
Tekrarla sonu(k)

    t=1
    Her i=1 → eczaneSayısı[max_r]
      nobetSayısı[bölgeEczaneKümesi[max_r][i ]] > 0 doğru iken devam et
        çözüm[max_r][t]= bölgeEczaneKümesi[max_r][i ]
        nobetSayısı[çözüm[max_r][t]]--
        t++
      Devam et sonu
    Tekrarla sonu(i)
```

ENÇ için geliştirilen alt sınır algoritması (ENÇDS) aşağıda verilmiştir.

Algoritma 19. ENÇ Dal Sınır Algoritması (ENÇDS)

Ana fonksiyon:

```
Gün=0
Bölge=1
Simetri_kır()
Eniyiçözüm= ÜS()
DS(gün+1,bölge,çözüm,nöbetler)
```

DS fonksiyonu:

```
Her  j=1 → bölgeEczaneSayısı [ bölge ]
    Eğer nöbetSayısı [ bölgeEczaneKümesi [ bölge ] [ j ] ] ≥ 1
        nöbetçi= bölgeEczaneKümesi [ bölge ] [ j ]
        Düğümyarat (gün, bölge, nöbetçi, çözüm, nöbetsayısı)
    Eğer sonu
Tekrarla sonu (j)
```

Düğümyarat fonksiyonu:

```
Çözüm[gün][bölge]= nöbetçi
nöbetSayısıGüncelle(nöbetsayısı,nöbetçi)
DSASA algoritmasını çalıştır. Altsınır = DSASA()
    Eğer ÜS() < Eniyiçözüm
        Eniyiçözüm=ÜS()
    Eğer sonu

    Eğer altsınır < Eniyiçözüm
        Eğer bölge < K
            DS(gün, bölge+1,çözüm,nöbetler)
        Değilse eğer gün < T
            bölge=1
            DS(gün+1,bölge,çözüm,nöbetler)
        Eğer sonu
    Eğer sonu
```

3.6 Dal-Fiyat Algoritması (DFA)

Dal-sınır algoritması ile orta ve büyük boyutlardaki problemleri kabul edilebilir sürelerde çözememiz dolayısı ile yeni bir kesin çözüm yöntemi geliştirme çalışmalarına başlanmıştır. ENÇ probleminin çok periyotlu bir problem olması gerçeğinden yola çıkarak, problemin tek periyotlu problemlere ayrıştırılmasının mümkün olduğu görülmüş ve problem yeniden formüle edilmiştir. Yapılan ayrıştırma sonucu ana (master) problem ve fiyatlandırma (pricing) alt problemi ortaya çıkmaktadır.

Ana problemin doğrusal gevşetilmiş en iyi sonucunu bulmak için kolon türetme yöntemi kullanılmaktadır. Her kolon tek günlük olurlu bir çizelgeye karşılık gelmektedir. Tek günlük olurlu tüm çizelgeler arasında mevcut sonucu iyileştirecek olan çizelge fiyatlandırma problemi aracılığı ile bulunmaktadır. Planlama ufkundaki gün sayısı kadar yüksek kalitede tek günlük çizelge elde edildiğinde ana problemin tüm kısıtları sağlanıyorsa eğer, çok periyotlu problem için olurlu bir çözüm bulunmuş olmaktadır. Kolon türetme yönteminin dal-sınır algoritmasına entegre edilmiş hali dal-fiyat algoritması olarak adlandırılmaktadır.

Bu bölümde ENÇ problemine özgü geliştirilen dal-fiyat algoritması (ENÇDFA) anlatılmaktadır. İlk olarak kolon türetme ve dal-fiyat algoritmalarının genel yapısı ve işleyişi sunulacaktır. Algoritmanın temel haline ek olarak, çeşitli iyileştirmeler üzerine çalışılmış ve test edilmiştir. Algoritma üzerinde olası diğer iyileştirme olanakları gelecek araştırma konuları olarak önerilmiştir.

Bu bölümde ele aldığımız ENÇ problemi (bir önceki bölümde olduğu gibi) GM1'i temel alır. Ayrıca her *j* eczanesinin nöbet sayısının n_j olarak belirlenmiş olduğunu varsaydık. Bu n_j değerleri ilgili bölgenin nöbet alt ve üst sınırları arasında yer alır ve (GM1-3) ve (GM1-4) kısıt kümelerini sağlar.

3.6.1 Genel Yapı ve İşleyiş

Dantzig-Wolfe ayrıştırmasının mümkün olduğu problemlerde, ayrıştırılmış problemin doğrusal gevşetilmiş en iyi sonucu orijinal formülasyondan elde edilecek doğrusal gevşetme sınırına göre daha iyi sonuç vermektedir (Barnhart vd.,1996). Bu özellik, dal-fiyat algoritmasında yaratılan düğümlerde daha kaliteli alt sınırlar elde edilmesini sağlamaktadır.

Dal-fiyat algoritmaları yapısal olarak dal-sınır algoritmalarına benzer olmak ile birlikte, düğümlerde alt sınırlar elde etmek için kullanılan problemin doğrusal gevşetmesini çözerken farklılıklar içermektedir. Dal-fiyat algoritmaları düğümlerdeki doğrusal gevşetme problemini kolon türetme algoritması ile çözmektedir.

Kolon türetme algoritmasının temel özelliği, problemi tek seferde çözmek yerine, karar değişkenlerinin küçük bir alt kümesi ile problemi çözmek ve sonrasında kademeli bir şekilde karar değişkenlerini probleme ekleyip tekrar çözmektir. Bu özellik, çok fazla sayıda karar değişkeni içeren problemleri çözmede büyük bir avantaj sağlamaktadır. Bu problemler için doğrusal gevşetme problemi çözüldüğünde, karar değişkenlerinin büyük bir kısmı optimal çözümde sıfır değeri almaktadır. Bu yöntem büyük problemler için verimsiz olmaktadır. Kolon türetme yöntemi kullanıldığında ise, her seferinde daha küçük bir problem çözülmüş olacaktır. Karar değişkenlerinin büyük bir kısmı dışarıda tutularak, sadece en karlı azaltılmış maliyete (reduced cost) sahip değişkenler probleme eklenecektir. Bu amaçla fiyatlandırma alt problemi kullanılmaktadır. Bu şekilde ilerlenerek, eklenecek değişkenin azaltılmış maliyeti karlı olmaktan çıktığı an algoritma sona erecektir.

Kolon türetme algoritmasının başlangıcında, ana problem için olurlu bir çözüm vermek gerekmektedir. Bu amaçla, rassal bir şekilde oluşturulan T günlük olurlu bir çizelge başlangıç kolonları olarak ele alınıp, ilk fiyatlandırma alt problemi bu başlangıç çözümünden elde edilecek fiyat bilgileri ile çözülecektir. Kolon türetme algoritması, tüm kolonların sadece bir alt kümesi ile çalıştırıldığı için çözülen problem kısıtlı ana problem olarak adlandırılmaktadır (KAP).

3.6.2 Ana Problem

ENÇ problemi ayrıştırılarak ana problem ve fiyatlandırma alt problemi elde edilmiştir. Ana problemde amaç tek günlük olurlu çizelgelerden planlama ufku kadarını nöbet kısıtlarını sağlayacak ve toplam maliyeti enazlayacak şekilde seçmektir. Fiyatlandırma alt probleminde ise amaç tek günlük olurlu çizelgelerden ana problemin amaç fonksiyonunu en çok iyileştirecek olanı seçmektir. Bu problem, ENÇ probleminin tek periyoda indirgenmiş halinin kısıtlarını taşımaktadır. Dolayısı ile fiyatlandırma problemi, ek kısıtları olan bir p-ortanca problemidir.

Ana problem aşağıda sunulmuştur:

Kümeler:

I: mahalleler kümesi

J: eczaneler kümesi

S : tek günlük olurlu çizelgeler kümesi

S_j : *j* eczanesinin nöbette olduğu tek günlük olurlu çizelgeler kümesi

Parametreler:

$$a_{ij}^{s} = \begin{cases} 1 & j\ eczanesi\ s\ çizelgesinde\ i\ müşterisine\ en\ yakın\ eczane\ ise \\ 0 & Diğer\ durumda \end{cases}$$

Karar değişkenleri:

λ_s : *s* çizelgesinin en iyi çözümde kullanıldığı gün sayısı

Amaç Fonksiyonu:

$$Min\ z = \sum_{s \in S} \lambda_s \sum_{i \in I} \sum_{j \in J} a_{ij}^{s} d_{ij} h_i \qquad \text{(AP1)}$$

Kısıtlar:

$$\sum_{s \in S} \lambda_s \geq T \qquad \text{(AP2)}$$

$$\sum_{s \in S_j} \lambda_s \leq n_j \qquad \forall j \qquad \text{(AP3)}$$

$$\lambda_s \in Z^+ \qquad \forall s \in S \qquad \text{(AP4)}$$

Amaç fonksiyonu (AP1) tek günlük çizelge maliyetlerinin bu çizelgelerin kullanılma sayısı ile çarpımlarının toplamını enazlamaktır. Kısıt kümesi (AP2) tek günlük çizelgelerden en az planlama ufkundaki gün sayısı kadar seçilmesini garantilemektedir. Enazlama problemi olması dolayısıyla bu kısıt her zaman eşitlik olarak sağlanacaktır. Kısıt kümesi (AP3) ise her eczaneye planlama ufku boyunca en fazla n_j defa nöbet atanmasını sağlamaktadır. Bu kısıt da her zaman

eşitlik olarak sağlanacaktır çünkü bir bölgedeki eczanelerin n_j parametrelerinin toplamı *T*'ye eşittir. Bu kısıtların eşitsizlik olarak yazılmalarının sebebi, fiyatlandırma alt problemindeki arama uzayını daraltmaktır.

Ana probleme girdi olarak ele alınan tek günlük çizelgeler aşağıdaki fiyatlandırma probleminin değişik parametre değerleri altındaki en iyi çözümleri olarak bulunacaktır.

3.6.3 Fiyatlandırma Alt Problemi

Bu bölümde fiyatlandırma alt problemini açıkladık.

Kümeler:

J_k: *k* bölgesinde bulunan eczaneler kümesi

Parametreler:

v : Ana problemdeki (AP2) kısıtı ile ilişkili ikili değişkenin en iyi değeri

u_j : Ana problemdeki (AP3) kısıt kümesi ile ilişkili ikili değişkenin en iyi değeri

Karar Değişkenleri:

$$x_{ij} = \begin{cases} 1 & i \text{ müşterisi } j \text{ eczanesine atanmış ise} \\ 0 & \text{Diğer durumda} \end{cases}$$

$$y_j = \begin{cases} 1 & j \text{ eczanesi açılmış ise} \\ 0 & \text{Diğer durumda} \end{cases}$$

Amaç Fonksiyonu:

$$Min\ F = \sum_{j \in J} \left(\left[\sum_{i \in I} x_{ij} d_{ij} h_i \right] - u_j y_j \right) - v \qquad \text{(FAP1)}$$

Kısıtlar:

$$\sum_{j \in J} x_{ij} \geq 1 \qquad \forall i \qquad \text{(FAP2)}$$

$$x_{ij} \leq y_j \qquad \forall i, j \qquad \text{(FAP3)}$$

$$\sum_{j \in J_k} y_j = 1 \qquad \forall k \qquad \text{(FAP4)}$$

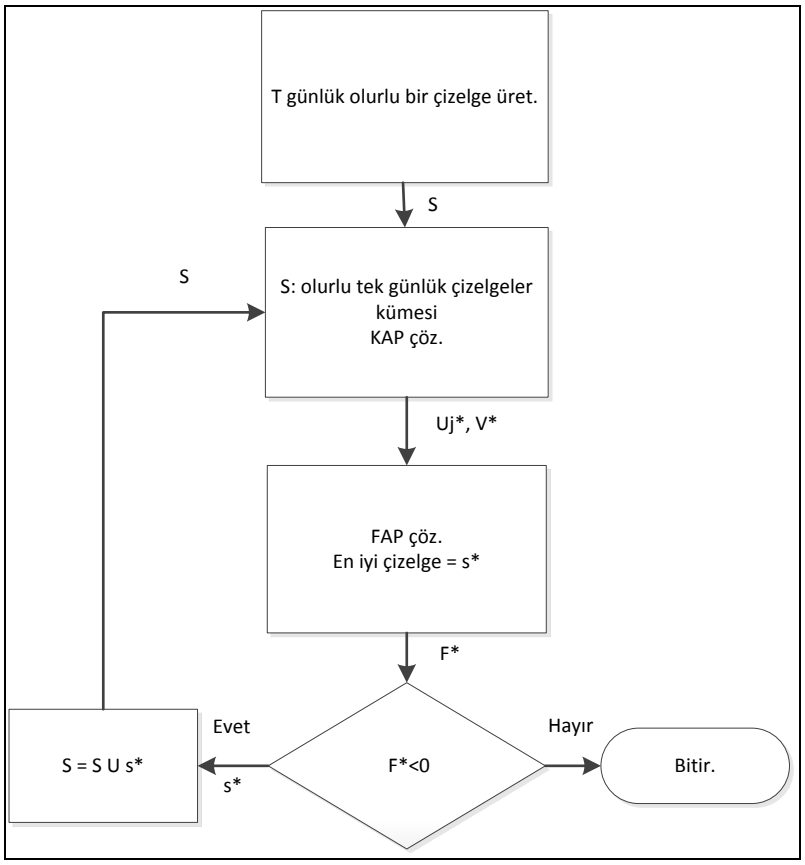
$$x_{ij} \leq 1 \quad x_{ij} \in R^+, \quad y_j \in \{0,1\} \qquad \text{(FAP5)}$$

Amaç fonksiyonunda, çizelge maliyetinden u_j değerlerinin nöbet atama karar değişkenleri ile çarpımlarının toplamı ve v değeri çıkarılmaktadır. u_j değerleri pozitif olmadıkları için nöbette olan her eczane için maliyet artmaktadır. Öte yandan v değişkeni negatif olmadığı için maliyeti azaltıcı etki yapmaktadır. Fiyatlandırma alt probleminde amaç fonksiyonu değeri en negatif olan çizelge bulunacak olup, amaç fonksiyonu değeri 0 veya pozitif olana kadar kolon türetme algoritması tarafından her iterasyonda çağırılacaktır. Kısıt kümeleri (FAP2), (FAP3) ve (FAP4) olurlu bir tek gün çizelgesi oluşturulmasını sağlamaktadırlar.

Kolon Türetme Algoritması (KTA)

Kolon türetme algoritmasının çalışma prensibi Şekil 2'de gösterilmektedir. FAP'nin en iyi amaç fonksiyonu değeri negatif olmayana kadar algoritma çalışmaktadır. FAP tarafından bulunan her bir çizelge olurlu çizelgeler kümesine eklenerek KAP yeniden çözülmektedir.

Kolon türetme algoritmasında, her döngüde çözülen fiyatlandırma alt problemini en iyi çözmek yerine sezgisel yaklaşımların kullanımına rastlanmaktadır (Barnhart vd.,1996). Sezgisel olarak bulunan azaltılmış maliyet karlı olmaktan çıktığı anda fiyatlandırma alt problemi tekrar optimal çözümü elde edecek şekilde çözülmektedir.



Şekil 2. Kolon türetme algoritmasının şematik gösterimi

3.6.4 Dallandırma

Kök düğümde tamsayı çözümler bulunamadığı takdirde kök düğümden dallanma yaparak dal-fiyat algoritmasına devam edilmesi gerekmektedir. Üzerinde dallandırmaya gittiğimiz değişkenler eczane çiftlerinin bulunan en iyi çözümde kaç defa birlikte nöbet tuttukları bilgisini taşımaktadır. λ_s^* , bir düğümde çözülen ana problem doğrusal gevşetmesindeki λ_s karar değişkenlerinin en iyi değerleri olarak tanımlanırsa, her bir eczane çifti için KAP'nin en iyi çözümünde birlikte nöbette oldukları gün sayısını aşağıdaki şekilde bulabiliriz:

$$\beta_{gh} = \sum_{s \in S_g \cap S_h} \lambda_s^* \qquad \text{(DY1)}$$

Tüm (g,h) ikililerinden $\left|\beta_{gh} - \lfloor \beta_{gh} \rfloor - 0{,}5\right|$ değeri en küçük olan ikili seçilir. Bu şekilde, tüm eczane ikilileri içerisinde birlikte nöbette oldukları gün sayısı en kesirli olan ikili seçilmiş olur.

İki çocuk düğüm aşağıdaki kısıtlardan her birinin sırasıyla (ve ayrı ayrı) KAP'ye eklenmesiyle elde edilir.

$$\sum_{s \in S_g \cap S_h} \lambda_s \leq \lfloor \beta_{gh} \rfloor \qquad \text{(DY2\_sol)}$$

$$\sum_{s \in S_g \cap S_h} \lambda_s \geq \lceil \beta_{gh} \rceil \qquad \text{(DY2\_sağ)}$$

Çocuk düğümlerden sol çocuk seçilerek derin öncelikli (depth-first) dallandırma yapılmaktadır. Eklenen kısıtlar dolayısı ile bir düğümdeki KAP olurlu çözüm vermeyebilir. Bu durumda, bu düğüm budanarak olurlu çözümü olan en derindeki düğüm ile dallandırmaya devam edilir. Bir diğer durumda ise, KAP'nin verdiği en iyi çözüm tamsayı bir çözüm olabilir. Bu durumda, bulunan en iyi üst sınır çözümü güncellenerek bu düğüm budanır. Olurlu düğümlerden en derinde yer alanı ile dallandırmaya devam edilir.

Dallandırma FAP'nin amaç fonksiyonunu da etkilemektedir. Eğer (DY2_sol) (veya (DY2_sağ)) kısıtı ile ilişkili ikili değişkenin en iyi değerini ρ_{gh} olarak tanımlarsak, FAP'nin amaç fonksiyonu sol çocuk için

$$F_{g,h} = \sum_{j \in J}\left(\left[\sum_{i \in I} x_{ij} d_{ij} h_i\right] - u_j y_j\right) - v - \rho_{gh}(y_g + y_h), \qquad \text{(DY3\_sol)}$$

sağ çocuk için ise

$$F_{g,h} = \sum_{j \in J}\left(\left[\sum_{i \in I} x_{ij} d_{ij} h_i\right] - u_j y_j\right) - v + \rho_{gh}(y_g + y_h) \qquad \text{(DY3\_sağ)}$$

olmaktadır.

Herhangi bir düğümde dallandırma yapıldığında, bu düğüme ait FAP'nin amaç fonksiyonu *F* olmak üzere, çocuk düğümlerin amaç fonksiyonu sol çocuk için

$F_{g,h} = F - \rho_{gh}(y_g + y_h)$, (DY4_sol)

sağ çocuk için ise

$F_{g,h} = F + \rho_{gh}(y_g + y_h)$ (DY4_sağ)

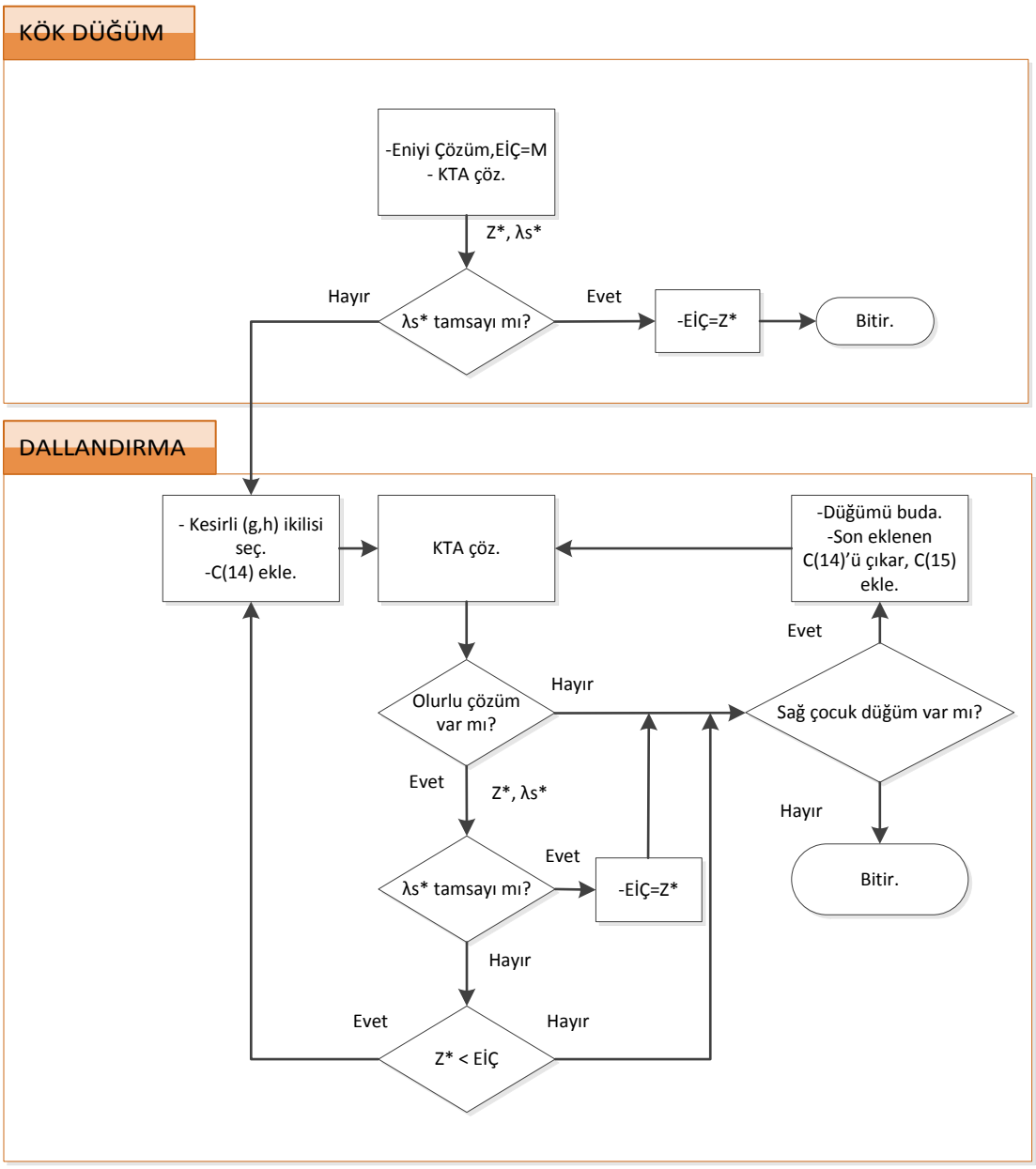
olarak güncellenecektir.

Önerilen dallandırma stratejisi ile birlikte KTA ele alındığında dal-fiyat algoritmasının çalışma prensibi aşağıdaki iş akış şeması ile gösterilmektedir. Aşağıdaki durumlardan biri oluştuğunda o düğüm budanmaktadır:

i. KAP olurlu değilse,
ii. KAP' nin en iyi çözümü tamsayı ise veya
iii. KAP' nin en iyi amaç fonksiyonu değeri üst sınırdan büyük veya ona eşitse.

Her zaman sol çocuk düğüm ile ilerlendiği için, düğümün budanması halinde sağ çocuk düğümü ile devam edilmektedir. Eğer sağ çocuk var ise son eklenen (DY2_sol) kısıtı yerine (DY2_sağ) kısıtı konularak sağ çocuk için KTA çalıştırılmaktadır. Eğer tüm düğümler budandı ise algoritma sona erer.

Bir düğüm için yukarıdaki durumlar oluşmadığı takdirde KTA'nın bulduğu en iyi çözümdeki karar değişkenleri işlenerek eczane çiftlerinin kaç kere birlikte nöbette oldukları bulunur. Belirtilen ölçüde en iyi olan çift seçilerek (DY2_sol) kısıtı eklenir ve KTA çalıştırılır.



Şekil 3. ENÇDFA'nın iş-akış şeması olarak gösterimi

* C(14): DY2_sol, C(15): DY2_sağ

3.6.5 Gelişmiş ENÇDFA

Dal fiyat bölümünde öncelikle ENÇ problemi için geliştirmiş olduğumuz temel dal-fiyat algoritması anlatılmıştır. Dal-fiyat algoritmaları üzerinde çeşitli noktalarda iyileştirme olanakları mevcuttur. Bu noktaları şu şekilde özetlemek mümkündür:

i. *Başlangıçta tek günlük çizelgelerin seçimi*: Kök düğümde çözülen KTA ilk olarak T kadar tek günlük çizelge ile başlar. Mevcut algoritmada bu tek günlük çizelgeler T günlük olurlu bir çizelge oluşturacak şekilde rassal olarak seçilmiştir. Bu noktada, KTA'nın daha iyi kolonlar ile başlamasının ikili değişkenlerin başlangıçtaki kalitesini arttırması ve KTA'nın performansına olumlu yönde katkı yapması beklenebilir.

ii. *Fiyatlandırma alt probleminin çözümü*: Fiyatlandırma alt probleminin çözümünün zor olduğu durumlarda, kesin çözüm yöntemi yerine sezgisel bir algoritma kullanılabilir. Sezgisel fiyatlandırma algoritması (SFA) azaltılmış maliyeti pozitif (negatif) olmayan kolonlar bulduğu sürece kullanılabilir. SFA'nın kolon bulamadığı durumda kesin çözüm yöntemine geçilerek KTA'ya devam edilir. Bu yöntemde, kolonların bir kısmı kesin çözüm yöntemine göre daha kolay bir yöntem ile bulunmak ile birlikte AP'nin en iyi sonucuna ulaşmak için daha fazla kolon türetmek gerekebilir. Bu yüzden, toplamda dal-fiyat algoritmasının performansına etkisinin deneyler ile test edilmesi gerekmektedir.

iii. *Kolon yönetimi*: KTA'nın ilerleyen aşamalarında büyük pozitif azaltılmış maliyete sahip kolonların silinmesi KAP'nin çözüm süresini hızlandıracaktır. Bunun yanında FAP ile tek bir kolon yerine birden fazla kolon üretilip bu kolonların iyi bir alt kümesinin AP'ye eklenmesi düşünülebilir.

iv. *KTA'nın çözümünün erken sonlandırılması*: Herhangi bir düğümde AP'nin en iyi çözümünün elde edilmesi dal-fiyat algoritması için kaliteli sınırlar elde edilmesini sağlamaktadır. Ancak, bazı problemlerde AP için en iyi değeri elde etmek çok sayıda KTA döngüsü gerektirebilir. Böyle durumlarda, AP'nin en iyi çözümünü elde etmeden KTA algoritması erken durdurulabilir. En son döngüdeki KAP en iyi değeri ve FAP'nin bulduğu azaltılmış maliyet değeri kullanılarak AP için bir alt sınır elde edilebilir. Problemimiz için aşağıdaki eşitsizlik geçerli olacaktır:

$$z^* \geq z^{KAP} + TF^*$$

Yukarıdaki eşitsizlikte, z^* AP'nin KTA sonundaki en iyi değerini, z^{KAP} KTA'nın durdurulduğu döngüdeki KAP'nin en iyi değerini, F^* ise FAP'nin bulduğu en iyi azaltılmış maliyeti vermektedir. T ise problemdeki periyot sayısını temsil etmektedir. F^* değeri negatif olduğu için ve AP'nin en iyi çözümünde $\sum_{s\in S} \lambda^*_s = T$ olacağı için yukarıdaki eşitsizlik sağlanacaktır. Böylece, KTA' nın herhangi bir döngüsünde AP için bir alt sınır elde etmek mümkündür.

v. *Dallandırma ağacı tasarımı*: Mevcut dallandırma ağacını değişik dallandırma ve arama yöntemleri ile test edip iyi tasarımlar elde etmek mümkündür.

Yukarıda bahsedilen iyileştirme olanaklarından fiyatlandırma alt probleminin çözümü ile ilgili kısım mevcut problemimiz için diğerlerine göre daha fazla önem teşkil etmektedir. Bunun sebebi, fiyatlandırma problemimizin bir p-ortanca problemi olması ve çözümünün zor olmasıdır. Dolayısı ile iyileştirme çalışmalarımız bu konuda yoğunlaşmıştır.

3.6.6 Fiyatlandırma Alt Problemi Sezgiselleri

Fiyatlandırma Alt Problemini kesin çözmek yerine sezgisel olarak çözmek üzere geliştirdiğimiz üç algoritmayı açıkladık.

Fiyatlandırma Alt Problemi Sezgisel-1 (FAPS1)

İlk olarak, iyi kolonları hızlı bir şekilde bulacak bir sezgisel geliştirilmiştir. Bu sezgiselde, herhangi bir eczanenin açık olduğu durumda ona kesin atanacak müşterilerin kat edeceği ortalama mesafe bilgisi kullanılmıştır. Her eczane için bu müşterilerin kümesi aşağıdaki gibi hesaplanmaktadır:

$$D_j = \{i \in I \mid d_{ij} \leq d_{ij'} \; \forall j' \in J, r_j \neq r_{j'}\}$$

Bu kümeler kullanılarak her eczane için c_j ölçüsü hesaplanmaktadır.

$$c_j = \left[\sum_{i \in D_j} d_{ij} - u_j\right] \Big/ \left[\sum_{i \in D_j} h_i\right]$$

Her eczanenin sahip olduğu c_j değeri göz önüne alınarak her bölgeden seçilecek eczane şu şekilde bulunmaktadır:

$$j_k^* = argmin_{j \in J_k} c_j$$

Herhangi bir eczanenin seçilmesi durumunda kolonun maliyeti eczane ile ilgili olan ikili değişken değeri kadar artacaktır. Çizelgenin gerçek maliyeti hesaplanırken talep ağırlıklı uzaklıkların toplamı ele alınmasına rağmen bu ölçüde müşterilerin servis almak için kat edecekleri mesafelerin toplamı maliyet olarak düşünülmüştür. Bu maliyetten ikili değişken değeri çıkarılmış ve toplam müşteri sayısına bölünmüştür. Böylece, j eczanesi açıldığında müşteri başına ortalama ne kadar maliyet oluşacağı kestirilmeye çalışılmıştır. Her bölgeden, bu maliyeti en düşük olan eczane seçilmiştir. Ortalama maliyet kullanılarak, müşterilere yakın olmayan eczanelerin seçilmesi önlenmiştir. Aksi takdirde, c_j değeri küçük olan eczaneler genellikle D_j kümesi çok küçük olan eczaneler olacaktır.

Fiyatlandırma Alt Problemi Sezgisel-2 (FAPS2)

Bu sezgiselde, önceden belirlenmiş bir bölge sırasında sıra ile eczane seçimi yapılmaktadır. Bölge sırası dal-sınır algoritmasında uygulanan sıralama yönteminden alınmıştır. Ele alınan bölge sırası şu şekilde olsun: $\{k_1, k_2, \dots, k_K\}$. Buna ek olarak, $\{j_1 \in J_{k_1}, j_2 \in J_{k_2}, \dots, j_n \in J_{k_n}\}$ şeklinde bir yarı çizelge olduğunu varsayarsak, bir sonraki sıradaki bölgeden seçilecek eczane, $j_{n+1} \in J_{k_n+1}$, aşağıdaki gibi belirlenmektedir:

$J' = \{j_1, j_2, \dots, j_n, j_{n+1}\}$ olsun.

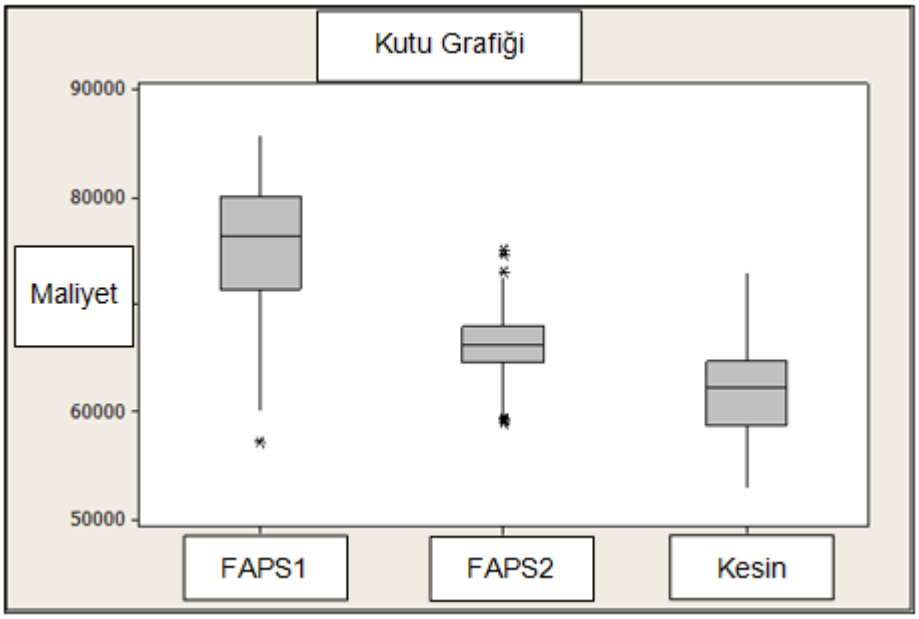
$$j_{n+1} = argmin_{j \in J_{k_{n+1}}} \left\{ \sum_{j \in J'} \sum_{i \in I} x_{ij} d_{ij} h_i - u_j \right\}$$

Bu şekilde bulunan eczane, sıradaki bölgede bulunan ve önceden belirlenen eczaneler ile birlikte toplam maliyeti en az yapacak eczane olmaktadır.

FAPS2 ile FAPS1' in buldukları kolonların kalitesini karşılaştırmak için, bir örnek üzerinde bu iki sezgiselin buldukları kolonların gerçek maliyetleri karşılaştırılmıştır.

Şekil 4'de görüldüğü üzere FAPS2 algoritması FAPS1'e göre oldukça iyi performans göstermektedir. En sağdaki kutu grafiği ise kesin çözüm yöntemi ile bulunan kolonların maliyetlerinin dağılımını göstermektedir. FAPS2' nin bulduğu kolonların maliyetlerinin orta

noktası, kesin çözüm yönteminkinden çok uzakta değildir. Bu sebeple, kesin çözüm yönteminden önce FAPS2' nin kullanımının, çözüm süresinde iyileştirme yapacağı düşünülmüştür.



Şekil 4. FAP çözüm yöntemlerinin buldukları kolonların maliyet dağılımı

Fiyatlandırma Alt Problemi Melez Sezgisel (FAPMS)

Bu sezgiselde, FAPS1 ve FAPS2'nin iyi yönlerinden faydalanılmak istenmiştir. FAPS1 hızlı çalışmakta fakat FAPS2 kadar kaliteli kolonlar bulamamaktadır. Bu sebeple, KTA'nda ilk olarak FAPS1'in kullanılıp daha sonrasında FAPS2'ye geçilmesinin iyi bir yöntem olacağı düşünülmüştür. Eğer FAPS1 azaltılmış maliyeti negatif bir kolon türetemez ise veya belirli sayıda döngü boyunca KAP amaç fonksiyonu değerini geliştiremez ise, FAPS2 algoritması kullanılmaya başlanmaktadır. FAPS2 kullanılırken, KAP amaç fonksiyonu değerini iyileştiren bir kolon bulunduğunda FAPS2'ye geri dönülmektedir. Benzer ilişki FAPS2 ve kesin çözüm yöntemi arasında da bulunmaktadır. Kesin çözüm yöntemi ile kolonlar bulunurken KAP amaç fonksiyonu değeri iyileştiğinde tekrar FAPS1'e geri dönülmektedir. Kesin çözüm yönteminin bulduğu kolon negatif olmadığı an algoritma sona ermektedir. FAPMS algoritmasının işleyişi aşağıdaki gibidir:

Algoritma 20. Fiyatlandırma Alt Problemi Melez Sezgisel (FAPMS) Algoritması

Adım 1: FAPS1 çöz. Üretilen kolonu KAP'ne ekle ve KAP çöz. Eğer KAP amaç fonksiyonu değeri son Q döngü boyunca iyileşmediyse, Adım 2'ye git.

Adım 2: FAPS1 çöz. Üretilen kolonu KAP'ne ekle ve KAP çöz. Eğer KAP amaç fonksiyonu değeri

iyileşirse, Adım 1'e git. Eğer KAP amaç fonksiyonu değeri son Q döngü boyunca iyileşmediyse, Adım 2'ye git.

Adım 3: FAP'ni kesin çözüm yöntemi ile çöz. Üretilen kolonu KAP'ne ekle ve KAP çöz. Eğer KAP amaç fonksiyonu değeri iyileşirse, Adım 1'e git.

FAPS1, FAPS2 ve FAPMS algoritmaları ayrı ayrı dal-fiyat algoritmalarında kullanılmış ve değişik örnekler üzerinde test edilmiştir. Bu testler sonucunda FAPMS algoritması ile çalıştırılan dal-fiyat algoritmasının daha iyi performans gösterdiği görülmüş olup, sonraki deney sonuçlarında bu algoritma kullanılmıştır. Bir takım ön deneyler de FAPMS algoritmasındaki Q parametresini belirlemek için yapılmıştır. Bu deneyler sonucunda, Q parametresinin değerinin 25 olarak alınmasına karar verilmiştir.

3.6.7 Üst Sınır Bulma

Dal-fiyat algoritmasında arama yaparken üst sınır değerleri elde etmek budanacak düğüm sayısı üzerinde olumlu etki yapmaktadır. Kaliteli üst sınırlar elde edilebilirse, bir düğümdeki alt sınır değeri ile karşılaştırıldığında üst sınır değeri alt sınır değerinden küçük ise o düğüm budanabilmektedir. Bu amaçla, dal-fiyat algoritmasının her düğümünde çağırılmak üzere hızlı çalışacak bir üst sınır algoritması geliştirilmiştir. Bu algoritma, AP'nin en iyi çözüm bilgisini girdi olarak almakta ve T günlük olurlu bir çizelge vermektedir. Bu çizelgenin T' günü aşağıdaki gibi oluşturulabilir:

$$T' = \sum_{s \in S} \lfloor \lambda_s^* \rfloor$$

Sonrasında, T-T' günlük problem için olurlu bir çözüm bulmak gerekmektedir. Bunun için, eczanelerin nöbet sayıları güncellendikten sonra dal-sınır algoritması için geliştirilen üst sınır algoritmalarından biri kullanılabilir. Ancak, hızlı olması bakımından mevcut algoritmada T-T' günlük çizelge rassal bir şekilde oluşturulmaktadır.

ÜSBA'nın performansı T' / T oranına bağlıdır. Bu oranın yüksek olduğu düğümlerde iyi bir üst sınır elde edilebilir. ÜSBA'nın performansını ölçebilmek için bir takım ön deneyler yapılmıştır. Yapılan ön deneylerde, ÜSBA'nın kök düğümde bulunan üst sınır değerini geliştirme sayısı çok az kalmıştır. Ancak, algoritma hızlı çalıştığı için üst sınır değerini geliştiremediği durumlarda dahi çözüm süresinde önemli bir fark yaratmamaktadır. Diğer taraftan, üst sınır algoritması ile bulunabilecek tek bir üst sınır çözümü dahi çözüm süresine önemli iyileştirmeler getirebilir. Özellikle en büyük problemde, kök düğümde üst sınır elde etmediğimiz için ÜSBA'nın kullanımı daha önemli olmaktadır.

3.7 Tabu Arama Algoritması

Proje kapsamında ilk olarak Tabu Arama (TA) algoritması geliştirdik. Bu algoritmanın iki farklı türevini BM1 ve GM1 problemlerini çözmek için kullandık. Bu bölümde TA algoritmasının genel işleyişi, komşuluk yapısı ve arama stratejisini aktardık.

Genel işleyişi

TA algoritmasında iki karar verilmektedir. İlk olarak, planlama ufkundaki her bir periyotta açık olacak eczanelere karar verilir. İkinci olarak ise, müşterileri planlama ufku boyunca kat edecekleri talep ağırlıklı uzaklıkları en azlayacak şekilde açık olan eczanelere atamasına karar verilmektedir.

TA algoritması tek ve çok nöbetli problemlerde benzer şekilde çalışmaktadır. Ancak, çok nöbetli problem için bölge kavramını ve her eczanenin birden fazla nöbet tutabilmesini ifade edebilmek için bazı değişiklikler yapılmaktadır. TA algoritmasının ana adımları tek (çok) nöbetli problem için Algoritma 21'de belirtilmiştir.

Komşuluk Yapısı

TA algoritmasında, tek nöbetli problem için *takas* ve *taşı* komşuluk yöntemleri kullanırken, çok nöbetli problem için *takas* ve *ekle-çıkart* komşuluk yöntemleri kullanılmaktadır. Bu komşuluk

yöntemleri proje konusu probleme özgü olarak geliştirilmiş yöntemlerdir. Ambulans yerleşim problemini çözmek için Gendreau vd. (1997) ve Doerner vd. (2005) tarafından önerilen TA algoritmalarında bir ambulansın yer değiştirdiği komşuluk yapıları tanımlanmıştır. Gendreau vd. (2001) ise aynı problem için mevcut ambulansların bir alt kümesinin yer değiştirdiği bir komşuluk yapısı tanımlamıştır.

Algoritma 21. ENÇ Problemi için Tabu Arama Algoritması

(dış_döngü ≤ maksimum_dış_döngü) **doğru iken devam et**
  Eğer dış_döngü = 1
    Başlangıç çözümünü oluştur
  Değilse
    Yeni başlangıç çözümü oluştur
  Eğer sonu
  (iç_döngü ≤ maksimum_iç_döngü) **doğru iken devam et**
    En iyi takası ara
    En iyi taşımayı (ekle-çıkart) ara
    Eğer $F_{eniyi\_aday\_takas} < F_{eniyi\_aday\_taşı\ (ekle\text{-}çıkart)}$
      En iyi takası uygula
    Değilse
      En iyi taşımayı (ekle-çıkart) uygula
    Eğer sonu
    Frekansları güncelle
    Tabu statüsünü güncelle
    Eğer $F_{mevcut} < F_{eniyi}$
      En iyi çözümü güncelle
    Eğer sonu
  Tekrarla sonu (iç_döngü)
Tekrarla sonu (dış_döngü)

TA algoritması için olurlu çözümler ile ilerlemek bir zorunluluk değildir. Örneğin Gendreau vd. (1997) çözümlerinin kapsama kısıtlarını sağlamama durumuna izin vermiş ancak bu şekildeki çözümlerin değerlerini belirlerken kısıtları ihlal etme derecelerini de göz önüne almışlardır. Bilgin ve Azizoğlu (2009) esnek imalat sistemleri için makina uçlarını da göz önüne alan karmaşık bir atama problemi üzerinde çalışmışlardır. Bu problemde olursuz bir çözüm elde ettikten sonra tekrar olurlu bir çözüme geçmek zor olacağı için sadece olurlu çözümler üreten bir komşuluk yapısı tanımlamışlardır. Proje konusu problem için olurlu çözüm elde etmek kolaydır (Eczanelerin nöbet sayılarını göz önüne alarak yapılan her türlü atama olurlu bir çözüm olacaktır). Bu sebeple geliştirilen komşuluk yöntemleri algoritmanın her adımında olurlu bir çözüm elde etmeyi garanti etmektedir.

Takas algoritması hem tek hem de çok nöbetli problemlerde uygulanmaktadır. Takas algoritmasında, iki eczanenin nöbet günleri birbiri ile değiştirilir. Bir eczane seçilir ve farklı bir günde nöbetçi olan tüm diğer eczaneler ile olası takaslar belirlenir. Her bir takasın maliyeti hesaplanır. Eğer iki eczanenin nöbet günlerinin takası bilinen en iyi takas değerini geliştiriyorsa, yeni çözüm aday çözüm olur. Tüm olası takaslar kontrol edildikten sonra, içlerindeki en iyisi belirlenir ve en iyi aday çözüm olarak seçilir. Çok nöbetli problemde, takas algoritması yine bir eczane için olası takaslara bakar, fakat sadece kendi bölgesi içerisindeki eczaneler ile olanlar kabul edilebilir. Planlama ufkunun her gününde her bölgeden sadece bir eczane nöbette olduğu için, eğer başka bir bölgeden bir eczane ile takas yapılırsa olursuz bir çözüm elde edilir. Bu sebeple, algoritma sadece aynı bölgedeki eczaneler arasındaki takaslara bakar.

Taşı algoritması sadece tek nöbetli problemde uygulanmaktadır. Taşı algoritmasında, nöbette olan bir eczane seçilir ve planlama ufkunun diğer bir gününe taşınır. Diğer bir deyişle, seçilen bir eczane gerçekte açık olduğu günden farklı bir günde açık olduğunda çözümün kalitesinin iyileşip iyileşmeyeceği kontrol edilir. Tüm olası taşımalar kontrol edildikten sonra, en çok iyileşme gösteren en iyi aday taşıma belirlenir. Çok nöbetli problemde, eğer bir eczaneyi başka bir güne taşırsak, eczanenin başlangıçta atandığı günde hiç açık eczane olmayıp eczanenin taşındığı günde iki eczane açık olacaktır. Bu yüzden, ekle-çıkart algoritması geliştirilmiştir.

Ekle-çıkart algoritması sadece çok nöbetli problemde uygulanmaktadır. Ekle-çıkart algoritması için, tüm bölgelerdeki nöbet sayıları için alt ve üst sınırlar hesaplanmıştır. Planlama ufku boyunca, bir bölgedeki eczanelerin ortalama nöbet sayıları toplam gün sayısının o bölgedeki

eczane sayısına bölünmesiyle elde edilmiştir. Üst ve alt nöbet sayıları, ortalama nöbet sayılarının tam sayı olmadığı durumlarda sırasıyla üste ve alta yuvarlanarak hesaplanmıştır. Ekle-çıkart algoritmasını bir bölgeye uygulayabilmek için, o bölgedeki nöbet sayısı üst ve alt sınırlarının birbirinden farklı olması gerekmektedir. Aksi durumda, algoritma olursuz bir çözüm elde edecektir. Algoritma ilk önce alt nöbet sayısında nöbet tutan bir eczane (j_1) seçer. Daha sonra, bu eczane ile aynı bölgede olan ve üst nöbet sayısında nöbet tutan bir eczanenin (j_2) nöbet tuttuğu bir gün (t) belirlenir. Bu günde j_1 eczanesi nöbetçi olarak atanır ve j_2 eczanesinin nöbeti kaldırılır. Yeni çözümün değeri belirlenir. Olası tüm eczane ikilileri ve günler için çözümler incelenir ve en iyi çözüm ekle-çıkart algoritmasının çözümü olur.

TA'ya Özgü Kararlar

Daha iyi çözümler ararken, yerel optimal çözümlerden kurtulmak için tüm arama yöntemlerinde tabu süresi (tabu tenure) kullanılır. Yakın geçmişte ziyaret edilen çözümler için bir tabu süresi atanır ve algoritmanın çözüm uzayının küçük bir kısmındaki çözümleri araması engellenmiş olur. Tek nöbetli problemde, günler ve eczaneler için tabu listesi tutulmaktadır. Çok nöbetli problemde, ek olarak bölgeler için de tabu listesi tutulur. Bu sayede belirli bir günde değişiklik yapılmış ise gün için belirlenen tabu süresi boyunca bu günde bir değişiklik yapılmasına izin verilmez. Benzer şekilde tabu süreleri değişiklik yapılan eczaneler ve bölgeler için de dikkate alınır. Tabu yıkma kriteri olarak bilinen en iyi çözümden daha iyi bir çözüm elde etme kullanılmıştır.

İçteki her bir döngü ve her arama türü için, aday çözümlerin tabu durumu kontrol edilir. Eğer aday çözümler tabu listesinde ise, bu çözümler tabu yıkma ölçütü (aspiration criterion) karşılamadıkları sürece en iyi aday çözüm olarak gösterilmezler. Tabu yıkma ölçütü, algoritmanın tabu listesinde olsa dahi çok iyi amaç değerine sahip olan çözümlere ulaşmasını sağlar. TA algoritmasında, eğer tabu listesinde olan gün, eczane veya bölgeyi içeren bir çözüm bilinen en iyi çözümden daha iyi bir amaç değeri veriyorsa, bu çözümün en iyi aday çözüm olarak gösterilmesine izin verilir.

Önceki adımları izleyerek, her bir arama türü için en iyi aday çözümler komşu arama algoritması ile belirlenmiştir. Sonrasında, her arama yönteminden en iyi iki alternatif karşılaştırılır ve içlerinde daha iyi olan içteki döngü için en iyi çözüm olarak seçilir. Seçilen en iyi çözüm, bir sonraki iç

döngü için başlangıç çözümü olur ve bu çözümün komşu çözümleri aranır. İç döngü sayısı maksimum değerine ulaşıncaya kadar bu süreç devam eder.

Her bir iç döngüden sonra, yeni bulunan çözümün bilinen en iyi çözümden daha iyi olup olmadığı kontrol edilir. Eğer daha iyi ise, bulunan en iyi sonuç güncellenir ve yeni çözüm ilerleyen döngülerdeki karşılaştırmalarda en iyi çözüm olarak kullanılır.

TA algoritmasında, yoğunlaşma (intensification) ve çeşitlendirme (diversification) stratejileri kullanılmıştır. İlk olarak başlangıç çözümü oluşturulup, takas, taşı, ve ekle-çıkart yöntemleri ile arama güçlendirilmiştir. Belirli bir döngü sayısından sonra, *maksimum_dış_döngü*, arama uzayını çeşitlendirmek için yeni bir başlangıç çözümü oluşturulur. Bu teknik, yerel optimal çözüme takılıp kalmamak ve arama uzayının daha geniş bir kısmını aramak için kullanılır. Bununla birlikte, tamamen yeni veya rassal bir çözümden başlamak da tercih edilmez. Bu sebeple daha önce ziyaret edilen çözümlerle ilgili bilgiler tutulur ve bu çözümlerden uzak başlangıç çözümleri oluşturulmaya çalışılır. Rolland vd. (1997) p-ortanca problemi için geliştirdikleri TA yönteminde bir aday yerin incelenen çözümlerde olma frekansını tutmuş ve çeşitlendirme adımında yeni bir çözüm oluşturulurken yüksek frekanslı yerleri daha az tercih edilecek şekilde cezalandırmıştır. Bilgin ve Azizoğlu (2009) bir esnek imalat sisteminde işlerin makinalara atanması problemi için geliştirdikleri TA yönteminde, bir işin çözümlerde makinalara atanma frekansını tutmuş ve sıkça ataması yapılan işlerin getirilerini atanma sıklıkları kadar azaltarak cezalandırmıştır. Geliştirilen algoritmada aynı günde açık olan eczane ikililerinin frekansının kaydı tutulmaktadır. Yeni başlangıç çözümü oluşturulurken, o zamana kadar değerlendirilen çözümlerde yüksek frekansa sahip olan eczane ikilileri planlama periyodunun farklı günlerine atanır. Algoritmanın birinci kısmında, aynı günde açılma frekansı en yüksek olan eczaneler bulunur. İkinci kısımda, bu eczaneler farklı günlere atanır. Yüksek frekanslı tüm eczaneler farklı günlere atandıktan sonra, kalan eczanelerin atamaları boş günlere eczane sırası izlenerek yapılır. TA algoritmasının ilk kullanacağı çözüm Algoritma 5 (BM1Üst) ve Algoritma 6 (GM1Üst) ile oluşturulur.

Her bir dış döngü için, iç döngü sayısı maksimuma ulaştığında, yeni çözüm sezgiseli çalıştırılır ve yeni bir çözüm elde edilir. İçteki döngüler boyunca, aynı günde açık olan eczanelerin frekansı tutulur. Yeni başlangıç çözümü algoritması bu frekansları kullanır ve sıklıkla aynı günde açılma

eğilimi olan eczaneleri planlama ufkunun farklı günlerine atar. Böylece, yeni ve farklı bir çözüm elde edilir. Bu yeni çözüm, arama uzayının farklı bölümlerini arayabilmemizi mümkün kılar.

Maksimum döngü sayısı ve iyileşme yapılamayan döngü sayısı algoritmayı durduran ölçütlerdir. Maksimum döngü sayıları, hem içteki hem de dıştaki döngüyü sınırlar. Böylece, önceden belirlenmiş bir döngü sayısı elde edildikten sonra algoritma durur. İkinci olarak ise, amaç fonksiyonunda iyileşme sağlanılamayan döngü sayısı önceden belirlenmiş bir sayıya ulaştığında algoritma durur. Bu tip durdurma koşulu içteki döngüler için kullanılır ki böylece algoritma iyileştirme sağlamayan yönlerde arama yapmayı atlayıp arama uzayının diğer bölgelerine hareket eder. Durdurma koşulu sağlandıktan sonra, TA algoritması durur ve bulunan en iyi çözümü raporlar.

3.8 Değişken Komşuluk Arama Algoritması

Bu bölümde Eczane Nöbet Çizelgeleme Problemi (ENÇP) için en iyi çözüme yakın sonuçlar elde etmek amacıyla Temel Değişken Komşuluk Arama (TDKA) sezgiseli ve büyük boyutlu problemler için Değişken Komşuluk Ayrıştırma Arama (DKAA), Değişken Komşuluk Kısıtlama Arama (DKKA) sezgisellerini önerdik. TDKA, DKAA ve DKKA sezgiselleri için iyi bir başlangıç çözümü elde etmek amacıyla Azaltılmış Değişken Komşuluk Arama (ADKA) sezgiselini geliştirdik.

ENÇP *p*-ortanca problemine amaç fonksiyonu ve kapasitesiz aday tesisler bakımından benzerlik göstermektedir. Ayrıca, Ağlamaz (2011) ENÇP'ni *p*-ortanca problemine indirgeyerek NP-Zor bir problem olduğunu ispatlamıştır. Hansen ve Mladenoviç (1997) *p*-ortanca problemine Değişken Komşuluk Arama (DKA) sezgiselini uygulamışlar ve DKA sezgiselinin TSP kütüphanesinin ortalama ve büyük boyutlu problemlerinde ısrarla daha iyi sonuçlar verdiğini görmüşlerdir. Hansen ve Mladenoviç (2001a) Değişken Komşuluk Ayrıştırma Arama (DKAA) sezgiselinin orta boyutlu problemler için DKA sezgiselinе göre her zaman iyi sonuçlar vermediğini ancak büyük boyutlu problemler için çok faydalı olabileceğini göstermişlerdir.

Bu bölümde ENÇP için Temel DKA (TDKA) sezgiseli geliştirilmiştir. Komşuluk yapıları, yerel arama metodu ve durdurma koşulları tanımlanmıştır. İyi bir başlangıç çözümü elde etmek için yerel arama adımını atlayan Azaltılmış DKA (ADKA) sezgiseli önerilmiştir. Büyük boyutlu problemleri çözmek için TDKA sezgiselinin iki türevi tanımlanmıştır. İlk tür çok daha küçük

ayrıştırılmış problemleri çözen DKAA sezgiselidir. İkinci tür ise yerel aramayı kısıtlanmış bir arama uzayında yapan ve ilk olarak bu çalışmada tanımlanan Değişken Komşuluk Kısıtlama Arama (DKKA) sezgiselidir.

3.8.1 Temel Değişken Komşuluk Arama

Olurlu bir çözüm olan z, $K \times T$ boyutunda bir matris olarak tanımlanmaktadır, $z \in \mathbb{N}^{K \times T}$, ve z çözümünün z_{kt} elemanı k bölgesinde t gününde nöbetçi olan eczanenin indisini ifade etmektedir. Dikkat edilirse, $z_{kt} = \sum_{j \in J_k} j \times y_{jt}$, burada J_k, k bölgesinde bulunan eczanelerin kümesini ifade etmektedir. Olurlu çözümlerin kümesi Z, $Z = \{z^1, z^2, \dots\}$, olarak ifade edilmektedir.

ENÇP için geliştirilmiş olan TDKA algoritması Algoritma 22'de verilmiştir. Başlangıç adımında, olurlu bir başlangıç çözümü şu şekilde oluşturulur: her bir j eczanesine kendi bölgesindeki ilk boş günden başlanarak ardışık n_j adet nöbet atanır. Daha sonra, temel adıma geçmeden önce başlangıç çözümünü iyileştirmek için sonraki bölümlerde açıklanacak olan Azaltılmış DKA (ADKA) algoritması uygulanır.

Bir olurlu çözümden diğerine değiştirme (swap) algoritması kullanılarak geçilir. Bir $z \in Z$ çözümünü düşünelim. Bir değiştirmede (swap), bir bölge, k olsun, ve iki farklı eczanenin nöbetçi olduğu iki gün seçilir, t_1 ve t_2 olsun, öyle ki $z_{kt_1} \neq z_{kt_2}$. z çözümünün z_{kt_1} ve z_{kt_2} elemanlarının yerleri değiştirilerek yeni çözüm z' elde edilir. Çok değiştirmenin olduğu durumlarda, her bir değiştirme için farklı bir bölge seçilir.

İki olurlu çözüm, z^1 ve z^2, arasındaki uzaklık uygulanan değiştirmelerin sayısıyla ölçülür. Bunu saymak için yeni bir operatör, $\ominus$, şu şekilde tanımlanmıştır:

$$z^1 \ominus z^2 = \frac{\sum_{t=1}^{T} |OD_t^1 \setminus OD_t^2|}{2},$$

burada OD_t^s kümesi $z^s, s = 1,2$ çözümünde t gününde nöbetçi olan eczaneler kümesini göstermektedir. Bir değiştirme (swap) çözümün iki ayrı kolonunda değişiklik yaptığı için çift saymayı önlemek adına eşitlik 2 'ye bölünmüştür. Komşuluk yapıları N_k, $k = \{1, \dots, k_{max}\}$ aşağıda tanımlanan uzaklık metriği ρ kullanılarak oluşturulur,

$\rho(z^1, z^2) = z^1 \ominus z^2 = z^2 \ominus z^1 = k, \forall z^1, z^2 \in Z$

burada z^1 ve z^2 olurlu olan Z uzayından iki çözümdür.

Algoritma 22. TDKA Algoritması

Başlangıç Adımı: Aramada kullanılacak komşuluk yapılarını, N_k, $k = \{1, \dots, k_{max}\}$, seç, başlangıç çözümü (çizelgesi) z 'yi oluştur ve durdurma koşulunu belirle.

Temel Adım: Aşağıdaki adımları durdurma koşulu sağlanıncaya kadar tekrarla,

(1) $k = 1$ olsun,
(2) $k = k_{max}$ oluncaya kadar aşağıdaki adımları tekrarla,
    (a) *Sallama:* z çözümünün k. komşuluğundan rastgele z' çözümünü oluştur $(z' \in N_k(z))$,
    (b) *Yerel Arama:* z' çözümünde yerel en iyi çözümü, z", değiştirme algoritması kullanarak bul,
    (c) *Hareket:* Eğer $f(z") < f(z)$ ise, mevcut çözümü değiştir $(z = z")$, ve aramaya birinci komşuluktan devam et N_1, $(k = 1)$, aksi takdirde, komşuluğu bir birim arttır $(k = k + 1)$.

Algoritmanın temel adımı $k = 1$ alarak başlar ve $k = k_{max}$ olana kadar devam eder, burada k_{max} parametresi $k_{max} \leq K_2$ koşulunu sağlar ve K_2 iki ya da daha fazla eczane içeren bölgelerin sayısıdır. Değiştirme yapabilmek için bir bölgede en az iki farklı eczane olması gerektiği için K yerine K_2 kullanılır. k_{max} parametresi için üç farklı seviye tanımlanmış ve test edilmiştir.

Rastgele çözümler sallama adımında 2(a) Algoritma 23 ile oluşturulur. Öncelikle, z çözümünden z' çözümünü oluşturmak için k adet bölge seçilir. Daha sonra, her bölgeden iki farklı eczane seçilir. Son olarak, eğer seçilen bir j eczanesi için $n_j > 1$ ise bu j eczanesinin nöbetçi olduğu günlerden biri seçilir. Sonra, z' çözümünü oluşturmak için her bölgede bir değiştirme (swap) olmak üzere toplam k adet değiştirme uygulanır.

Yukarıda bahsedilen bölgeleri, eczaneleri ve günleri seçme yöntemi için kontrollü rastgele yöntem kullanılır. Bu yöntemde (i) çok sayıda eczane içeren bölgelere, (ii) n_j değerleri yüksek olan eczanelere, (iii) amaç fonksiyon değerini daha çok arttıran günlere daha çok önem verilir. Dikkat edilirse, (i) ve (ii) kuralları daha büyük arama uzayı sağlar. Önceki kurallar mümkün olan

değiştirme sayısını arttırırken, son kural (iii). kural için aday günlerin sayısını arttırır. Her bölge, eczane ve günlere rastgele değerler atanır ve bu değerler aşağıdaki gibi hesaplanır:

- $rand\_region_k = U(0,1) * |J_k|$ bölgeler için, burada J_k, k bölgesindeki eczaneler kümesi ve $U(0,1)$, 0 ile 1 arasında düzgün rastgele değişkendir,
- $rand\_pharmacy_j = U(0,1) * n_j$ eczaneler için
- $rand\_day_t = U(0,1) * (f_t - f_{min} + 1)$ günler için, burada f_t, t gününün amaç fonksiyon değeridir öyle ki $F = \sum_t f_t$ ve $f_{min} = \min_t\{f_t\}$)

Her ihtiyaç duyulduğunda, en yüksek değere sahip olan bölge, eczane ve gün seçilir. Arama uzayının farklı bölgelerinde arama yapabilmek için, Sallama Algoritması ziyaret edilmemiş bölgelere öncelik verir. Bu amaçla, algoritma seçilen bölgeler için $rand\_region_k$ değerlerini $rand\_region_k = -1$ olarak günceller. Bölgeler için $rand\_region_k = -1$ olan bölgelerin sayısını k^- olarak tanımlayalım. Bir iterasyonda, algoritma k adet bölge seçer. Eğer $K_2 - k^- < k$ ise, algoritma yeteri kadar negatif olmayan $rand\_region_k$ değerine sahip bölge olmadığı sonucuna varır ve bütün bölgelere yeni $rand\_region_k$ değeri atar. Diğer yandan, Sallama Algoritması her iterasyonda yeni $rand\_pharmacy_j$ ve $rand\_day_t$ değerleri oluşturur.

Algoritma 22'nin 2(b) adımında rastgele oluşturulan z' çözümü etrafında yerel arama yapılır. TDKA algoritmasında uygulanan yerel arama metodu her zaman komşuluk yapısından, N_k, bağımsızdır ve $k = 1$ olarak alır. Yerel arama birinci bölgeden başlar ve son bölgeye kadar mümkün olan tüm değişim kombinasyonlarını uygular. Yerel aramada yerel en iyi çözüm bulununcaya kadar ilk iyileştirme tekniği uygulanır, yani uygulanan bir değişim iyileştirme sağladığında değiştirme çözüme uygulanır ve yerel minimum güncellenir. Arama güncellenmiş çözüm üzerinden tekrar başlatılır.

Algoritma 22, hareket adımında 2(c), yerel en iyi çözüme, z'', ulaşıldığında z'' ve z çözümlerinin amaç fonksiyon değerleri karşılaştırılır. Eğer $f(z") < f(z)$ ise, mevcut çözüm bulunan yerel en iyi çözüm ile değiştirilir, $(z = z")$, ve aramaya birinci komşuluktan devam edilir N_1, $(k = 1)$, aksi takdirde, komşuluk bir birim arttırılır, $(k = k + 1)$.

Algoritma 23. Sallama Algoritması

(1) **Sıralama:**

(a) Bölgeleri $rand\_region_k$ değerlerine göre azalan sırada sırala,

(b) Eczaneleri $rand\_pharmacy_j$ değerlerine göre azalan sırada sırala,

(c) Günleri $rand\_day_t$ değerlerine göre azalan sırada sırala,

(2) İlk k bölgeyi seç ve K^* seçilen bölgelerin kümesi olsun,

(3) Her $k \in K^*$ için, ilk iki eczaneyi, $j_{1k}^*,\ j_{2k}^*$, seç, öyle ki , $j_{1k}^*,\ j_{2k}^* \in J_k$,

(4) Her $k \in K^*$ ve $j_{hk}^* \in J_k,\ h = 1,2$ için, eğer $n_{j_{hk}^*} > 1$ ise, j_{hk}^* eczanesinin nöbetçi olduğu ilk günü seç,

(5) Her $k \in K^*$ için, j_{1k}^* ve j_{2k}^* eczanelerinin seçilen günlerdeki nöbetlerini değiştir,

z çözümünden z' çözümünü oluştur,

(6) **Güncelleme:**

(a) Her $k \in K^*$ için, $rand\_region_k$ değerlerini -1 olarak güncelle,

(b) k^- parametresi $rand\_region_k$ değerleri -1 olan bölgelerin sayısı olsun, eğer $K_2 - k^- < k$ ise $rand\_region_k$ değerlerini baştan oluştur,

(c) Rastgele $rand\_pharmacy_j$ ve $rand\_day_t$ değerlerini oluştur.

Üç farklı durdurma koşulu tanımlanmıştır. Bunlar a) En çok CPU zamanı, b) En çok genel tekrar sayısı, c) En çok dış tekrar sayısıdır (Algoritmanın bilinen en iyi çözümü başlangıç çözümü olarak alıp temel adıma dönmesini sayar). En çok CPU zamanı koşulu diğer durdurma koşullarıyla aynı anda kullanılır. TDKA, Değişken Komşuluk Ayrıştırma Arama (DKAA) ve Değişken Komşuluk Kısıtlama Arama (DKKA) sezgisellerini doğru şekilde karşılaştırabilmek için aynı en çok CPU zamanı koşulu kullanılmıştır. Her iki en çok genel tekrar sayısı ve en çok dış tekrar sayısı için ikişer parametre seviyeleri belirlenmiş ve bu seviyeler test edilmiştir.

3.8.2 Azaltılmış Değişken Komşuluk Arama

Hansen v.d. (2001) hızlı ve kaliteli bir başlangıç çözümü elde etmenin büyük boyutlu problemleri çözmek için önemli olduğunu vurgulamışlardır. Algoritmanın çözüm süresini ciddi derecede azaltmak için TDKA algoritmasının yerel arama adımını atarak Azaltılmış DKA (ADKA) sezgiselini geliştirmişlerdir. Grujicic ve Stanimiroviç (2012) özel polis kuvvetlerinin acil servis

ağlarını optimize etmek için TDKA algoritmasına başlamadan önce iyileştirilmiş bir başlangıç çözümü elde etmek için ADKA sezgiselini kullanmışlardır.

ADKA algoritması TDKA algoritmasının yerel arama 2(b) adımı dışında bütün adımlarını uygular. Her adımda, ADKA mevcut çözümün k. komşuluğundan rastgele bir çözüm oluşturur ve eğer bu çözüm mevcut çözümden daha iyiyse bu çözüme hareket eder. En çok iyileştirme olmayan tekrar sayısı durdurma koşulu olarak kullanılmış ve $100 * K$ olarak belirlenmiştir. Bu metotla çok iyi başlangıç çözümleri elde edilmiştir.

TDKA, DKAA ve DKKA algoritmaları için aynı başlangıç çözümü oluşturma metodu kullanılmıştır. Öncelikle, bir önceki bölümde tanımlandığı gibi bir başlangıç çözümü oluşturulur, daha sonra bu çözüm ADKA kullanılarak iyileştirilir.

3.8.3 Değişken Komşuluk Ayrıştırma Arama

Literatürde, Değişken Komşuluk Ayrıştırma Arama (DKAA) algoritması özellikle büyük boyutlu problemler için kısa hesaplama zamanında iyi kalitede çözümler elde etmek için önerilir. Bu amaca, çözüm uzayını ayrıştırarak ve yerel aramayı daha küçük bir uzayda uygulayarak ulaşılmıştır.

DKAA rastgele k adet bölge seçer ve seçilen bölgelerdeki nöbet değişiminden etkilenebilecek müşterileri dikkate alır. Dikkat edilirse bu müşteriler, müşteriler kümesinin bir alt kümesidir. DKAA ayrıştırılmış çözüm uzayında bir sezgisel arama uygular ve bir durdurma koşuluyla durur. Daha sonra, ayrıştırılmış uzay için elde edilen çözüm ile bütün problemin diğer kısmı birleştirilir.

DKAA algoritmasının adımları Algoritma 24'de verilmiştir. Başlangıç adımında bir mümkün çözüm oluşturulur, komşuluk yapıları ve durdurma koşulu tanımlanır. Aynı zamanda, bütün bölgeler için k bölgesi seçildiğinde dikkate alınması geren müşterilerin kümesi olan I_k tanımlanmıştır.

$I_k = \{i: \alpha_i^k = 1, i \in I\}$ olsun, burada $d_i^k = \min_{k_2 \in K \setminus \{k\}} \max_{j \in J_{k_2}} d_{ij}$ ve

$$\alpha_i^k = \begin{cases} 1, & if\ there\ exists\ a\ j \in J_k\ such\ that\ d_{ij} \leq d_i^k; \\ 0, & otherwise. \end{cases}$$

d_i^k parametresi, i müşterisinin k bölgesinde hiç eczane olmadığında gideceği en uzak yolun uzunluğudur.

Algoritmanın 1. adımında, DKAA $k = 2$ alarak başlar. Ayrıştırılmış problemde, tek bir bölge olduğunda her çizelge aynı amaç fonksiyon değerini verdiği için $k = 1$ durumu atlanmıştır.

Algoritmanın 2. adımında, DKAA komşuluk yapısını N_k olarak alır ve sırasıyla sallama, yerel arama ve hareket adımlarını takip eder. 2(a) adımında, DKAA k adet bölge seçmek zorundadır. İlk olarak, en yüksek $rand\_region_k$ değerine sahip bölgeyi seçer. Daha sonra, her iterasyonda daha önceden seçilen bölgelere en yakın olan eczaneyi, eczaneler arasındaki ortalama uzaklığı hesaplayarak, k adet bölge seçilinceye kadar ekler. Seçilen bölgelerin $rand\_region_k$ değerleri -1 olarak güncellenir. Bütün $rand\_region_k$ değerleri -1 olduğunda, DKAA yeni $rand\_region_k$ değerlerini oluşturur. Dikkat edilirse, seçilen bir bölge sonraki iterasyonlarda birinci bölge olamaz, ama hala seçilebilir. Bir iterasyonda seçilen bölgelerin kümesi K^* ve seçilen bölgelerdeki değişimlerden (swaps) etkilenebilecek müşterilerin kümesi de $I^* = \bigcup_{k \in K^*} I_k$ olsun.

Seçilen bölgelerin birbirinden uzakta olduğu bir durumu düşünelim. Bu durumda, her müşteri sadece bir bölgedeki değişimlerden etkilenecek ve amaç fonksiyon değeri değişmeyecektir. Bu durum yukarıda tanımlanan $k = 1$ durumuna benzer bir durumdur. Bundan dolayı, birbirine yakın k adet bölgenin seçilmesi amaçlanmıştır ve böylece bazı müşteriler için farklı bölgelerden aday eczaneler olacak ve yapılacak değişimler amaç fonksiyon değerini değiştirebilecektir.

Verilen bir problem örneği için eczaneler ve müşteriler arasındaki uzaklıklar bilinmektedir. k_1 ve k_2 bölgeleri arasındaki uzaklık $\overline{d_{k_1 k_2}} = \frac{\sum_{j_1 \in k_1} \sum_{j_2 \in k_2} d'_{j_1 j_2}}{|J_{k_1}| \times |J_{k_2}|}$ olarak tanımlanmıştır, burada $d'_{j_1 j_2} = \min_i \{d_{i j_1} + d_{i j_2}\}$ 'dir.

TDKA algoritmasına benzer şekilde, DKAA seçilen her bölgede bir değiştirme yaparak z çözümünden z' çözümünü elde eder. DKAA, K^* içindeki bölgeleri, I^* içindeki müşterileri ve bütün T günlük planlama ufkunu dikkate alarak ayrıştırılmış çözümü, $w \in \mathbb{N}^{k \times T}$, oluşturur.

2(b) adımında, DKAA w çözümü üzerinde sezgisel arama uygulayarak w' çözümünü Hansen ve Mladenovic (2001) ve Hansen v.d. (2001) 'de tanımlandığı gibi elde eder. Dikkat edilirse, bu

amaç için herhangi bir sezgisel metot uygulanabilir. TDKA algoritmasına benzer şekilde, DKAA algoritması bu adımda değiştirme algoritmasını kullanır. Temel fark şudur, TDKA değiştirme algoritmasını bütün probleme uygularken, DKAA bunu daha az bölgeye, eczaneye ve müşteriye sahip olan ayrıştırılmış probleme uygular.

Algoritma 24. DKAA Algoritması

Başlangıç Adımı: Başlangıç çözümü (çizelgesi) z 'yi oluştur, aramada kullanılacak komşuluk yapılarını, $N_k,\ k = \{2, \dots, k_{max}\}$, oluştur, müşteri alt kümelerini $I_k,\ k \in K$, oluştur ve durdurma koşulunu belirle.

Temel Adım: Aşağıdaki adımları durdurma koşulu sağlanıncaya kadar tekrarla,

(1) $k = 2$ olsun,

(2) $k = k_{max}$ oluncaya kadar aşağıdaki adımları tekrarla,

(a) *Sallama:* z çözümünün k. komşuluğundan rastgele z' çözümünü oluştur $(z' \in N_k(z))$, ve w çözümü, $w \in \mathbb{N}^{k \times T}$, z' çözümü için rastgele seçilen bölgeler $k \in K^*$ ve müşterilerden $I^* = \cup_{k \in K^*} I_k$ oluşsun,

(b) *Yerel Arama:* w çözüm uzayında yerel en iyi çözümü, w', değiştirme algoritması kullanarak bul, ve bütün uzay için, Z, yerel en iyi çözümü z" ile göster, $(z" = (z' \backslash w) \cup w')$,

(c) *Hareket:* Eğer $f(z") < f(z)$ ise, mevcut çözümü değiştir $(z = z")$, ve aramaya ikinci komşuluktan devam et $N_2,\ (k = 2)$, aksi takdirde, komşuluğu bir birim arttır $(k = k + 1)$.

DKAA ayrıştırılmış problem için w' çözümünü elde eder. Daha sonra, bu çözümü z' çözümünün kalan kısmıyla birleştirir, $(z' \backslash w)$, ve z" çözümünü elde eder $(z" = (z' \backslash w) \cup w')$.

2(c) adımında, z" ve mevcut çözüm z 'nin amaç fonksiyon değerleri karşılaştırılır. Eğer $f(z") < f(z)$ ise, mevcut çözüm bulunan yerel en iyi çözüm ile değiştirilir, $(z = z")$, ve aramaya ikinci komşuluktan devam edilir $N_2,\ (k = 2)$. Aksi takdirde, komşuluk bir birim arttırılır, $(k = k + 1)$.

3.8.4 Değişken Komşuluk Kısıtlama Arama

Değişken Komşuluk Kısıtlama Arama (DKKA) algoritması özellikle büyük boyutlu problemler için kısa hesaplama zamanında iyi kalitede çözümler elde etmek için önerilmiştir. Problemi ayrıştırmadan yerel aramayı daha küçük uzayda uygulayarak bunu başarmak amaçlanmıştır.

DKAA ve DKKA algoritmaları arasında üç temel farklılık vardır: (i) bölge seçme yöntemi, (ii) yerel arama, ve (iii) en az k değeri. DKAA birbirine yakın bölgeleri seçer ve sadece seçilecek olan birinci bölge negatif olmayan $rand\_region_k$ değerine sahip olmalıdır. DKKA ise bölgeleri rastgele birbirlerine olan yakınlıklarını dikkate almadan seçer ve seçilen her bölge negatif olmayan $rand\_region_k$ değerine sahip olmalıdır. DKKA ve TDKA algoritmaları bölgelerin, eczanelerin ve günlerin rastgele değerlerini güncellemek için aynı yöntemi takip ederler.

Yerel arama adımında, DKAA ve DKKA algoritmaları değiştirme algoritmasını sadece seçilen bölgelerde uygularlar. DKAA bir değiştirmenin kazancını, I^* içindeki müşterileri ve K^* içindeki bölgeleri dikkate alarak hesaplarken, DKKA bütün müşterileri ve bölgeleri dikkate alarak hesaplar.

DKAA algoritmasında, $k = 1$ olma durumu bir anlam ifade etmediği için, k değeri 2 ile k_{max} arasında değişir. Buna rağmen, DKKA algoritmasında bütün müşteriler ve bölgeler dikkate alındığı için bir bölgede yapılan bir değişim amaç fonksiyon değerini değiştirebilir. Bundan dolayı, DKKA, $k = 1$ durumuna da izin verir.

TDKA, DKAA ve DKKA algoritmalarının yerel arama yöntemleri üzerine detaylı bir analiz yapılmıştır. Komşuluk büyüklüğü (cardinality) ve yeni bir çözüme hareket (move) ettikten sonra yeni amaç fonksiyonu hesaplama karmaşıklığı (complexity) analiz edilmiştir. Sonuçlar Tablo 1'de verilmiştir. TDKA için, bir bölge için $T!$ mümkün değiştirme vardır. Bundan dolayı, yerel aramanın komşuluk büyüklüğü $O((T!)^K)$ 'dır. DKAA ve DKKA yerel aramayı k bölgede uygular. k algoritmalar için bir parametre olduğu için, komşuluk büyüklüğü $O(T!)$ 'dir.

Bir çözümden diğerine geçtikten sonra, çizelgenin sadece iki günü değişir ve sadece bu günlerin yeni amaç fonksiyon değerlerini hesaplamak yeterlidir. TDKA ve DKKA bütün müşteri kümesini dikkate alır. Her iki algoritma, her müşteri için K adet bölgedeki en yakın nöbetçi eczaneyi seçer ve yeni amaç fonksiyonu değeri hesaplamanın hesaplama karmaşıklığı $O(IK)$ 'dır. Diğer

taraftan, DKAA sadece bütün müşterilerin bir alt kümesini dikkate alır $(i \in I^*)$. K^*, burada $|K^*| = k$, içindeki bölgelerden herhangi birine uygulanan bir değiştirme sadece bu müşterilerin eczane atamalarını değiştirebilir. DKAA için, yeni amaç fonksiyonu değeri hesaplamanın hesaplama karmaşıklığı $O(I)$ 'dır.

Yerel arama, indeksi en küçük olan bölgeden başlar ve bölge indeksine göre artan sırada bölgeleri tarar. Gelecek araştırmalar için, bölgeleri farklı sırada inceleyen çeşitli yöntemler geliştirilebilir.

Tablo 3. Yerel arama karmaşıklık analizi

	TDKA	**DKAA**	**DKKA**
Komşuluk Büyüklüğü	$O((T!)^K)$	$O(T!)$	$O(T!)$
Amaç Fonk. Hesaplama Karmaşıklığı	$O(IK)$	$O(I)$	$O(IK)$

3.9 İki Amaçlı Eczane Nöbet Çizelgeleme Problemi

ENÇ probleminde amaç toplam kat edilen mesafeyi talep ağırlıklı olarak en azlamaktır. Bu amaç fonksiyonu ile hizmet alanlara planlama ufku boyunca en iyi hizmetin verilmesi hedeflenir. Ancak önerilen çözümde sadece hizmet alanlar dikkate alınır ve hizmet verenlerin bu çözümden nasıl etkilendiği incelenmez.

Hizmet verenlerin de dikkate alınması yaklaşımı ile ENÇ problemi için ikinci bir amaç tanımladık. Bu amaç planlama ufku boyunca her bir eczanenin iş yükünü belirlemeyi ve en az iş yüküne sahip olan eczaneye mümkün olduğunca fazla müşteri atanmasını hedefler. ENÇ tarafından *j* eczanesine atanan toplam müşteri talebi, w_j, (İA-1) eşitliği ile bulunabilir.

$$w_j = \sum_{t=1}^{T} \sum_{i=1}^{I} h_i x_{ijt} \qquad \text{(İA-1)}$$

Bu bölümde İki Amaçlı ENÇ (İAENÇ) probleminin matematiksel modelini sunduk. Ayrıca her eczaneye atanabilecek toplam talep miktarının alt ve üst sınırlarını bulabilmek için kesin ve sezgisel yöntemler önerdik. Tüm etkin sonuçları bulmak üzere bir algoritma geliştirdik.

3.9.1 Matematiksel Programlama Modeli

İAENÇ için ikinci amaç fonksiyonunu (İA-2)'de tanımladık.

$$Max\ w_0 = \min_{j \in J} w_j \qquad \text{(İA-2)}$$

Bu fonksiyon, en az talep ataması yapılan eczanenin toplam talebini, w_0, en büyüklemeyi amaçlar.

Ancak bir eczaneye atanan müşteri miktarı, eczanenin etrafındaki müşterilere ve diğer eczanelerin yerlerine bağımlıdır. Örneğin nüfus yoğunluğunun yüksek olduğu bölgelerdeki bir eczaneyi ele alalım. Birçok çözümde bu eczane en az müşteri atanan eczane olmayacaktır. Ancak bazı çözümlerde bu eczaneye atanan müşteri sayısı, olabilecek en alt seviyede olmasına rağmen matematiksel model tarafından en kötü durumdaki eczane olarak dikkate alınmayacaktır. Bu sebeple bir eczanenin iş yüküne ek olarak, bu eczaneye atanabilecek en az ve en fazla müşteri sayısını da dikkate alan bir yaklaşım geliştirmeyi düşündük. Bir eczaneye planlama ufku boyunca atanabilecek en az müşteri sayısı L_j ve en fazla müşteri sayısı U_j parametreleri ile tanımlanmış olsun (Bu iki parametrenin değerinin hesaplanması konusuna ayrıca değineceğiz). Bu durumda j eczanesinin normalize edilmiş iş yükü p_j, (İA-3) eşitliği ile hesaplanabilir. Güncellenmiş ikinci amaç fonksiyonu (İA-4) ise en düşük normalize edilmiş iş yükü olan p_0 değerini en çoklamayı amaçlar.

$$p_j = \frac{w_j - L_j}{U_j - L_j} \qquad \text{(İA-3)}$$

$$Max\ p_0 = \min_{j \in J} p_j \qquad \text{(İA-4)}$$

İAENÇ probleminin matematiksel modeli şu şekilde tanımladık:

İAENÇ Modeli

Amaç fonksiyonu:

$$Min\ F_1 = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{t=1}^{T} h_i d_{ij} x_{ijt} \qquad \text{(GM1-1)}$$

$$Max\ F_2 = p_0 \qquad \text{(İA-5)}$$

Kısıtlar:

GM1-2,..., GM1-7)

ENÇ problemi için gerekli temel kısıtlar kümesi.

$$p_j = \frac{w_j - L_j}{U_j - L_j} \ \forall j \qquad \text{(İA-6)}$$

Her eczanenin normalize edilmiş iş yükü hesaplanmalıdır.

$$p_0 \leq p_j \ \forall j \qquad \text{(İA-7)}$$

En düşük normalize edilmiş iş yükü tüm eczanelerin normalize edilmiş iş yüklerinden küçük olmalıdır.

$$p_j, w_j \geq 0 \ \forall j \qquad \text{(İA-8)}$$

$$p_0, w_0 \geq 0 \qquad \text{(İA-9)}$$

Negatif olmayan sürekli karar değişken tanımları.

Bu problemin etkin sonuçlarını bulmak için ε-kısıt yaklaşımı ile aşağıdaki tek amaçlı matematiksel modeli tanımladık.

İAENÇ(ε) Modeli

Amaç fonksiyonu:

$$Min\ F = \left[\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{t=1}^{T} h_i d_{ij} x_{ijt}\right] - \rho p_0 \qquad \text{(İA-10)}$$

Kısıtlar:

$$p_0 \geq \varepsilon \qquad \text{(İA-11)}$$

En düşük normalize edilmiş iş yükü ε kadar olmalıdır.

GM1-1,...,GM1-7 ve İA2-6,...,İA2-9.

İAENÇ problemi için gerekli temel kısıtlar kümesi.

İAENÇ(ε) modelinin amaç fonksiyonu İAENÇ modelinin iki amaç fonksiyonunun ağırlıklı toplamı şeklinde tanımlanmıştır. Bu sayede İAENÇ(ε) modeli zayıf etkin olan ama etkin olmayan sonuçları bulmayacaktır. İlk amaç fonksiyonun ağırlığı bir, ikinci amaç fonksiyonun ağırlığı ise $-\rho$ olarak kullanılmıştır. İlk amaç fonksiyonun değeri tam sayı ve ikinci amaç fonksiyonu bire normalize edilmiş bir değerdir. Herhangi bir $\rho < 1$ değeri belirlemek önceliğin ilk amaca verilmesine ve aynı ilk amaç değerine sahip sonuçlar arasından en yüksek ikinci amaç değerini veren çözümün seçilmesi için yeterli olacaktır.

İAENÇ(ε) modeli ile ilgili cevaplanması gereken iki önemli konu vardır. Bunlar (i) rasyonel olmayan müşteri eczane atamaları, (ii) L_j ve U_j parametre değerlerinin belirlenmesidir.

Rasyonel olmayan müşteri eczane atamaları

ENÇ modelinin amaç fonksiyonu sebebi ile her müşteri her gün kendisine en yakın nöbetçi eczaneye atanmaktadır. Ancak İAENÇ(ε) modeli bir müşteriyi kendisine en yakın nöbetçi eczane yerine daha uzaktaki bir nöbetçi eczaneye atayabilir. Bu sayede uzaktaki eczaneye daha fazla müşteri atanmış ve verilen ε değeri için (İA-11) kısıtı bu eczane açısından sağlanmış olur. Rasyonel olmayan bu atamaların önlenmesi için aşağıda sunulan (İA-12) kısıt kümesinin İAENÇ(ε) modeline eklenmesi gerekmektedir.

$$x_{iht} \leq 1 - y_{jt} \; \forall i, j, h, t \mid d_{ij} < d_{ih}, r_j \neq r_h \qquad \text{(İA-12)}$$

Bu kısıt, *i* müşterisinin kendisine daha yakın olan *j* eczanesi nöbetçi olduğunda, müşterinin kendisine daha uzak olan *h* eczanesine atanmasını engellemektedir. Aynı günde bir bölgeden tek eczane nöbetçi olacağı için *j* ve *h* eczanelerinin farklı bölgelerden olması koşulunu kısıta ekledik ve toplam kısıt adedini azaltmayı amaçladık.

Müşteri taleplerinin alt ve üst sınırlarının belirlenmesini bir sonraki bölümde tartıştık.

3.9.2 Talep Alt ve Üst Sınırlarının Belirlenmesi

İAENÇ modeli bir eczanenin normalize edilmiş iş yükünü hesaplayabilmek için L_j ve U_j parametrelerine ihtiyaç duyar. Bu iki parametre sırasıyla, bir eczaneye olurlu (ve rasyonel) bir çözümde atanabilecek en az ve en fazla müşteri sayısına karşılık gelmektedir.

Verilen bir *j* eczanesi için bu parametrelerin kesin değerlerinin hesaplanması amacıyla İAENÇ matematiksel modeline benzer LU(*j*) matematiksel modelini önerdik.

LU(*j*) Modeli

Amaç fonksiyon

$$Min\ veya\ Max\ w_j^* = \sum_{i=1}^{I}\sum_{t=1}^{T} h_i x_{ijt} \qquad \text{(İA-13)}$$

Kısıtlar

GM1-2,...,GM1-7 ve İA-2,...,İA-12.

Olurlu bir çözüm elde etmek ve rasyonel atamalar yapılması için bu kısıt kümelerine ihtiyaç duyulur.

LU(*j*) matematiksel modelini enazlamak amacı ile çözerek L_j parametresinin optimal değeri olan L_j^* değeri, ençoklamak amacı ile U_j parametresinin optimal değeri olan U_j^* değeri elde edilir.

Tüm eczanelerin alt ve üst sınır değerlerin hesaplanması için toplamda 2*J* adet matematiksel model çözülmesi gerekir. Bunun yerine L_j ve U_j parametrelerinin değerlerini güvenli şekilde tahmin eden sezgisel yöntemler önerdik. Parametreler $L_j^m \leq L_j^*$ ve $U_j^m \geq U_j^*$ *j* eczanesinin iş yükü alt ve üst sınırını tahmin etmek için kullanılan *m*. sezgisel yöntem ile elde edilen değerler olsun.

Proje kapsamında bir alt bir de üst sınır sezgiseli geliştirdik. Bu sezgisel yöntemler *i* müşterisi ile nöbetçi *j* eczanesi arasında tanımlı iki uzaklık bilgisine ihtiyaç duyar, $\underline{d_{ij}}$ ve $\overline{d_{ij}}$. Bu uzaklıklar *j* eczanesinin bulunduğu bölgedeki (r_j bölgesi) eczaneler yok sayılarak hesaplanır ve *i* müşterisinin atanabileceği en yakın ve en uzak eczanelerin mesafeleri gösterir. Bu bölgedeki eczanelerin yok sayılmasının sebebi *j* eczanesi nöbetçi iken diğerlerinin nöbetçi olamayacağı ve *i* müşterisinin *j* eczanesi dışındaki en iyi ve en kötü alternatifinin ne olacağı bilgisine ihtiyaç duyulmasıdır.

$$\underline{d_{ij}} = \min_{h|r_h \neq r_j} \{d_{ih}\} \qquad \overline{d_{ij}} = \min_{k|k \neq r_j} \left\{ \max_{h \in J_k} \{d_{ih}\} \right\}$$

Mesafe alt sınırı için r_j bölgesi dışındaki eczanelerin en yakınının uzaklığı bulunur. Üst sınır için ise r_j bölgesi dışındaki her *k* bölgesi için *i* müşterisine en uzak eczanenin mesafesi bulunur, daha

sonra ise bu mesafelerin en yakını seçilir. Çünkü her gün her bölgeden bir eczane nöbetçi olacak ve *i* müşterisi hiçbir durumda bu mesafeden daha uzak bir eczaneye atanmayacaktır.

Bu uzaklıklar kullanılarak L_j^1 ve U_j^1 değerleri İA-14 ve İA-15 eşitliklerinde hesaplanır.

$$L_j^1 = n_j \sum_{i|d_{ij}<\underline{d_{ij}}} h_i \quad \forall j \qquad \text{(İA-14)}$$

Alt sınır uzaklığından daha yakın tüm müşteriler *j* eczanesi nöbetçi olduğu günler (n_j defa) *j* eczanesine atanacaklardır. Toplam ifadesi altında küçük eşit yerine küçüktür kullanılmasının sebebi, *j* eczanesi ve diğer bir eczane arasında mesafede eşitlik olması durumunda müşterinin diğer eczaneye atanacağının varsaymamızdır. Bu sayede $L_j^1 \leq L_j^*$ eşitsizliği korunmuş olur.

$$U_j^1 = n_j \sum_{i|d_{ij}\leq\underline{d_{ij}}} h_i \quad \forall j \qquad \text{(İA-14)}$$

Üst sınır uzaklığından daha yakın tüm müşteriler *j* eczanesi nöbetçi olduğu günler (n_j defa) *j* eczanesine atanacaklardır. Toplam ifadesi altında küçük eşit kullanılmasının sebebi, *j* eczanesi ve diğer bir eczane arasında mesafede eşitlik olması durumunda müşterinin *j* eczanesine atanacağının varsaymamızdır. Bu sayede $U_j^1 \geq U_j^*$ eşitsizliği korunmuş olur.

Optimal ve sezgisel yöntemlerin ihtiyaç duydukları süreleri 100 müşteri, 10 eczane, 7 gün ve 4 bölgenin olduğu bir örnek üzerinde karşılaştırdık. Optimal değerleri hesaplamak yaklaşık 6 saat, sezgisel değerleri hesaplamak ise bir saniyeden kısa sürdü.

3.9.3 Tüm Etkin Sonuçları Bulma Algoritması

İAENÇ(ε) modelini farklı ε değerleri ile çözerek tüm etkin sonuçların bulunması mümkündür. Bunun için Algoritma 25'i önerdik. Bu algoritma $\varepsilon = 0$ değeri ile başlar. Olurlu bir çözüm bulduğu sürece en düşük normalize iş yükü değerine sahip eczaneyi belirler ve bir sonraki ε değeri için bu eczaneye bir fazla müşteri atanması durumunda elde edilecek oranı hesaplar. Daha sonra ilk adıma döner ve güncellenmiş ε değeri ile yeni bir problem çözer. Olurlu bir çözüm olmaması durumunda daha önce bulunan çözümleri raporlar ve sona erer.

Algoritma 25 her olurlu çözüm sonrasında bir eczanenin iş yükünü bir birim artırır. Toplam talep miktarı $H = T\sum_{i=1}^{I} h_i$ birimdir. Algoritma 25 sonlu bir algoritmadır ve *O* (*H*) iterasyonda sona erer.

Algoritma 25. İAENÇ(ε) ile Tüm Etkin Sonuçların Bulunması

Adım 0: ε=0 ve c=1 olarak belirle

Adım 1: İAENÇ(ε) modelini çöz.

Adım 2: Eğer olurlu bir çözüm var ise;

2.1 Çözümü kaydet, (F_1^*, p_0^*).

2.2 En düşük normalize iş yükü değerine sahip eczane *j* olsun (öyle ki $p_0^* = p_j$)

Eğer aynı değere sahip birden fazla eczane varsa en büyük $U_j - L_j$ değeri olanı seç.

2.3 Seçilen eczanenin iş yükü w_j olmak üzere ε değerini güncelle

$$\varepsilon = \frac{1 + w_j - L_j}{U_j - L_j}$$

2.4 c=c+1 olarak güncelle ve Adım 1'e git.

Adım 3: Kaydedilen sonuçları raporla

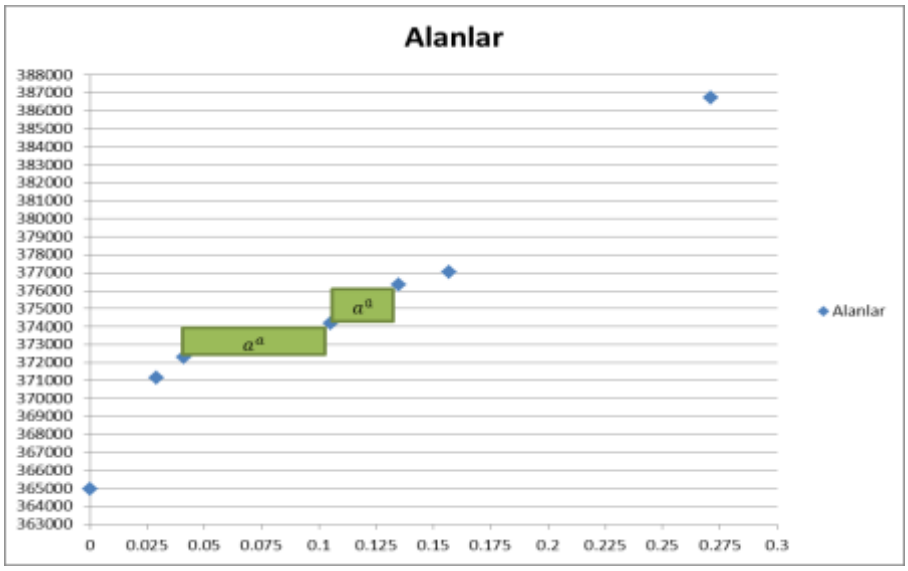
3.10 İki Amaçlı Değişken Komşuluk Arama Algoritması

Bu bölümde yukarıda tanımlanan ÇENP için kısa hesaplama zamanında en iyi etkin çözümleri yada etkin çözüme yakın çözümleri bulabilmek amacıyla İki-Amaçlı Değişken Komşuluk Arama (İDKA) algoritması geliştirilmiştir. Geliştirilen algoritma birinci amaç fonksiyonu olarak İAENÇ probleminde tanımlanan talep ağırlıklı toplam mesafeyi, F_1, ikinci amaça fonksiyonu olarak İAENÇ probleminde tanımlanan en düşük normalize edilmiş iş yükünü, F_2, almaktadır.

Liang ve Lo (2010), ÇDKA algoritmasını çok amaçlı optimizasyon problemlerini çözmek için Tek-Amaçlı DKA (TDKA) algoritmasından yola çıkarak geliştirilmiştir. İDKA algoritmasının temel fikri rastgele komşuluk seçme yöntemi, temel çözümü seçmek için geliştirilen yeni bir yöntem ve komşu çözümün kullanımı bakımından TDKA algoritmasından farklıdır. İDKA algoritması Algoritma **26.** İDKA AlgoritmasıAlgoritma 26'da verilmiştir. Bu çalışmada Liang ve Lo (2010) tarafından tanımlanan İDKA sezgiseli İAENÇ problemine uyarlanmış ve tek amaçlı ENÇ probleminde iyi sonuçlar veren Değişken Komşuluk Kısıtlama Arama (DKKA) algoritması İDKA algoritmasının komşuluk arama adımında kullanılmıştır.

İDKA algoritması rastgele mümkün bir başlangıç çözümünü Temel Değişken Komşuluk Arama (TDKA) algoritmasında kullanılan yöntemle oluşturarak başlar. Bu başlangıç çözümü yaklaşılmaya çalışılan Pareto eğrisinin (Pareto front) ilk ve tek elemanıdır. Daha sonra algoritma TDKA algoritmasında kullanılan komşuluk yapılarını oluşturur. Arama döngüsü mevcut etkin çözümler kümesinden bir mevcut çözümün seçilmesiyle başlar. Mevcut çözüm seçilirken etkin sonuçla kümesindeki her çözümün koordinatıyla bitişik etkin çözümlerinin koordinatlarının kesişimiyle oluşan alt, a^{a}, ve üst, $a^{ü}$, dikdörtgen alanlar toplamı en yüksek olan etkin çözüm daha önce temel çözüm olarak seçilmemişse mevcut çözüm olarak seçilir. Daha sonra seçilen çözüm ziyaret edildi olarak işaretlenir. Şekil 5'de etkin çözümler arasındaki uzaklıkların alanlar cinsinden nasıl hesaplandığı gösterilmiştir. Her etkin çözüm için bu alt ve üst alanlar toplamı hesaplanmıştır ve toplam alanı en yüksek olan ve daha önce temel çözüm olarak seçilmemiş olan etkin çözüm temel çözüm olarak seçilmektedir. Burada en yüksek alanı seçerek etkin çözümler arasındaki uzaklıkları dengelemek yani yaklaşılmaya çalışılan Pareto eğrisinin bütün yönlerinde etkin çözümler elde etmek amaçlanmıştır. Ayrıca, kümenin iki ucuna yani iki amaca ayrı ayrı daha fazla öncelik vererek etkin sonuçlar kümesinin genişliğini yani uç noktaları arasındaki uzaklığı arttırmak amacıyla birinci amaç fonksiyon değerinin en iyi olduğu çözümün alt alanı $a^{a} = \frac{Ü S - AS}{10 * J}$ olarak ve ikinci amaç fonksiyon değerinin en iyi olduğu çözümün üst alanı $a^{ü} = \frac{Ü S - AS}{10 * J}$ olarak alınmıştır. Burada, $Ü S$ ilk oluşturulan başlangıç çözümünün birinci amaç fonksiyon değeridir ve AS ise birinci amaç fonksiyonunun alt sınır algoritmasıyla (Ağlamaz ve Özpeynirci, 2011) hesaplanan alt sınırıdır. Mevcut bütün etkin çözümler ziyaret edilmişse bütün tümünün durumu ziyaret edilmemiş olarak güncellenir ve en yüksek alana sahip olan etkin çözüm temel çözüm olarak seçilir.

Komşuluk, oluşturulan komşuluk yapılarından rastgele olarak seçilir. Burada, TDKA sezgiselinin sallama algoritmasının aynısı kullanılır, ancak kaçıncı komşuluğa geçileceği rastgele belirlenir. Yani bölgelere, eczanelere ve günlere rastgele değerler atama yöntemi ve seçilen bölgelerin, eczanelerin ve günlerin rastgele değerlerini güncelleme yöntemi TDKA algoritmasında kullanılan yöntemin aynısıdır. Ancak, TDKA komşuluk sayını bir arttırarak maksimum komşuluk sayısına, k_{max}, kadar iç iterasyonu devam ettirirken İDKA komşuluk sayısını 1 ile maksimum komşuluk sayısı arasında rastgele olarak belirler ve belirlenen komşulukta seçilen temel çözümünün etrafında rastgele z' çözümü oluşturulur. Maksimum komşuluk sayısı TDKA algoritmasında belirlenen, $\lceil K/2 \rceil$, olarak alınmıştır.



Şekil 5. Etkin çözümler arasındaki uzaklık

Algoritma 26. İDKA Algoritması

Başlangıç Adımı: Başlangıç çözümü (çizelgesi) z 'yi oluştur, aramada kullanılacak komşuluk yapılarını, N_k, $k = \{1, \dots, k_{max}\}$, oluştur, başlangıç çözümünü yaklaşılan Pareto eğrisinin ilk elemanı olarak al,

Arama Yöntemi: Aşağıdaki adımları durdurma koşulu sağlanıncaya kadar tekrarla,

(1) Bir seçme yöntemine dayanarak yaklaşılan Pareto eğrisi üzerinden temel çözüm z 'yi seç,
(2) Komşuluk yapılarından rastgele bir komşuluk seç,
(3) *Sallama:* z çözümünün k. komşuluğundan rastgele z' çözümünü TDKA sallama algoritmasını kullanarak oluştur $(z' \in N_k(z))$,
(3) *Yerel Arama:* z' çözümünün seçilen komşuluktaki bütün komşu çözümlerini bul,
(4) *Güncelleme:* Bulunan bütün komşu çözümleri kullanarak yaklaşılan Pareto eğrisi güncelle,
(5) Temel çözüm halen yaklaşılan Pareto eğrisinde korunuyorsa ziyaret edildi olarak işaretle.

Algoritmanın yerel arama adımında, z' çözümünün belirlenen komşuluktaki bütün komşu çözümleri DKKA algoritması kullanılarak bulunur. Daha sonra, bulunan bütün komşu çözümler kullanılarak baskılanan (dominated) ve az baskın (weakly non-dominated) olan çözümler elenerek etkin çözümler kümesi (yaklaşılan Pareto eğrisi) güncellenir ve sadece etkin (non-dominated) çözümler kalır. Son olarak, seçilmiş olan temel çözüm halen etkin sonuçlar kümesinde ise ziyaret edildi olarak işaretlenir.

Alternatif çözüm sayısını arttırmak amacıyla, her 50 iterasyon sonunda $(50., 100., 150., \dots, 100 * T.)$ eğer seçilen temel çözümde en az iş yükü sıfıra eşit olan bir yada birden fazla eczane bulunuyorsa, algoritma en az iş yükü sıfır olan eczanelerden birini, j, rastgele seçer. Daha sonra, bu eczaneye kendi bölgesinden olmayan en yakın eczaneyi, $(j' : j' \in J \backslash \{J_{r_j}\})$, bulur. j' eczanesine ise kendi bölgesinden olan en uzak eczaneyi, j", bulur. Son olarak, j' ve j" eczanelerinin nöbetlerini değiştirir. Eğer $n_{j'} > 1$ ya da $n_{j"} > 1$ ise, bu eczanelerin nöbetçi oldukları ilk günleri alır ve o günlerdeki nöbetlerini değiştirir. Böylece, kritik eczaneye (en az iş yükü sıfır olan eczane) en yakın olan eczane kapatılarak daha uzakta bir eczane açılmış ve bazı müşterilerin kritik eczaneye atanması sağlanarak kritik eczanenin iş yükünün arttırılması amaçlanmıştır.

İDKA algoritması durdurma koşulu sağlanıncaya kadar devam eder. Burada, durdurma koşulu en çok tekrar sayısı olarak seçilmiş ve $100 * T$ olarak belirlenmiştir. Ayrıca, büyük boyutlu problemler için $1{,}5$ saat CPU zamanı sınırıda en çok tekrar sayısı koşuluyla aynı anda kullanılmıştır.

4. BULGULAR

Bu bölümde geliştirdiğimiz yöntemleri test etmek amacıyla çeşitli büyüklüklerde örnek problemler ve İzmir için gerçek hayat problemleri oluşturduk. Öncelikle bu örnek problemleri nasıl geliştirdiğimizi aktardık. Daha sonra önerdiğimiz yöntemleri bu problemler üzerine test ettik ve sonuçlarını raporladık.

4.1 Örnek Oluşturma

Bu bölümde rassal örneklerin oluşturulma yönteminden bahsettik. Öncelikle BM1 ve GM1 problemleri için ortak olan özellikleri aktardık. Rassal örnek oluşturmak için Özpeynirci ve Köksalan (2010) tarafından kullanılan yönteme benzer bir yöntemi kullandık. Müşteri ve eczane düğümlerinin yerlerini 1000X1000'lik bir kare üzerinde birbiçimli (uniform) dağılım kullanılarak belirledik. Düğümler arası uzaklıkları öklid uzaklık yöntemi ile belirleyerek en yakın tamsayıya yuvarladık. Müşteri talepleri 1 ile 15 birim arasında birbiçimli dağılım ile belirlenerek en yakın tamsayıya yuvarladık.

Üç farklı örnek boyutu üzerinde çalıştık. İlki küçük boyutlu problemler olarak adlandırdığımız en fazla 60 müşteri, 90 eczane ve 15 gün olan örneklerdir. İkincisi ise büyük boyutlu problemler olarak adlandırdığımız 350 müşteri, 250 ile 1000 arasında eczanenin olduğu 25 ile 100 gün arasında planlama ufkuna sahip örneklerdir. Sonuncusu ise bir sonraki bölümde detaylı olarak anlatacağımız, İzmir'e ait gerçek örneklerdir.

Küçük boyutlu problemler: Müşteri ve eczane sayıları ile planlama periyodu uzunluğu için 9 farklı örnek büyüklüğü belirledik. Bu örnek büyüklükleri belirlerken gerçek hayat problemine oransal anlamda benzer olmalarına dikkat ettik. Gerçek hayat probleminde I=350 müşteri, J=1000 eczane ve T=90 gün vardır. Örnek büyüklüklerini (I, J, T) şablonunda, (20,10,5), (20,20,10), (20,30,15), (40,20,5), (40,40,10), (40,60,15), (60,30,5), (60,60,10) ve (60,90,15) şeklinde belirledik. Bu özellikleri hem BM1 hem de GM1 problemleri için kullandık. GM1 probleminde bu özelliklere ek olarak eczanelerin nöbet bölgelerine göre bölünmesi gerekmektedir. Bu amaçla 1000X1000'lik karenin satır ve sütunları $\sqrt{K}$ parçaya ayrılarak K eşit bölüm elde ettik. Aynı bölümde yer alan eczanelerin aynı nöbet bölgesinde olduğu varsaydık. Her bölgede en az bir eczane olmasını garantilemek için ilk K eczaneyi her bölgeye birer tane olacak şekilde (ancak halen bölge içindeki yerleri rassal olacak şekilde) atadık. Bölge sayısı, 20 veya daha az eczane

olan örnekler için 4 diğerleri için 9 olarak belirledik. Küçük boyutlu problem toplamda 9 farklı problem boyutu tanımladık ve her problem boyutunda 10 rassal örnek oluşturduk.

Büyük boyutlu problemler: Büyük boyutlu örnekleri sadece çok nöbetli problemler için oluşturduk. Problemler (*I*,*J*,*T*,*K*) formatında (350,250,25,49), (350,500,50,49) ve (350,1000,100,49) boyutlarındadır. Bu 3 farklı problem boyutunun her birisi için 5 rassal örnek oluşturduk.

Projenin ilk yılı içerisinde gerçekleştirdiğimiz deneylerde kullandığımız örnekler ile daha ileriki dönemlerde kullandığımız örnekler arasında iki temel fark vardır:

1) İlk yıl hem basit hem de gerçekçi problemler üzerinde çalıştık. İlerleyen dönemlerde ise sadece gerçekçi problemler üzerine yoğunlaştık.
2) Gerçekçi problemlerde bir eczanenin nöbet sayısına alt ve üst sınırlar tanımlamıştık. Bunlara ilk tip örnekler diyelim. Ancak gerçek hayatta bir sonraki planlama periyodunda hangi eczanelerin bölge üst sınırda nöbet tutacağı biliniyor. Bu sebeple projenin ilerleyen dönemlerinde her eczanenin nöbet sayısını sabitlemeyi tercih ettik. Bu şekilde gerçeğe daha yakın bir problem elde ettik. Aynı zamanda, hangi eczanelerin alt hangilerinin üst sınırda nöbet tutacağına karar vermek zorunda kalmadığımız için göreceli olarak daha küçük bir çözüm uzayında çalışmaya başladık. Bunlara da ikinci tip örnekler diyelim.

Matematiksel modellerin sonuçlarını ve alt sınırların performanslarını raporladığımız 4.3 Bölümü ve Tabu Arama algoritmasının sonuçlarını raporladığımız 4.4 Bölümü'nde hem basit hem de gerçekçi problemler üzerinde çalıştık. Bu bölümlerdeki gerçekçi problemler için ilk tip örnekler ile deneyler yaptık. Diğer bölümlerde ise sadece gerçekçi problemlerin ikinci tip örnekleri ile çalıştık.

4.2 İzmir Örneklerini Oluşturma

İzmir uygulaması için ihtiyaç duyulan, (i) müşteri düğümlerinin yerleri ve (ii) müşterilerin talep miktarları ile ilgili iki varsayımda bulunduk. Nöbetçi eczaneye ihtiyaç duyan müşterilerin kesin yerlerini belirlemek çok zordur. Bu sebeple aynı mahallede yaşayanları temsil edecek tek bir düğüm olduğunu varsaydık. Bu şekilde geniş bir coğrafyaya yayılmış olan çok fazla sayıdaki gerçek müşteri düğümünü daha az sayıdaki temsili düğümler ile gösterdik. Mahallelerin nüfuslarını da düğümlerin göreceli taleplerini temsil etmek için kullandık. İzmir Büyükşehir Belediyesi sınırları içerisinde 350 mahalle vardır ve her mahalleyi temsil eden düğümün mahalle

muhtarlığında olduğu varsaydık. Mahalle nüfusları bilgisini Türkiye İstatistik Kurumu (TÜİK) İzmir Bölge Müdürlüğü ile yapılan yazışmalar sonrasında satın aldık.

Eczanelerin ve mahalle muhtarlıklarının yerlerini İzmir Büyükşehir Belediyesi Coğrafi Bilgi Sistemi ile belirledik. Müşteri ve eczane düğümleri arasındaki mesafeler, muhtarlıkların ve eczanelerin gerçek koordinatları kullanılarak düz çizgili uzaklıklar (rectilinear) olarak hesapladık. Gerçek hayat uygulamasında, bu mesafeler üzerinde makul değişiklikler yaptık (Örneğin İzmir Körfezi'nin karşılıklı iki yakasındaki düğümler arasındaki mesafeler).

Eczacı Odasından 2010 yılı 3. dönemi ile 2011 yılı ilk dönemine ait bilgileri aldık. Bu bilgiler eczane isimlerini, eczane bölgeleri ve eczanelerin nöbet tarihleridir. Bu bilgiler Tablo 4'de özetlenmiştir. Kapanan veya yeni açılan eczanelerin olması, bazı eczacıların sağlık sebepleri ile nöbet dışı kalmaları gibi sebeplerle dönemler arasında eczane sayıları değişiklik göstermektedir. Planlama dönemindeki resmi tatiller ve hafta sonu sayılarının farklılık göstermesi sebebi ile planlama dönemindeki gün sayısı da farklılık göstermektedir.

Tablo 4. İzmir uygulaması

Dönem	***J***	***T***
2010 / 3	1046	97
2011 / 1	1053	101

4.3 Matematiksel Modellerin Sonuçları ve Alt Sınırlar

Bu bölümde, geliştirilen matematik modeller ve alt sınır algoritmalarının rassal örneklerindeki performanslarını BM1 ve GM1 problemleri için raporladık.

Bu bölümde, BM1 ve GM1 problemlerinin matematik modellerini GAMS 22.5 modelleme yazılımı ile yaptık ve çözüm için CPLEX 11.2 genel çözücüsünü kullandık. Alt ve üst sınır algoritmalarını C programlama dilinde ve Codeblocks ortamında kodladık. Tüm rassal deneyleri AMD Üç Çekirdekli 1.80 GHz işlemciye, 4 GB RAM belleğe ve Windows 7 işletim sistemine sahip bir HP diz üstü bilgisayarda yaptık.

Tüm deneylerde 1 saatlik çözüm süresi sınırı kullandık. Eğer çözücü bu süre içerisinde en iyi sonucu bulduğunu ispatlamamış ise bulduğu en iyi olurlu sonucu ve en iyi alt sınırı raporladık.

Tablo 5'de BM1 problemi ile ilgili sonuçları verdik. İlk üç sütun sırası ile müşteri ve eczane sayıları ile planlama periyodu uzunluğunu gösterir. Dördüncü sütun BM1Alt alt sınırını hesaplamak için gereken CPU zamanını saniye olarak verir. Beşinci sıradaki Aralık sütununda CPLEX çözücüsünün bulduğu en iyi sonucun amaç fonksiyonu değeri ile BM1Alt algoritmasının bulduğu sonucun değeri arasındaki farkın BM1Alt algoritmasının bulduğu sonucun değerine oranını raporladık.

$$Aralık = 100 \times \frac{(CPLEX\ En\ iyi\ Sonuç - BM1Alt)}{BM1Alt}$$

CPLEX CPU sütunu CPLEX çözücüsünün çalışma süresini raporlar. Sürenin karşılaştırılabilir olması için zaman limitine takılmadan sonra eren örneklerin sürelerinin ortalamaları verdik. Son sütun ise 10 rassal örnekten bir saatlik süre limiti içerisinde optimal olarak çözülenlerin adedini gösterir.

Sonuçlardan anlaşılacağı üzere BM1Alt algoritması oldukça hızlı çalışmakta ve saniyenin binde ikisi kadar sürede sonuç vermektedir. Bu algoritmanın önerdiği alt sınır bilinen en iyi sonuca çok yakın değerler almaktadır. Bu değer CPLEX çözücüsünün optimal sonucu bulduğu örnekler için ortalama %0,39 ve en fazla %0,97'dir. Ancak CPLEX çözücüsünün süre limitine takıldığı örnekler için aralık ortalama %0,74 ve en fazla %3,32'dir. Çözüm sürelerinin *J* ve *T* değerlerinin artması ile etkilendiği gözlemlenmektedir.

Tablo 6 GM1 problemi ile ilgili sonuçları raporlar. İlk dört sütun sırası ile müşteri, eczane ve nöbet bölgesi sayıları ile planlama periyodu uzunluğunu vermektedir. Beşinci sütun GM1Alt2 alt sınırını hesaplamak için gereken CPU zamanını saniye olarak gösterir. GM1Alt2 alt sınırını hem kısa sürede çalışması hem de GM1Alt1 alt sınırından daha iyi sonuç vermesi sebebi ile raporladık, aynı sebeple GM1Alt1 alt sınırını çalışmanın devamında kullanmadık.

Altıncı sıradaki Aralık sütunu CPLEX çözücüsünün bulduğu en iyi sonucun amaç fonksiyon değeri ve GM1Alt2 algoritmasının bulduğu değer kullanılarak hesaplanmış ve raporlanmıştır. CPLEX CPU sütunu CPLEX çözücüsünün çalışma süresini raporlamaktadır. Sürenin karşılaştırılabilir olması için zaman limitine takılmadan sonra eren örneklerin sürelerinin ortalamaları verilmiştir. Son sütun ise 10 rassal örnekten bir saatlik süre limiti içerisinde optimal olarak çözülenlerin adedini vermektedir.

Tablo 5. BM1 problemi sonuçları

I	*J*	*T*	*BM1Alt* CPU (sn)	Aralık (%)	CPLEX CPU (sn)*	Optimal
20	10	5	0,001	0,38	0,74	10
20	20	10	0,001	0,38	22,59	10
20	30	15	0,001	0,48	693,59	6
40	20	5	0,001	0,40	15,59	10
40	40	10	0,002	0,38	-	0
40	60	15	0,001	0,36	-	0
60	30	5	0,001	0,37	183,64	10
60	60	10	0,001	0,45	-	0
60	90	15	0,002	1,72	-	0

* 3600 saniye limitinden önce optimal sonuca ulaşılan örneklerin CPU zamanı ortalaması

Tablo 6. GM1 problemi sonuçları

I	*J*	*T*	*K*	GM1Alt2 CPU (sn)	Aralık (%)	CPLEX CPU (sn)*	Optimal
20	10	5	4	0,001	4,30	0,39	10
20	20	10	4	0,001	7,09	226,58	10
20	30	15	9	0,001	1,31	107,84	8
40	20	5	4	0,001	8,41	7,38	10
40	40	10	9	0,001	4,81	1491,02	4
40	60	15	9	0,001	5,34	-	0
60	30	5	9	0,001	7,24	23,86	10
60	60	10	9	0,001	11,20	-	0
60	90	15	9	0,002	13,60	-	0

* 3600 saniye limitinden önce optimal sonuca ulaşılan örneklerin CPU zamanı ortalaması

Sonuçlardan anlaşılacağı üzere GM1Alt2 algoritması oldukça hızlı çalışmakta ve saniyenin binde ikisi kadar sürede sonuç vermektedir. Bu algoritmanın önerdiği alt sınırın kalitesi (bilinen en iyi sonuca yakınlığı) BM1Alt algoritmasına göre daha düşüktür. Bu farkın sebebinin GM1

probleminin birden fazla nöbet içermesi olduğu düşünülebilir. Bu değer CPLEX çözücüsünün optimal sonucu bulduğu örnekler için ortalama %5,90 ve en fazla %16,12'dir. Ancak CPLEX çözücüsünün süre limitine takıldığı örnekler için aralık ortalama %8,75 ve en fazla %25,15'dir. Nöbet bölgesi sayısının artması problemin çözüm süresini artırmaktadır. Ayrıca çözüm sürelerinin *J* ve *T* değerlerinin artması ile arttığı gözlemlenmektedir.

4.4 Tabu Arama

Bu bölümde, tek nöbetli (BM1) ve çok nöbetli (GM1) problemler için yapılan deneylerin sonuçları sunduk. Deneyleri AMD Triple Core 1.80 GHz işlemcili 4GB RAM 'e sahip Windows 7 Professional yüklü HP Laptop bilgisayarda yaptık. TA ve alt sınır algoritmalarını C++ dilinde ve Codeblocks ortamında kodladık.

Testlere başlamadan önce, tabu süresi ve maksimum döngü sayısının en iyi değerlerini bulmak için ön deneyler yaptık. Ön deneyler boyunca 6 farklı problem boyutu kullandık. Ön deneyler boyunca tabu süresi ve döngü sayısı parametrelerini test ettik.

Tek nöbetli problem için, gün ve düğüm tabu süresini 5, döngü sayıları ise iç döngü için 15 ve dış döngü için 30 olarak seçtik. Çok nöbetli problem için, gün tabu süresini 5, düğüm tabu süresi 5, bölge tabu süresi 3 ve döngü sayıları iç ve dış döngü için sırasıyla 10 ve 20 olarak belirledik.

Ön deneyler gerçekleştirildikten sonra, TA algoritmasının performansı test ettik. Tek nöbetli problem için hazırlanan Tablo **7**'de, *I*, *J* ve *T* sütunları sırasıyla müşteri, eczane ve gün sayılarını temsil etmektedir. LB, TA ve OPT sütunları sırasıyla alt sınır, TA algoritması ve matematik modelin çalışma sürelerini göstermektedir. Beklendiği üzere, çalışma süresi problem boyutu arttıkça çok artmaktadır. Matematik model için çözüm süresi limiti 1 saat olarak belirlenmiştir, 3600 saniyelik çözüm süresi probleme bu süre limiti içinde optimal çözüm bulunamaması anlamına gelmektedir. Matematik modelin optimal olarak çözebildiği örnek sayısı # sütununda gösterilmektedir. Eczane sayısı, *J*, 40 ve daha fazla olduğu durumlarda matematik model problemleri süre limiti içinde çözememektedir. Gerçek Aralık sütunu TA algoritmasının bulduğu sonuç ile matematik modelin bulduğu sonucun arasındaki mesafeyi yüzde olarak göstermektedir. Algılanan Aralık sütunu ise TA algoritmasının bulduğu sonuç ile alt sınır algoritmasının bulduğu değerin arasındaki mesafeyi yüzde olarak göstermektedir. Algılanan

aralık ölçütü özellikle matematik model ile çözülemeyen büyük boyutlu problemlerde alt ve üst sınır arasındaki farkı ölçmek amacıyla kullanılmaktadır.

Tablo 7. Tek nöbetli problem için deney sonuçları

I	*J*	*T*	**LB (saniye**	**TA (saniye)**	**OPT (saniye)**	**#**	**Gerçek Aralık (%)**	**Algılanan Aralık (%)**
20	10	5	0,001	0,299	0,740	10	0,0	0,4
20	20	10	0,001	1,911	22,591	10	0,2	0,6
20	30	15	0,001	4,533	1.856,153	6	0,2	0,7
40	2	5	0,001	3,355	15,589	10	0,1	0,5
40	40	10	0,002	24,097	3.600,000	0	0,5	0,7
40	60	15	0,001	45,130	3.600,000	0	1,2	1,3
60	30	5	0,001	13,883	183,641	10	0,4	0,8
60	60	10	0,001	103,670	3.600,000	0	1,2	1,3
60	90	15	0,002	165,894	3.600,000	0	2,6	2,6

Tek nöbetli problem için çözüm süreleri ve çözüm farkları Tablo C2'de özetlenmiştir. TA algoritması tarafından bulunan sonuçlar çoğunlukla optimal çözümün %1 aralığındadır. TA algoritmasının ve matematik modellerin çözüm sürelerini karşılaştıracak olursak, TA algoritması açık bir şekilde çok daha az bir sürede sonuçlar bulmaktadır ve bu sonuçların kalitesi matematik modelinkinden uzak değildir.

Tablo 8. Tek nöbetli problem için ortalama sonuçlar

	Gerçek	**Algılanan**
Aralık (%)	0,717	0,974
	TA	**OPT**
Çözüm süresi (sn)	40,308	1830,968

Çok nöbetli problem için yapılan deneyler, küçük ve büyük boyutlu olarak ikiye ayrılmıştır. İzleyen iki bölümde, ilk olarak küçük boyutlu problemler için deney sonuçları tartışılmakta ve

sonrasında büyük boyutlu problemleri çözmek için TA algoritmasında yapılan değişiklikler ve deney sonuçları tartışılmaktadır.

Sonuçlar, çözüm süresi ve alt sınır ile optimal çözüm arasındaki fark bakımından karşılaştırılmıştır. Eğer optimal çözüm mevcut değilse, üst sınırlar karşılaştırma için kullanılmıştır.

Küçük Boyutlu Problemler

Tablo 9'da, *I*,*J*,*T* ve *K* sütunları sırasıyla müşteri, eczane, gün ve bölge sayılarını temsil etmektedir. Sonraki dört sütun, sırasıyla alt sınır, matematik modelin doğrusal gevşetimi, TA algoritması ve matematik modelin çalışma sürelerini göstermektedir. Matematik model 60 veya daha fazla eczanenin olduğu problemleri 1 saatlik süre limitinde optimal çözememektedir. Matematik modelin optimal olarak çözebildiği örnek sayısı # sütununda gösterilmektedir. Problem boyutu arttıkça çözüm süresi uzamaktadır. Bununla birlikte, daha büyük problemler için bile önerilen TA algoritması ortalamada 1 dakikadan kısa sürede sonuç vermektedir.

Tablo 10 TA ve matematik model için ortalama fark ve çözüm süresi değerlerini özetler. TA algoritması çok kısa sürede kabul edilebilir sonuçlar bulabilmektedir. Algılanan aralık değerini %7.6 civarında olmasına rağmen, gerçekte bu değer %1'in altındadır. Bu durumda TA algoritmasının performansının kabul edilebilir olduğu sonucuna varılabilir. Ayrıca farkın TA algoritmasından değil, alt sınır algoritmasının yeteri kadar iyi sonuç vermemesi sebebi ile olduğu düşünülebilir.

Büyük-Boyutlu Problemler

Büyük boyutlu problemler için TA algoritmasında bazı değişiklikler yapılmıştır. Mevcut algoritmada, komşu çözüm arama tekniklerinde her eczane için komşu çözüm araması yapılmaktadır. Büyük boyutlu problemlerde eczane sayısı çok fazla olduğu için, makul çözüm sürelerinde sonuçlar bulmak çok zor olmaktadır.

İlk değişiklik olarak, en iyi aday çözüm üzerinde iyileştirmenin görüldüğü döngü sayısı tutulmaya başlamıştır. Bu değer önceden belirlenen bir seviyeye ulaştığında, seçilen eczane için daha iyi sonuçların aranması durdurulup diğer eczaneye geçilir. Bu şekilde, bir eczane için yapılan

arama kısıtlanmış olup, en iyi aday sonucu bulmaya çalışılmaz. Sadece yeterince iyi bir çözüm bulunduktan sonra diğer eczaneler aranarak yeni çözümlere bakılır.

Tablo 9. Küçük boyutlu çok nöbetli problem için deney sonuçları

I	*J*	*T*	*K*	**LB (saniye)**	**LP (saniye)**	**TA (saniye)**	**OPT (saniye)**	#	**Gerçek Aralık (%)**	**Algılanan Aralık (%)**
20	10	5	4	0,001	0,249	0,066	0,387	10	0,0	4,3
20	20	10	4	0,001	0,789	0,651	226,578	10	0,1	7,2
20	30	15	9	0,001	1,241	3,663	806,803	8	0,2	1,5
40	20	5	4	0,001	0,663	0,376	7,385	10	0,2	8,6
40	40	10	9	0,001	4,850	5,222	2.758,768	4	0,9	5,7
40	60	15	9	0,001	27,124	18,444	3.601,574	0	2,0	7,0
60	30	5	9	0,001	1,155	1,514	23,863	10	0,0	7,3
60	60	10	9	0,001	38,010	13,486	3.601,745	0	1,9	12,8
60	90	15	9	0,002	252,658	46,058	3.603,627	0	3,3	14,6

Tablo 10. Küçük boyutlu çok nöbetli problem için ortalama sonuçlar

	Gerçek	**Algılanan**
Aralık (%)	0,961	7,667
	TA	**OPT**
Çözüm süresi (sn)	9,942	1.625,637

Diğer bir değişiklik olarak, çok nöbetli problemin bölge kavramı kullanılmaktadır. Değiştirilmiş TA algoritmasında, aranılacak bölgeler dış döngü numaraları ile ilişkilendirilmiştir. Bu yolla, her dış döngü her zaman baştan başlamak yerine değişik bir eczane kümesinden başlar. Böylece, daha iyi çözümler daha geniş bir uzay araştırılmış olur. Bu sebeple, maksimum iç döngü sayısı *K* olarak değiştirilerek her döngüde bir bölge taranabilmektedir. Gendreau vd. (1997), Gendreau vd. (2005), ve Doerner vd. (2005) tanımladıkları komşuluk yöntemlerinde olası tüm komşuları aramak yerine daha az komşuyu arayabilmek için aday çözümlerin sadece belirli bir uzaklıkta olmasına izin vermişlerdir. Yukarıda bahsedilen yöntem de benzer bir amaç gütmekte ve bu

amacı probleme özgü bir yöntem ile gerçekleştirmiştir, her dış döngü için çözüm uzayının bir kısmı incelenmiştir.

Tablo 11. Büyük boyutlu çok nöbetli problem için sonuçlar

I	***J***	***T***	***K***	**Örnek #**	**LB (saniye)**	**TA (saniye)**	**Algılanan Aralık (%)**
350	250	25	49	1	0,008	17.180	10,0
				2	0,008	17.194	10,4
				3	0,008	17.326	11,6
				4	0,008	16.939	10,4
				5	0,009	16.632	9,5
350	500	50	49	1	0,021	120.159	17,7
				2	0,020	134.734	20,6
				3	0,017	76.620	16,8
				4	0,016	75.793	15,4
				5	0,016	82.272	19,1
350	1000	100	49	1	0,032	540.240	49,6
				2	0,016	587.043	50,9
				3	0,031	537.777	40,6
				4	0,031	540.238	51,9
				5	0,016	576.574	47,8

Gerçek Hayat Problemleri

Büyük boyutlu problemlerin için oluşturulan 15 rassal örneğin test sonuçları Tablo 11'de verilmiştir. Optimal ve doğrusal gevşetme değerlerini süre limiti içinde elde edilemediği için, bu değerlere tabloda yer verilmemiştir. TA algoritması en küçük problem için 16.000 saniyeden (~ 4.5 saat) uzun sürede sonuç vermektedir. Ancak TA algoritmasının süresindeki artış eczane sayısının artışından çok daha hızlı olmaktadır. Algılanan aralık değerleri ise özellikle büyük problemlerde %50 mertebesine ulaşmaktadır. En büyük problem boyutu, gerçek hayat problemine benzer büyüklüktedir. Bu boyuttaki bir problem için TA algoritması 6 günden uzun süre çalışmaktadır.

Mevcut ve önerilen çözümler iki dönem için karşılaştırılmış ve sonuçlar Tablo **12**'de verilmiştir. Her iki dönem için GM1 Alt 2 algoritması ile bulunan alt sınır LB satırında verilmiştir. Mevcut çözüm ve TA algoritması ile bulunan çözüm ilgili satırlarda verilmiş ve alt sınırdan uzaklığı hesaplanmıştır. 2010 yılı 3. döneminde ciddi bir iyileştirme potansiyeli olduğu açıktır. 2011 yılı ilk dönemi için de %1.1 civarında bir iyileştirme mümkündür.

Tablo 12. Tabu arama ile İzmir uygulaması

Planlama Dönemi	**Açıklama**	**Maliyet (Milyon km)**	**Çözüm Süresi (saniye)**	**Aralık (%)**
2011 / 1	Mevcut	410.89	-	39.8
	TA	317.12	55.428,838 (~15.4 Saat)	7.9
	LB	293.92	0,087	-
2010 / 3	Mevcut	309.01	-	9.4
	TA	305.87	36.624,911 (~10 Saat)	8.3
	LB	282.54	0,085	-

Her iki periyot için de, tüm insanların planlama periyodunun her bir gününde eczanelere talebinin olacağı varsayılmıştır. Bu varsayımın gerçek hayatta geçerliliği yoktur. Ancak, mevcut ve önerilen çözümler aynı yöntem ile ölçüldüğü için karşılaştırmanın adil olduğu düşünülmektedir.

4.5 Dal-Sınır Algoritması

Bu bölümde ENÇDS ve Geliştirilmiş ENÇDS algoritmaları ile yaptığımız deneylerin sonuçlarını sunduk. Her iki algoritmayı C dilinde kodladık ve deneyleri Windows 7 Professional yüklü Intel Core2Duo işlemcili HP Desktop, 3.00GHz, 4GB RAM özelliklerine sahip bir bilgisayarda gerçekleştirdik.

Deneyleri 9 farklı problem büyüklüğünde yaptık. Her problem büyüklüğü için 10 farklı örnek rassal çözdük. Deneyler için bir saatlik süre limiti belirledik. Algoritmanın optimal çözüm veremediği örneklerde bir saat içerisinde bulduğu en iyi sonucu raporladık.

4.5.1 ENÇDS Algoritması

Bu bölümde ENÇDS algoritması ile yapılan deneyleri sunduk. Deney sonuçlarını öncelikle üst sınır algoritmalarının en iyisini belirlemek amacıyla analiz ettik. Daha sonra ise en iyi üst sınır algoritması ile IBM ILOG CPLEX 12.0 genel amaçlı çözücünün performanslarını karşılaştırdık.

ÜS4 üst sınır algoritması için ön çalışmalar yaptık. Bu çalışmalarda ÜS4 algoritmasının ihtiyaç duyduğu iki parametre α ve β sırası ile %50 ve %25 olarak belirlenmiştir. Bu şekilde dal-sınır ağacında planlama ufkunun yarısına gelene kadar ÜS3 algoritması ile üst sınır bulunacak kalanında ise ÜS1 algoritması kullanılacaktır. Ayrıca ÜS3 algoritması tüm mahalleleri değil, en kritik %25 mahalleyi inceleyecektir. ÜS4 algoritması ÜS3 algoritmasına göre hem süre hem de çözüm kalitesi açısından daha iyi çalışmaktadır. Bu sebeple ÜS3 algoritması ile ilgili sonuçlar raporlanmamıştır.

ENÇDS algoritması her bir üst sınır algoritması ile ayrı ayrı koşulup, üst sınır algoritmalarının performansları karşılaştırılmıştır. Sonuçları Tablo 13'de ve Tablo 14'de raporlanmıştır. Her iki tabloda da I, J, T ve K sütunları sırasıyla müşteri, eczane, planlama ufku ve bölge sayısını temsil etmektedir. Tablo 13'de ÜS1, ÜS2 ve ÜS4 sütunları bu üst sınır algoritması ile oluşturulan dal-sınır algoritmasının ortalama CPU zamanını vermektedir. Bu ortalamaya süre limiti sebebi ile duran örnekler dahil edilmemiştir. Süre limiti sebebi ile durulan örnek sayısı parantez içinde verilmiştir. Tablo 13'den de görüleceği üzere algoritmaların süresi müşteri sayısının artışından çok fazla etkilenmemekte ancak eczane ve planlama ufkunun artması ile hızla artmaktadır.

Tablo 14'de ÜS1, ÜS2 ve ÜS4 sütunları bu üst sınır algoritması ile oluşturulan dal-sınır algoritmasının 10 örnekten kaçında üçü arasındaki en iyi sonucu verdiğini göstermektedir. Bu tabloda amaç süre limiti içinde çözülemeyen örneklerde en iyi olurlu çözüm bulan algoritmayı belirlemektir. Örneğin, ilk problem büyüklüğünde tüm algoritmalar optimal çözümü verdikleri için hepsi 10 değerini almıştır. Ancak, 3. problem büyüklüğünde (I=20, J=30, T=5 ve K=4) ÜS1 ve ÜS2 algoritmaları sadece 4 defa en iyi sonucu verirken ÜS4 algoritması 7 kere en iyi sonuca ulaşmıştır. Toplamda 90 örneğin 83'ünde ÜS4 algoritması en iyi sonucu vermiştir.

Tablo 13. Üst sınır algoritmalarının zaman ortalamaları

I	*J*	*T*	*K*	*ÜS1 (sn)*	*ÜS2 (sn)*	*ÜS4 (sn)*
20	10	5	4	0,004	0.007	0.008
20	20	10	4	524,052 (2)	474,680 (1)	388,584 (1)
20	30	15	9	– (10)	– (10)	– (10)
40	20	5	4	4,209	0,079	0,080
40	40	10	9	– (10)	– (10)	– (10)
40	60	15	9	– (10)	– (10)	– (10)
60	30	5	9	445,708 (1)	88,510 (1)	88,208 (1)
60	60	10	9	– (10)	– (10)	– (10)
60	90	15	9	– (10)	– (10)	– (10)

Tablo 14. Üst sınır algoritmalarının göreceli karşılaştırması

I	*J*	*T*	*K*	*ÜS1*	*ÜS2*	*ÜS4*
20	10	5	4	10	10	10
20	20	10	4	9	10	10
20	30	15	9	4	4	7
40	20	5	4	10	10	10
40	40	10	9	2	7	10
40	60	15	9	4	3	8
60	30	5	9	9	10	10
60	60	10	9	0	7	10
60	90	15	9	2	1	8
Toplam				50	62	83

ÜS1 algoritması rassal bir şekilde olurlu çözümler oluşturduğu için her bir düğümde hızlı çalışmaktadır, fakat aynı sebeple de düşük kalitede sonuçlar vermektedir. ÜS2 algoritması ise içerisindeki küme operasyonları dolayısıyla çok fazla zaman harcamaktadır. Bu noktadan sonraki deneylerde dal-sınır algoritmasında (ENÇDS) ÜS4 algoritmasının kullanılmasına karar verilmiştir.

Tablo 15'de ENÇDS algoritmasının sonuçları CPLEX çözücüsünün bulduğu çözümlere göreceli olacak şekilde raporlanmıştır. İlk dört sütun problem büyüklüğü bilgileridir. CPLEX ve ENÇDS sütunları sırasıyla CPLEX ve ENÇDS tarafından 3600 saniyelik süre limiti içerisinde 10 örnekten kaçı için optimal çözümü elde edebildiklerini göstermektedir. Sonraki iki sütun CPLEX'in örnekler içerisindeki minimum ve maksimum çözüm sürelerini temsil etmektedir. Bir sonraki sütun, # sütunu, ENÇDS algoritmasının CPLEX'den daha kısa sürede optimal sonucu bulduğu örneklerin sayısını göstermektedir. Son iki sütun, CPLEX ile bulunan en iyi sonuç ile ENÇDS ile bulunan en iyi sonuç arasındaki aralığın en az ve en fazla olduğu değerleri raporlamaktadır. Değerler ENÇDS sonucunun CPLEX sonucundan çıkartılması ile elde edilen farkın CPLEX sonucuna bölünmesiyle bulunmuştur.

Tablo 15. ENÇDS ve CPLEX karşılaştırması

I	*J*	*T*	*K*	***Optimal***		***CPLEX Süre (sn)***		#	***Aralık (%)***	
				CPLEX	***ENÇDS***	***En az***	***En çok***		***En az***	***En çok***
20	10	5	4	10	10	0,02	0,78	10	0,00	0,00
20	20	10	4	9	9	0,13	3600,00	5	0,00	0,00
20	30	15	9	10	1	0,81	918,44	0	0,01	3,97
40	20	5	4	10	9	0,87	11,65	10	0,00	0,00
40	40	10	9	7	0	18,81	3600,00	0	1,34	3,75
40	60	15	9	0	0	3600,00	3600,00	0	3,36	7,77
60	30	5	9	10	10	0,87	43,85	0	0,00	0,00
60	60	10	9	0	0	3600,00	3600,00	0	3,17	6,45
60	90	15	9	0	0	3600,00	3600,00	0	3,51	7,62

Küçük problemlerde ENÇDS CPLEX'ten daha hızlı bir şekilde optimal çözümü bulmaktadır. Ancak planlama ufkunun 10 gün olduğu durumlarda bile optimal çözümün iki yaklaşım tarafından da bulunmadığı durumlar vardır. Problem boyutunun büyümesi ile beraber ENÇDS algoritmasının bulduğu sonuçlar CPLEX sonuçlarından uzaklaşmaktadır.

4.5.2 Geliştirilmiş ENÇDS Algoritması

Bu bölümde geliştirilmiş ENÇDS algoritmasını ile yaptığımız deney sonuçlarını raporladık. Düğüm seçme yöntemleri ve problem büyüklüğünü azaltma algoritmaları eczane nöbet

çizelgeleme dal-sınır algoritması (ENÇDSA)'na entegre edilmiştir. Problem büyüklüğünü azaltma algoritmaları orijinal veri dosyalarını okuduktan sonra çalıştırılmış, güncellenen veri dosyaları dal-sınır algoritmasına verilmiştir. Bu çalışmalara ek olarak, kök düğümde üst sınır değeri değişken komşuluk arama algoritmasının(DKA) bulduğu en iyi çözümün amaç fonksiyonu değeri olarak başlatılmıştır. Bu şekilde, algoritmanın iyi bir üst sınır değeri ile başlamasının performansına etkisi gözlemlenmek istenmiştir. DKA'nın çözüm süresi dal-sınır algoritmasının çözüm süresine eklenerek toplam süre raporlanmıştır. Elde edile dal-sınır algoritması GENÇDSA (Geliştirilmiş Eczane Nöbet Çizelgeleme Dal-Sınır Algoritması) olarak adlandırılmıştır.

Tablo 16'da ilk dört sütun problem büyüklüğünü anlatmaktadır. Sonraki iki sütun ise sırasıyla iki algoritmanın CPLEX'in bulduğu en iyi sonuçlar ile en az ve en çok ne kadar aralık değerine sahip olduğunu göstermektedir. Her iki algoritmanın her problem büyüklüğü için buldukları optimal çözüm sayısı aynı olduğu için tabloda gösterilmemiştir. Bu değerlerin aynı olması ise küçük problemlerde her iki algoritmanın da optimal çözümleri bulabilmesi ancak büyük problemlerde her ikisinin de başarısız olması ile açıklanabilir. Bu sebeple, sadece aralık değerleri gelişme göstermektedir. Tabloda görülebileceği gibi, GENÇDSA kullanılması ile problem boyutu arttıkça aralık değerleri önemli ölçüde iyileşme göstermektedir. Ancak, buradaki başarı dal-sınır algoritmasından daha çok DKA algoritmasının bulduğu yüksek kalitedeki çözümlerin dal-sınır algoritması başlangıcında kullanılması ile sağlanmış olabilir. Tablo 17 yardımıyla bu konuda daha fazla yorum yapabileceğiz.

Tablo 17'de GENÇDSA sonuçları CPLEX çözücüsünün bulduğu çözümlere göreceli olacak şekilde raporlanmıştır. İlk dört sütun problem büyüklüğü bilgileridir. CPLEX ve DS sütunları sırasıyla CPLEX ve GENÇDSA tarafından 3600 saniyelik süre limiti içerisinde 10 örnekten kaçı için optimal çözümü elde edebildiklerini göstermektedir. MinSüre ve Maxsüre adlı sütunlar CPLEX'in örnekler içerisindeki minimum ve maksimum çözüm sürelerini temsil etmektedir. Min_Gap ve Max_Gap sütunları, CPLEX ile bulunan en iyi sonuç ile GENÇDSA ile bulunan en iyi sonuç arasındaki aralığın sırasıyla en az ve en fazla olduğu değerleri raporlamaktadır. Değerler GENÇDSA sonucunun CPLEX sonucundan çıkartılması ile elde edilen farkın CPLEX sonucuna bölünmesiyle bulunmuştur. İmp sütunu ise, GENÇDSA'nın CPLEX'den daha kısa sürede optimal sonucu bulduğu örneklerin sayısını göstermektedir. Son sütun ise, GENÇDSA'nın DKA sonuçlarını kaç örnekte geliştirebildiğini göstermektedir.

Tablo 16. GENÇDSA ve ENÇDSA karşılaştırması

I	*J*	*T*	*K*	***ENÇDSA Aralık (%)***		***GENÇDSA Aralık (%)***	
				En az	***En çok***	***En az***	***En çok***
20	10	5	4	0,00	0,00	0,00	0,00
20	20	10	4	0,00	0,00	0,00	0,00
20	30	15	9	0,01	3,97	0,00	0,06
40	20	5	4	0,00	0,00	0,00	0,00
40	40	10	9	1,34	3,75	0,00	0,24
40	60	15	9	3,36	7,77	0,10	0,35
60	30	5	9	0,00	0,00	0,00	0,00
60	60	10	9	3,17	6,45	0,25	0,81
60	90	15	9	3,51	7,62	0,04	0,43

Tablo 17. GENÇDSA ve CPLEX karşılaştırması

I	*J*	*T*	*K*	*CPLEX*	*MinSüre (sn)*	*MaxSüre (sn)*	*DS*	*Min GAP (%)*	*Max GAP (%)*	*imp*	*DKA imp*
20	10	5	4	10	0,02	0,78	10	0,00	0,00	9	1
20	20	10	4	9	0,13	3600	10	0,00	0,00	7	5
20	30	15	9	10	0,81	918,44	0	0,00	0,06	0	2
40	20	5	4	10	0,87	11,65	10	0,00	0,00	10	7
40	40	10	9	8	18,81	3600	0	0,00	0,24	0	2
40	60	15	9	1	2418,8	3600	0	0,10	0,35	0	0
60	30	5	9	10	0,87	43,85	10	0,00	0,00	8	10
60	60	10	9	1	1362	3600	0	0,25	0,81	0	0
60	90	15	9	0	3600	3600	0	0,04	0,43	0	0

Son sütundaki bilgiler ışığında, dal-sınır algoritmasının DKA'nın kısa sürede bulduğu yüksek kalitedeki çözümleri büyük çaplı problemlerde bir saatlik süre içerisinde geliştiremediği gözlemlenmiştir. Yapılan iyileştirme çalışmalarına rağmen probleme özgü geliştirilen dal-sınır

algoritmasının ENÇ için kesin çözüm yöntemi olarak kullanılmasının pratikte mümkün olmadığı sonucuna varılabilir. Çünkü, gerçek problem boyutu yukarıda deneyleri yapılan rassal örneklere kıyasla çok büyüktür.

Dal-sınır algoritmasının istenilen performanstan uzak olması sebebi ile bir sonraki bölümde ENÇ için geliştirilen başka bir kesin çözüm yöntemi anlatılmaktadır.

4.6 Dal-Fiyat Algoritması

Bu bölümde Eczane Nöbet Çizelgeleme Dal-Fiyat Algoritması (ENÇDFA) ve Geliştirilmiş ENÇDS algoritmaları ile yaptığımız deneylerin sonuçlarını sunduk.

Algoritmalar C++ dilinde kullanılmış olup IBM ILOG CPLEX 12.5 çözücüsü ile iletişim Concert Teknoloji kullanılarak kurulmuştur. Deneylerin yapıldığı örnekler dal-sınır algoritması için kullanılan örneklerdir. Deneyleri Windows 7 Professional yüklü Intel Core2Duo işlemcili HP Desktop, 3.00GHz, 4GB RAM özelliklerine sahip bir bilgisayarda gerçekleştirdik.

4.6.1 ENÇDF Algoritması

ENÇDFA için test sonuçlarını raporlamadan önce Kolon türetme algoritması (KTA) tarafından bulunan alt sınırların kalitesini göstermek için Tablo 18 hazırlanmıştır. Bu tabloda örnek büyüklüklerini bildiren ilk dört sütundan sonra gelen ilk sütun, o problemin 10 test örneğinden kaç tanesinin en iyi çözümünün bilindiğini göstermektedir. “KTA en iyi” isimli sütun ise KTA algoritmasının elde ettiği en iyi amaç fonksiyonu değerinin test örneklerinin kaç tanesinde en iyi çözümünün amaç değerine eşit olduğunu göstermektedir. Bu örneklerde, KTA en iyi çözümün amaç fonksiyonu değerini elde etmesine rağmen bulduğu çizelge olurlu bir çizelge olmamaktadır. “KTA En İyi Tamsayı” sütunu ise KTA’ nın en iyi çözümü bulduğu örnek sayısını vermektedir. Sonraki iki sütun sırası ile KTA’nın ortalama çözüm süresini ve bulduğu ortalama kolon sayısını vermektedir. Son sütunda ise KTA’nın bulduğu alt sınır değeri LB_AO algoritması ile bulunan alt sınır değeri ile karşılaştırılmış ve yüzde iyileşme miktarı raporlanmıştır.

Tablo 18. KTA ile bulunan alt sınır değerlerinin kalitesi

I	J	T	K	Bilinen Enİyi	KTA Enİyi	KTA Enİyi Tam.	EnAz Süre	Ort. Süre	EnÇok Süre	Süre St.Sap.	Ort. Kolon Sayısı	Ort. İyileşme
20	10	5	4	10	10	10	0.22	0.46	0.89	0.19	13.10	0.47
40	20	5	4	10	10	10	3.80	5.32	6.44	0.86	43.20	0.57
60	30	5	9	10	10	5	6.55	11.96	16.29	3.29	87.30	0.27
20	20	10	4	10	10	7	1.62	3.61	5.14	1.11	42.80	0.42
20	30	15	9	10	10	0	2.84	4.81	8.13	1.56	67.30	0.19
40	40	10	9	10	10	0	14.20	26.43	42.28	9.99	137.50	0.45
40	60	15	9	9	9	1	74.55	101.64	126.81	18.98	290.80	0.45
60	60	10	9	10	7	2	97.20	133.16	191.24	31.41	340.10	0.51
60	90	15	9	7	4	0	326.92	594.89	1095.83	273.80	579.10	0.42

Yukarıda deney sonuçları raporlanan örneklerin sayısı 90 tanedir. Bu örneklerin 86 tanesinin en iyi çözümü bilinmektedir. KTA 80 örnekte en iyi çözümün amaç fonksiyonu değerini bulmayı başarmıştır. (% 96) Buna ek olarak, 35 örnekte de en iyi çözümü bulabilmiştir. Diğer bir deyişle, KTA en iyi çözümü bilinen örneklerin %42'sinde en iyi çözümü bulabilmektedir.

Son sütündaki değerler ile, KTA'nın LB_AO algoritmasına göre iyileştirmeler gösterdiği görülmektedir. Büyük problemlerde ve gerçek boyuttaki problem için alt sınırın kalitesi değişik sezgisellerin kalitesini görmek için önemli olmaktadır. Dolayısı ile KTA ile elde edilen daha kaliteli alt sınırlar ile sezgisellerin performansı daha iyi değerlendirilebilir.

Tablo 19'da ise dal-fiyat algoritması ile yapılan deneylerin sonuçları IBM ILOG CPLEX sonuçları ile karşılaştırmalı olarak raporlanmıştır. Bu deneylerde kök düğümden sonra bulunan kolonlar ile AP tamsayı kısıtları ile çözülmüş ve dallandırmaya geçmeden önce iyi bir üst sınır çözümü elde edilmesi planlanmıştır. Her iki yöntem için de sırası ile toplam 10 örnekten kaç tanesinin 9000 saniye süre limiti altında en iyi çözümünü bulabildikleri "opt" sütununda verilmektedir. "Ort. S" sütunu altında ise süre limiti altında en iyi çözümü bulunamayan örnekler için bulunan en iyi sonucun alt sınır değerinden uzaklığı yüzde cinsinden verilmiştir. "Ort. DS" ise açılan düğüm sayısını vermektedir. Süre kısıtı altında en iyi çözümü bulunan örnekler için özüm sürelerinin sırasıyla en az, ortalama, standart sapma ve en çok değerleri verilmiştir. En son sütunda iyi dal-fiyat algoritmasının örneklerin kaç tanesinde IBM ILOG CPLEX'nden daha iyi sonuç verdiği

raporlanmıştır. Dal-fiyat algoritması en iyi çözümü daha kısa sürede bulduysa ya da iki yöntem de süre limitine ulaştığında dal-fiyat algoritmasının üst sınır çözümü daha küçük ise, dal-fiyat algoritmasının daha iyi performans verdiği düşünülmüştür.

Tablo 19. ENÇDFA deney sonuçları

Test Örneği				ENÇDFA							IBM ILOG CPLEX						
				Çözüm Özellikleri			Süre				Çözüm Özellikleri		Süre				
I	J	T	K	Opt	Ort. S.	Ort. DS	EnAz	Ort.	St. S.	EnÇok	Opt	Ort. S.	EnAz	Ort. Süre	St. S.	EnÇok	İyi
20	10	5	4	10	-	0.0	0.2	0.5	0.2	0.9	10	-	0.0	0.2	0.2	0.8	1
40	20	5	4	10	-	0.0	3.8	5.3	0.8	6.4	10	-	0.9	4.2	2.8	11.7	2
60	30	5	9	10	-	2.0	10.5	14.4	1.6	16.5	10	-	0.9	14.2	16.1	43.9	4
20	20	10	4	10	-	1.6	2.6	4.0	1.0	6.5	10	-	0.0	1.4	1.5	4.8	1
20	30	15	9	10	-	63.8	7.1	17.3	6.7	28.0	10	-	0.8	100.5	273.0	918.4	2
40	40	10	9	10	-	109.6	25.1	62.0	35.0	146.8	8	0.00	18.8	985.8	977.5	2685.4	7
40	60	15	9	9	0.55	7338.4	112.9	1610.7	2839.1	8765.2	0	0.01	-	-	-	-	9
60	60	10	9	10	-	8.2	102.7	165.9	49.6	293.8	0	0.01	-	-	-	-	10
60	90	15	9	4	8.61	7581.7	505.6	2849.9	2276.5	6539.6	0	0.18	-	-	-	-	5

“İyi” sütunu incelendiğinde ENÇDFA‘nın büyük problemlerde daha iyi performans gösterdiği görülebilir. CPLEX son üç problemde hiçbir örnekte en iyi çözümü bulamazken dal-fiyat algoritması 30 örneğin 23 ünde en iyi çözümü bulmayı başarmıştır. 5. ve 6. problem büyüklüklerinde ise dal-fiyat algoritmasının ortalama ve en çok çözüm süresi değerleri CPLEX’e göre çok daha iyidir. Küçük örneklerde ise CPLEX daha iyi performans göstermiştir. Ancak, bu örnekler için çözüm süreleri çok kısa olduğu için önemli bir fark oluşmamaktadır. En büyük problemde, dal-fiyat algoritması 4 kere en iyi çözümü bulmakla birlikte en iyi çözümü bulamadığı örnekler için bulunan en iyi üst sınır değeri CPLEX tarafından bulunan değere göre daha kötü olmaktadır. Bu karşılaştırma her iki yöntem için raporlanan “Ort. S.” sütununda görülebilir.

Dal-fiyat algoritmasının açtığı düğüm sayısı incelendiğinde ise bazı problemlerde çok küçük değerler ile karşılaşılmaktadır. Sonuçlar detaylı incelendiğinde, 90 örnekten 36 tanesinin en iyi çözümünün kök düğümde bulunduğu tespit edilmiştir (%40). Bununda yanında örneklerin %69 ‘u 10 düğümden, %81’i 100 düğümden ve %89’u da 1000 düğümden az sayıda düğüm açılarak çözülebilmiştir.

4.6.2 Geliştirilmiş ENÇDF Algoritması

Bu bölümde Geliştirilmiş Eczane Nöbet Çizelgeleme Problemi İçin Dal-Fiyat Algoritması (GENÇDFA) ile yaptığımız deney sonuçlarını ENÇDFA ile karşılaştırmalı olarak raporladık. Önceki bölümde raporlanan sonuçlarda kök düğümden sonra bulunan kolonlar ile karar değişkenlerine tamsayı olma kısıtı konularak AP çözülmüş ve bulunan çözüm üst sınır olarak kullanılmıştır. AP'yi tamsayı problem olarak çözmek küçük örneklerde kolay olmak ile beraber en büyük örnek için uzun süre almaktadır. Dolayısı ile bu bölümde raporlanan sonuçlarda en büyük problem için kök düğümde tamsayı kısıtlı AP çözülmemiştir. Her düğümde ÜSBA algoritması çalıştırılmış, KTA'da ise FAPMS algoritması kullanılmıştır.

Tablo 20. GENÇDFA deney sonuçları

Test Örneği				**GENÇDFA**							**İyi**
				Çözüm Özellikleri			**Süre**				
I	**J**	**T**	**K**	**Opt**	**Ort.S.**	**Ort. DS**	**EnAz**	**Ort.**	**St. S.**	**EnÇok**	
20	10	5	4	10	-	0.00	0.20	0.44	0.14	0.67	3
40	20	5	4	10	-	0.00	3.57	5.07	1.45	7.99	8
60	30	5	9	10	-	1.80	6.86	11.92	2.79	15.18	8
20	20	10	4	10	-	0.40	1.34	3.28	1.30	5.12	8
20	30	15	9	10	-	59.80	5.62	14.86	11.45	37.94	7
40	40	10	9	10	-	453.60	27.11	103.39	192.31	649.87	6
40	60	15	9	9	0.18	5622.70	86.81	1239.73	2168.41	6662.74	6
60	60	10	9	10	-	211.00	139.45	242.44	252.41	959.66	2
60	90	15	9	7	4.45	4065.00	508.27	993.11	317.69	1555.21	6

Tablo 20'de "İyi" sütunu GENÇDFA'nın kaç örnekte ENÇDFA'dan daha iyi sonuç verdiğini göstermektedir. Buradaki iyi tanımı en iyi çözümün bulanabildiği örnekler için çözüm süresinin daha az, bulunamayan örnekler için ise elde edilen en iyi üst sınır değerinin daha küçük olmasıdır. GENÇDFA ile 54 örnekte ENÇDFA'nın sonuçlarına göre iyileşme sağlanmıştır. İyileşme daha çok büyük problemlerde olmuştur. Özellikle en büyük problemde 3 örneğin daha en iyi çözümü bulunabilmiş, çözüm süreleri iyileşme göstermiştir. En iyi çözümü bulunamayan örneklerde de daha iyi üst sınır değerleri bulunabilmiştir.

4.7 Değişken Komşuluk Arama Algoritmaları

Bu bölümde Eczane Nöbet Çizelgeleme problemi için geliştirdiğimiz Değişken Komşuluk Arama (DKA) algoritmaları olan Temel DKA (TDKA), Değişken Komşuluk Ayrıştırma Arama (DKAA) ve Değişken Komşuluk Kısıtlama Arama (DKKA) algoritmalarının performanslarını raporladık.

Tüm algoritmaları C dilinde kodladık ve deneyleri 4 GB bellekli Amd Phenom II X4 955 3.2 GHz işlemcili bir bilgisayarda gerçekleştirdik. Öncelikle rassal örnekler ile ilgili test sonuçlarını daha sonra ise İzmir iline ait gerçek veriler ile yaptığımız deneylerin sonuçlarını raporladık.

4.7.1 Rassal Örnek Testleri

TDKA, DKAA ve DKKA algoritmaları için en iyi değerlerini bulmak için k_{max} parametresi ve durdurma koşulları için başlangıç deneyleri yapılmıştır. k_{max} parametresi için üç seviye belirlenmiştir: 4, $[K/2]$ ve $\min(0,\ 9)$. Başlangıç deneyleri TDKA, DKAA ve DKKA algoritmaları için k_{max} parametresini $[K/2]$ olarak önermiştir.

Durdurma koşulları için dört seviye belirlenmiştir: iki seviye en çok genel tekrar sayısı için ($25{,}000$ ve $100 * T$), iki seviye en çok dış tekrar sayısı için (750 ve $10 * T$). Başlangıç deneyleri TDKA, DKAA ve DKKA algoritmaları için durdurma koşulunu $10 * T$ olarak önermiştir. Ayrıca en çok CPU zamanı koşulu $10 * T$ koşulu ile aynı anda kullanılmış ve $2{,}5$ saat olarak belirlenmiştir.

Ağlamaz ve Özpeynirci (2011), ENÇP için bir alt sınır geliştirmişlerdir. Bu alt sınır her müşteriyi ayrı ayrı ele alır ve her birinin bir çizelgede yürümesi gereken en az mümkün mesafeyi hesaplar. Verilen bir i müşterisi için, alt sınır en yakın olan eczaneyi bulur, j eczanesi olsun, bu müşteriyi bu eczaneye n_j gün atar ve daha sonra en yakın olan ikinci eczaneye geçer ve bunu planlama ufku sonuna kadar tekrar eder.

TDKA, DKAA ve DKKA algoritmaları her problem örneği için birer kez çalıştırılmıştır. Algoritmaların performansı iki kritere göre ölçülür: üst sınır ve alt sınır arasındaki aralık ve CPU zamanı. Aralık şu şekilde tanımlanır, $Aralık = \frac{ÜS - AS}{AS}$, burada $ÜS$ incelenen çözümün amaç fonksiyon değeri ve AS, Ağlamaz ve Özpeynirci (2011) tarafından tanımlanan alt sınır algoritmasıyla hesaplanan alt sınırdır. Aralıklar aynı zamanda küçük boyutlu problemlerin bir alt kümesinin en iyi çözümlerini AS alarakta incelenmiştir.

Mevcut sistemde, her bölge kendi çizelgesini diğer bölgelerden bağımsız olarak ve hiçbir bilgi alışverişi yapmadan hazırlamaktadır. Rastgele bir yöntemle mevcut sisteme iyi bir benzetim yapabileceğimize inanıyoruz. Bundan dolayı, temel bir performans ölçümü elde etmek amacıyla her problem örneği için rastgele 10 farklı çizelge oluşturduk. Bir çözüm elde edene kadar, her gün ve bölge için rastgele uygun bir eczaneye nöbet atadık ve n_j değerini güncelledik. Bu rastgele çözümlerin ortalama aralıklarını raporladık. Bu bilgiyle, önerilen metotların mevcut sisteme sağladığı katkıları değerlendirebiliriz. İki gerçek hayat problemi için rastgele yöntemle mevcut sistemin performanslarının birbirine çok yakın olmasını gözlemlemek ilginçtir (Bakınız Tablo 25).

TDKA, DKAA ve DKKA algoritmaları ile rastgele yöntemin test sonuçları karşılaştırılmıştır. Önceki bölümlerde bahsedildiği üzere, adil bir karşılaştırma yapabilmek için TDKA, DKAA ve DKKA algoritmalarının hepsi aynı başlangıç çözümüyle başlar. Aralıkların ve CPU zamanlarının ortalamalarının farkları (difference of the mean) her algoritma çifti dikkate alınarak test edilmiştir. Her problem örneği için iki gözlem olduğundan ikili t-test kullanılmış ve her algoritma için ayrı ayrı düşünülmüştür. Rastgele yöntem için CPU zamanları çok küçük olduğu için bunların farkları test edilmemiştir. Büyük boyutlu problemler için bütün algoritmalar CPU zamanı kısıtına takıldığı için bunların farkı test edilmemiştir.

$A^i = \{TDKA, DKAA, DKKA, Rastgele\}, i = 1,2$ bir algoritmayı göstersin ve A_o^i ise A^i algoritmasının kıyaslanan kriter için (aralık yada CPU) o. gözlemin değeri olsun. A^1 ve A^2 algoritmalarını kıyaslamak için, burada A^1 kıyaslanan kriter için (aralık yada CPU) daha küçük ortalamaya sahip olan algoritmadır, aşağıdaki hipotez tasarlanmıştır:

$$H_0: \mu_D \leq 0$$
$$H_\alpha: \mu_D > 0$$

$1 - \alpha = 0{,}95$ kararlılığındadır, burada$D_o = A_o^1 - A_o^2$ ve D ise D_o değerlerinin ortalamasıdır. Küçük problemler için 90 ve büyük problemler için 15 gözlem vardır. Tablo 21'de her kriter, problem büyüklüğü ve algoritma çifti için ikili t-test sonuçları raporlanmıştır. Küçük boyutlu problemler için t-istatistik değeri $t_{0,05;89} = 1{,}66$ ve büyük boyutlu problemler için $t_{0,05;14} = 1{,}76$ 'dir. Test sonuçlarının mutlak değerleri karşılık gelen değerlerden daha büyüktür. Bundan dolayı H_0 reddedilmiş ve bütün test edilen farkların istatistiksel olarak anlamlı (significant) olduğu

sonucuna varılmıştır. Anlatımı basitleştirmek adına, bu bölümün kalan kısmında istatistiksel olarak anlamlı ifadesi yerine anlamlı ifadesi kullanılacaktır.

Tablo 21. t-İstatistik tablosu

<table>
<tr><th>Kriter</th><th>Problem Büyüklüğü</th><th>Gözlem Sayısı</th><th colspan="5">t-istatistiği</th></tr>
<tr><td rowspan="8">Aralık (%)</td><td rowspan="4">Küçük</td><td rowspan="4">90</td><td colspan="2"></td><td colspan="3">A^2</td></tr>
<tr><td colspan="2"></td><td>DKKA</td><td>DKAA</td><td>Rastgele</td></tr>
<tr><td rowspan="3">A^1</td><td>TDKA</td><td>-3,87</td><td>-7,67</td><td>-22,14</td></tr>
<tr><td>DKKA</td><td></td><td>-6,90</td><td>-22,19</td></tr>
<tr><td>DKAA</td><td></td><td></td><td>-23,32</td></tr>
<tr><td rowspan="4">Büyük</td><td rowspan="4">15</td><td colspan="2"></td><td colspan="3">A^2</td></tr>
<tr><td colspan="2"></td><td>TDKA</td><td>DKAA</td><td>Rastgele</td></tr>
<tr><td rowspan="3">A^1</td><td>DKKA</td><td>-2,47</td><td>-4,98</td><td>-22,25</td></tr>
<tr><td>TDKA</td><td></td><td>-8,73</td><td>-30,05</td></tr>
<tr><td>DKAA</td><td></td><td></td><td>-31,98</td></tr>
<tr><td rowspan="4">CPU (Sn)</td><td rowspan="4">Küçük</td><td rowspan="4">90</td><td colspan="2"></td><td colspan="2">A^2</td><td rowspan="4"></td></tr>
<tr><td colspan="2"></td><td>DKKA</td><td>TDKA</td></tr>
<tr><td rowspan="2">A^1</td><td>DKAA</td><td>-6,73</td><td>-6,26</td></tr>
<tr><td>DKKA</td><td></td><td>-6,09</td></tr>
</table>

Tablo 22'de küçük boyutlu problem için test sonuçları verilmiştir. İlk dört kolon probem büyüklüğünü göstermektedir; müşteri sayısı (I), eczane sayısı (J), planlama ufku (T) ve bölge sayısı (K). Beşinci kolon, her problem boyutu için rastgele oluşturulan 10 çizelgenin ortalama aralığını göstermektedir. Sonraki üç kolon, TDKA, DKAA ve DKKA algoritmarından elde edilen ortalama aralıkları (her problem boyutunda 10 örnek) göstermektedir. Geriye kalan kolonlar da ortalama CPU zamanlarını göstermektedir.

Rastgele yöntem için, ortalama aralık problem boyutu arttıkça artar ve bütün TDKA, DKAA ve DKKA algoritmaları rastgele yöntemden daha anlamlı yüksek kaliteli çözümler oluşturur. Problem parametreleri rastgele yöntemin çözüm kalitesini etkiler, planlama ufku uzunluğu arttıkça aralık artar. Rastgele yöntemin çözüm kalitesi çok kötü değildir. Biz bunun, bölgelerin dağılımından ve her bölgede her gün bir eczanenin nöbetçi olması kısıtından kaynaklandığına

inanıyoruz. Bu durum müşterilerin kendilerine yakın bir nöbetçi eczane bulma olasılığını arttırmaktadır.

Tablo 22. Küçük boyutlu problemler için test sonuçları

I	J	T	K	Aralık (%)				CPU (Sn)		
				Rastgele	TDKA	DKAA	DKKA	TDKA	DKAA	DKKA
20	10	5	4	3,2	0,5	0,5	0,5	0,0	0,0	0,0
40	20	5	4	4,6	0,6	0,6	0,6	0,1	0,0	0,0
20	20	10	4	5,5	0,4	0,6	0,4	0,2	0,0	0,1
60	30	5	9	3,6	0,3	0,4	0,3	0,7	0,0	0,2
20	30	15	9	5,0	0,2	0,3	0,2	5,2	0,2	1,7
40	40	10	9	5,8	0,5	0,7	0,6	5,4	0,2	1,7
60	60	10	9	7,9	0,7	1,2	0,8	13,8	0,9	4,4
40	60	15	9	7,8	0,7	1,4	0,7	29,9	2,7	9,5
60	90	15	9	8,9	0,7	1,7	0,8	62,9	5,8	18,2

Küçük boyutlu problemler için deney sonuçlarına göre algoritmaların performansları tartışılmıştır. Çözüm kalitesi açısından, TDKA anlamlı şekilde DKKA ‘dan daha iyi, DKKA ise anlamlı şekilde DKAA ‘dan daha iyidir. CPU zamanı açısından, tam tersi sıra gözlemlenmiştir, DKAA anlamlı şekilde DKKA ‘yı geçer ve DKKA anlamlı şekilde TDKA ‘yı geçer.

Tablo 23’de TDKA, DKAA ve DKKA algoritmaları için detaylı test sonuçları raporlanmıştır. Çözüm kalitesi ve CPU zamanının minimum, ortalama, maksimum ve standart sapma değerleri raporlanmıştır. Aralıklar 0% ve $2{,}4\%$ arasında değişmektedir. Hiçbir keskin eğilim gözlenmemiştir. Buna rağmen, problem boyutu arttıkça minimum ve ortalama aralık değerleri artma eğilimi gösterirken, maksimum ve standart sapma değerleri azalma eğilimi göstermektedir. TDKA aralık değerleri küçük eşittir karşılık gelen DKKA aralık değerlerinden, ve DKKA aralık değerleri küçük eşittir karşılık gelen DKAA aralık değerlerinden. TDKA CPU zamanı değerleri büyük eşittir karşılık gelen DKKA CPU zamanı değerlerinden, ve DKKA CPU zamanı değerleri büyük eşittir karşılık gelen DKAA CPU zamanı değerlerinden. Problem boyutu arttıkça CPU zamanı değerleri hızlıca artar. Varyans katsayısı (coefficient of variation) değerlerini dikkate alırsak, çözüm kalitesi ve CPU zamanı ölçülerine göre algoritmalar birbirlerine baskın değildir.

Bir önceki bölümde bir saat CPU zamanı kısıtında sadece dört problem boyutunda her 10 problem örneğinin en iyi çözümünün bulunabildiğini raporlamıştık. Bu problem boyutları $I20\,J10\,T5\,K4, I40\,J20\,T5\,K4, I60\,J30\,T5\,K9$ ve $I20\,J30\,T15\,K9$ 'dur. TDKA için, karşılık gelen aralıklar en iyi çözümler alt sınır alınarak hesaplanmıştır. Aralıklar karşılıklı olarak $0{,}00\%, 0{,}01\%, 0{,}05\%$ ve $0{,}02\%$ 'dir. En yüksek aralık $0{,}19\%$ 'dir. Tablo 22'de bu aralıklar $0{,}5\%, 0{,}6\%, 0{,}3\%$ ve $0{,}2\%$ olarak raporlanmıştır. En iyi çözümleri alt sınır olarak kullanmak sezgisel algoritmaların önerdiği çözümlerin algılanan kalitesini iyileştirmiştir. Buna rağmen, orta ve büyük boyutlu problemlerin en iyi çözümlerini belirlemek pratik değildir. Algılanan çözüm kalitesi alt sınır kalitesine bağlıdır ve daha güçlü bir alt sınır geliştirmek gelecek araştırma konusudur.

Tablo 23. Küçük boyutlu problemlerde TDKA, DKAA ve DKKA için detaylı test sonuçları

I	J	T	K	Sezgisel Türü	Aralık (%)				CPU (Sn)			
					Min	Ortalama	Max	Std. Sapma	Min	Ortalama	Max	Std. Sapma
20	10	5	4	TDKA	0,0	0,5	1,6	0,6	0,0	0,0	0,0	0,0
				DKAA	0,0	0,5	1,6	0,5	0,0	0,0	0,0	0,0
				DKKA	0,0	0,5	1,6	0,6	0,0	0,0	0,0	0,0
40	20	5	4	TDKA	0,2	0,6	1,5	0,4	0,0	0,1	0,1	0,0
				DKAA	0,2	0,6	1,5	0,4	0,0	0,0	0,0	0,0
				DKKA	0,2	0,6	1,4	0,4	0,0	0,0	0,0	0,0
20	20	10	4	TDKA	0,0	0,4	1,0	0,4	0,1	0,2	0,3	0,1
				DKAA	0,0	0,6	1,7	0,5	0,0	0,0	0,1	0,0
				DKKA	0,0	0,4	1,1	0,4	0,0	0,1	0,1	0,0
60	30	5	9	TDKA	0,0	0,3	0,7	0,2	0,3	0,7	1,1	0,2
				DKAA	0,1	0,4	0,8	0,2	0,0	0,0	0,1	0,0
				DKKA	0,1	0,3	0,7	0,2	0,1	0,2	0,4	0,1
20	30	15	9	TDKA	0,1	0,2	0,5	0,2	1,3	5,2	9,2	2,3
				DKAA	0,1	0,3	0,7	0,2	0,0	0,2	0,7	0,2
				DKKA	0,0	0,2	0,5	0,2	0,4	1,7	3,2	0,8
40	40	10	9	TDKA	0,1	0,5	1,3	0,4	1,8	5,4	8,5	2,2
				DKAA	0,3	0,7	1,5	0,4	0,0	0,2	0,5	0,2
				DKKA	0,1	0,6	1,4	0,4	0,6	1,7	2,7	0,7
60	60	10	9	TDKA	0,5	0,7	1,0	0,2	12,6	13,8	15,6	1,1
				DKAA	0,8	1,2	1,7	0,3	0,5	0,9	1,1	0,2
				DKKA	0,5	0,8	1,1	0,2	3,5	4,4	5,7	0,6
40	60	15	9	TDKA	0,3	0,7	1,1	0,3	23,1	29,9	34,9	3,9
				DKAA	0,8	1,4	2,1	0,4	1,9	2,7	3,4	0,5
				DKKA	0,4	0,7	1,1	0,3	8,1	9,5	11,3	1,0
60	90	15	9	TDKA	0,3	0,7	1,1	0,2	57,1	62,9	74,7	4,9
				DKAA	0,8	1,7	2,4	0,5	4,6	5,8	6,5	0,7
				DKKA	0,4	0,8	1,4	0,3	16,6	18,2	20,6	1,2

Tablo 24. Büyük boyutlu problemler için test sonuçları

I	J	T	K	Aralık (%)				CPU (Sn)		
				Rastgele	TDKA	DKAA	DKKA	TDKA	DKAA	DKKA
350	250	25	49	9,1	1,0	1,6	1,2	9002,1	9000,1	9000,5
350	250	25	49	7,9	1,1	1,3	1,1	9000,5	9000,0	9000,1
350	250	25	49	8,2	1,1	1,6	1,2	9000,4	9000,0	9000,8
350	250	25	49	7,4	0,9	1,2	1,0	9000,0	9000,1	9000,5
350	250	25	49	8,3	1,0	1,3	1,2	9001,5	9000,1	9001,4
350	500	50	49	10,2	1,3	2,1	1	9007,0	9000,1	9003,2
350	500	50	49	10,6	1,2	2,0	1,1	9004,2	9001,0	9000,5
350	500	50	49	11,0	1,2	1,9	1,2	9006,4	9000,2	9002,1
350	500	50	49	10,6	1,0	1,6	1,0	9001,4	9001,0	9003,4
350	500	50	49	10,2	1,2	2,1	1,1	9002,8	9000,1	9000,5
350	1000	100	49	11,4	2,6	3,4	0,9	9001,6	9000,3	9000,8
350	1000	100	49	11,6	2,4	3,4	0,9	9001,4	9000,8	9004,2
350	1000	100	49	10,6	2,3	2,8	0,8	9004,3	9000,1	9000,7
350	1000	100	49	12,0	2,5	2,7	0,9	9001,7	9000,1	9000,3
350	1000	100	49	11,5	2,2	2,9	0,9	9003,6	9000,2	9000,7

Büyük boyutlu problemler için test sonuçları Tablo 24'de verilmiştir. Bütün TDKA, DKAA ve DKKA algoritmaları rastgele yöntemden anlamlı şekilde daha iyidir. Çözüm kalitesi yönünden, DKKA anlam şekilde TDKA 'dan daha iyi ve TDKA anlamlı şekilde DKAA 'dan daha iyidir. Buna rağmen, CPU zamanları bütün algoritmalar 2,5 saat zaman kısıtından sonra durdukları için karşılaştırılmamıştır.

4.7.2 İzmir Uygulaması

Önerilen DKA algoritmaları İzmir ilindeki Eczane Nöbet Çizelgeleme Problemi (ENÇP) üzerinde test edilmiştir. Bu bölümde, İzmir uygulaması ilgili bilgiler verilmektedir.

Veri kümesinin ait olduğu dönem, eczane sayısı ve planlama ufkunun uzunluğu Tablo 4'de verilmiştir. İlk dönemde 1046 eczane ve 97 gün, ikinci dönemde 1053 eczane ve 101 gün vardır. Her iki dönemde 350 müşteri ve 45 bölge yer alır.

TDKA, DKAA ve DKKA algoritmaları bu iki problemi çözmek için $2{,}5$ saat süre kısıtı altında çalıştırılmıştır. Sonuçlar Tablo 25'de sunulmuştur. Bu sonuçlara göre mevcut çözümler alt sınırdan yaklaşık %6 uzaklıktadır. Aynı zamanda, her iki problem için 10 rastgele çizelge oluşturulmuş ve bulunan çözümlerin ortalama maliyetleri mevcut çözümünlerin maliyetlerine çok yakın çıkmıştır. DKKA algoritması her iki problem için de TDKA ve DKAA algoritmalarından daha iyi sonuçlar önermiş ve aralığı $0{,}5\%$ seviyesine düşürmüştür.

Tablo 25. DKA ile İzmir uygulama sonuçların karşılaştırılması

Planlama Periyodu	**Karşılaştırma Türü**	**Rastgele**	**Mevcut**	**TDKA**	**DKKA**	**DKAA**	**Alt Sınır**
2010 /3	**Maliyet (Milyon km)**	309,27	309,01	294,76	293,46	295,34	291,89
	Aralık (%)	5,96	5,87	0,98	0,54	1,18	-
2011/1	**Maliyet (Milyon km)**	321,41	321,12	306,28	304,82	307,55	303,28
	Aralık (%)	5,98	5,88	0,99	0,51	1,41	-

4.8 İki Amaçlı ENÇ Problemi Testleri

Bu bölümde İki Amaçlı Eczane Nöbet Çizelgeleme (İAENÇ) problemi için geliştirdiğimiz kesin ve sezgisel yöntemler ile yaptığımız deneylerin sonuçlarını raporladık.

4.8.1 İki Amaçlı Kesin Çözüm Sonuçları

İAENÇ probleminin tüm etkin sonuçlarını bulmak için İAENÇ(ε) modelini kullanan ve ε değerini sistematik biçimde güncelleyen Algoritma 25'i kullandık. Matematiksel modeli ve algoritmayı GAMS ile kodladık. İş yükü alt ve üst sınırlarını sezgisel yöntem ile belirledik. Deneyleri 4 GB bellekli Intel Xeon 2.53 GHz X2 işlemcili Windows Server 2008 işletim sistemine sahip bir bilgisayarda IBM ILOG CPLEX 12.4 kullanan GAMS 23.9 ile gerçekleştirdik.

Deneylerde her İAENÇ(ε) modelinin çözümü için 3600 saniyelik süre limiti tanımladık. Ayrıca Algoritma 25'in toplam çalışma süresi için de dolaylı bir limit tanımladık. Eğer geçen toplam süre 3600 saniyeden fazla ise algoritma bir sonraki İAENÇ(ε) modelini çalıştırmadı.

Her problem büyüklüğü için 10 rassal örneği çözdük. Deney sonuçlarını Tablo 26'da raporladık. Tabloda ilk dört sütun problem büyüklüğünü gösteriyor. Sonraki üç sütun ise algoritmanın bulduğu en az, ortalama ve en çok etkin sonuç sayısını raporluyor. Son üç sütun ise algoritmanın toplam süresinin en az, ortalama ve en fazla değerlerini sunuyor. En yüksek etkin sonuç ortalaması 4,2 en yüksek değer ise 10. Etkin sonuç sayıları bölge ve gün sayısının az, eczane sayısının daha fazla olduğu problem büyüklükleri için daha yüksek denilebilir.

Etkin sonuçlarının sayısının fazla olmamasının sebebinin ikinci amaç fonksiyonumuzun yapısı olduğunu düşünüyorum. Bu amaç fonksiyonu toplam değil darboğaz cinsi bir yapıda olduğu için toplam iş yükü oranını değil, en düşük iş yüküne sahip eczanenin oranını dikkate alıyor. Toplam çözüm süresinin en yüksek değerinin raporladığımız son sütunda 3,600 saniyeden daha yüksek süreler görmemizin sebebi, algoritmanın son modeli çağırdığında henüz toplamda 3,600 saniyeden az çalışmış olmasıdır. Son modelin bitmesini takiben süre limiti kontrol edilir ve algoritma durur.

Tablo 26. İAENÇ probleminin tüm etkin sonuçları bulma

I	J	T	K	Etkin Sonuç Sayısı			Toplam Çözüm Süresi (saniye)		
				En az	Ort.	En çok.	En az	Ort.	En çok
20	10	5	4	1	2,0	5	0,1	0,7	2,4
20	20	10	4	1	1,4	3	12,9	3.806,9	6.736,4
20	30	15	9	1*	1,2	3	3.600,4	3.728,5	4.722,1
40	20	5	4	1	4,2	10	17,9	1.178,7	5.587,3
40	40	10	9	1*	1,0	1	2.832,5	3.639,8	4.680,2
40	60	15	9	1*	1,0	1	3.603,0	4.299,1	6.791,7
60	30	5	9	1	2,1	6	2,2	2.113,4	6.310,1
60	30	5	9	1*	1,0	1	3.602,8	3.603,6	3.605,7
60	90	15	9	-	-	-	-	-	-

*Süre limiti sonunda bulunan sonucun tek sonuç var ve bu sonucun en iyi olduğu ispatlanamadı.

4.8.2 İki Amaçlı DKA Sonuçları

İDKA algoritması C dilinde kodlanmış ve 4 GB bellekli Amd Phenom II X4 955 3.2 GHz işlemcili bilgisayarda test edilmiştir.

İDKA algoritması, küçük boyutlu, büyük boyutlu ve gerçek yaşam problemlerinde test edilmiştir. Küçük boyutlu problemlerde, 9 farklı problem boyutu ve her boyutta 10 farklı problem olmak üzere toplam 90 adet problem bulunmaktadır. Büyük boyutlu problemlerde ise 3 farklı problem boyutu ver her boyutta 5 farklı problem olmak üzere toplam 15 adet problem bulunmaktadır. Gerçek yaşamda ise 2 adet problem vardır. Her bir küçük boyutlu problem için 10 farklı rassal başlangıç değeri (seed) kullanılarak İDKA algoritması çalıştırılmış ve her bir başlangıç değeri için elde edilen etkin çözümlerin birleşimi alınmış ve bu kümede etkin olmayan çözümler elenmiştir. Elde edilen bu birleşim kümesi, her bir problemin ilk başlanıç değeri ile elde edilen çözümlerinin karşılaştırılması ve performansının ölçülmesi için kullanılmıştır. Aynı birleşim alma yöntemi, büyük boyutlu ve gerçek yaşam problemlerinde 5 farklı başlangıç değeri için İDKA algoritması çalıştırılarak elde edilmiştir.

İDKA algoritmasının performansını ölçmek amacıyla iki farklı ölçüm kullanılmıştır: (i) İsabet (hit) Oranı, (ii) Doğruluk (accuracy) Oranı (Zitzler v.d., 2003). Her iki ölçümde de bulunan etkin çözüm sayısının oranını hesaplamaktadır. Ancak, iki ölçüm farklı paydalar kullanmaktadır. İsabet oranı, paydada ilk başlangıç değeri alındığında yaklaşılan Pareto eğrisindeki etkin çözüm sayısını alırken doğruluk oranı, birleşim kümesindeki etkin çözüm sayısını alır. Bundan dolayı, her iki ölçüm ne kadar yüksek değer verirse o kadar iyi sonuç alınmış anlamına gelmektedir. Bu iki ölçümüm tanımları aşağıda verilmiştir:

$$\text{İ}sabet\ Oran\text{ı} = \frac{\sum_{i=1}^{|Y_{seed1}|} e_i}{|Y_{seed1}|}$$

$$Doğruluk\ Oran\text{ı} = \frac{\sum_{i=1}^{|Y_{seed1}|} e_i}{|Y_{birleşim}|}$$

burada $e_i = \begin{cases} 1, & eğer\ Y_{seed1}\ içindeki\ bir\ nokta\ Y_{birleşim}\ içinde\ varsa \\ 0, & aksi\ takdirde. \end{cases}$.

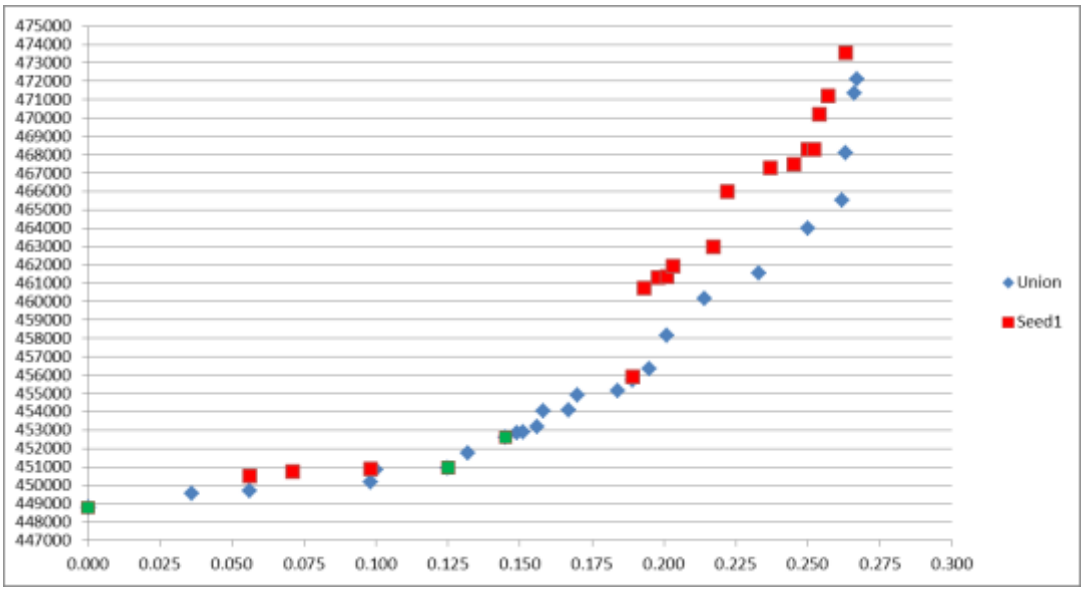
$Y_{birleşim}$ birleşim etkin sonuç kümesini ve Y_{seed1} başlangıç değeri 1 için elde edilen etkin sonuç kümesini göstermektedir.

Tablo 27'de küçük boyutlu problemlerde her problem boyutu için sırasıyla başlangıç çözüm değeri 1 olduğunda İDKA algoritmasıyla elde edilen ortalama etkin çözüm sayısı, birleşim kümedeki ortalama etkin çözüm sayısı, ortalama CPU zamanı, ortalama isabet oranı değeri ve ortalama doğruluk oranı değeri verilmektedir. Dikkat edilecek olursa, $|Y_{birleşim}| \geq |Y_{seed1}|$ olduğundan isabet oranı değeri her zaman doğruluk oranı değerinden büyüktür. Problem boyutu arttıkça isabet oranı ve doğruluk oranı değerleri genellikle azalmaktadır.

Şekil 6'da I40 J40 T10 K9 R8 problem örneği için birleşim etkin sonuç kümesi ve başlangıç değeri 1 olduğunda elde edilen etkin sonuç kümesi gösterilmiştir. İlk başlangıç değeri ile bulunan (ancak birleşim kümede yer almayan) etkin sonuçlar kırmızı, birleşim kümede yer alan (ancak ilk başlangıç değeri ile bulunamayan etkin sonuçlar mavi, her iki kümede de yer alan etkin sonuçlar ise yeşil ile gösterilmiştir.

Tablo 27. Küçük boyutlu problemler için İDKA test sonuçları

I	**J**	**T**	**K**	**Ortalama Etkin Çözüm Sayısı (seed1)**	**Ortalama Etkin Çözüm Sayısı (Birleşim)**	**CPU (Sn)**	**Ortalama İsabet Oranı**	**Ortalama Doğruluk Oranı**
20	10	5	4	2,2	2,3	0,0	1,0	0,9
40	20	5	4	5,6	6,3	0,1	0,9	0,8
20	20	10	4	10,9	13,2	0,3	0,6	0,5
60	30	5	9	3,2	4,6	0,2	0,3	0,3
20	30	15	9	6,6	9,6	2,7	0,7	0,6
40	40	10	9	11,3	14,0	1,3	0,3	0,3
60	60	10	9	6,9	14,6	2,5	0,3	0,2
40	60	15	9	8,5	11,9	7,3	0,1	0,0
60	90	15	9	6,9	11,8	12,1	0,2	0,1



Şekil 6. I40 J40 T10 K9 R8 problem örneği için etkin sonuçlar kümesi

Tablo 28. Büyük boyutlu problemler için İDKA test sonuçları

I	J	T	K	Ortalama Etkin Çözüm Sayısı (seed1)	Ortalama Etkin Çözüm Sayısı (Birleşim)	CPU (Sn)	Ortalama İsabet Oranı	Ortalama Doğruluk Oranı
350	250	25	49	20	20	5401,2	1,0	1,0
350	250	25	49	20	29	5402,1	0,6	0,4
350	250	25	49	1	1	5400,1	0,0	0,0
350	250	25	49	1	1	5408,4	0,0	0,0
350	250	25	49	19	19	5400,7	0,1	0,1
350	500	50	49	1	13	5410,9	0,0	0,0
350	500	50	49	1	1	5406,2	1,0	1,0
350	500	50	49	9	13	5409,0	0,3	0,2
350	500	50	49	1	1	5419,2	1,0	1,0
350	500	50	49	6	6	5401,0	1,0	1,0
350	1000	100	49	1	1	5500,9	0,0	0,0
350	1000	100	49	1	1	5443,5	0,0	0,0
350	1000	100	49	1	1	5443,2	0,0	0,0
350	1000	100	49	1	1	5618,4	1,0	1,0
350	1000	100	49	1	1	5430,4	1,0	1,0

Tablo 28 büyük boyutlu problemler için test sonuçlarını göstermektedir. Bütün problemler, 1,5 saat zaman sınırına takılmıştır ve genellikle birleşim etkin sonuç kümesinde bir çözüm bulunmaktadır.

İki gerçek hayat örneği için de büyük boyutlu problemlerde izlediğimiz metodu uyguladık. Deney sonuçları Tablo 29'da raporladık. Her iki problem örnek için de ilk başlangıç değeri ile tek sonuç elde ettik. Ayrıca birleşim kümesinde de tek etkin sonuç yer alıyor. İlk örnek için başlangıç değeri 1 ile bulunan sonuç birleşim kümesinde yer alırken, ikinci örnekte diğer başlangıç değerlerinden birisinin bulduğu sonuç yer almakta. Bu sebeple isabet ve doğruluk oranı ilk örnek için 1, ikincisi için 0 değerini alıyor.

Tablo 29. İDKA ile İzmir uygulaması test sonuçları

Planlama Periyodu	**Etkin Çözüm Sayısı (seed1)**	**Etkin Çözüm Sayısı (Birleşim)**	**CPU (Sn)**	**İsabet Oranı**	**Doğruluk Oranı**
2010/3	1	1	5444,6	1,0	1,0
2011/1	1	1	5411,3	0,0	0,0

5. SONUÇ

Bu projede bir gerçek hayat problemi olan eczane nöbet çizelgeleme problemleri üzerinde çalıştık. İzmir Eczacılar Odası ile görüşerek mevcut sistemi analiz ettik. Problemin basit ve gerçekçi türevlerini tanımladık, matematiksel modellerini oluşturduk ve hesaplama karmaşıklıkları üzerinde çalıştık.

Eczacılar Odası'ndan aldığımız geçmiş dönemlere ait nöbet çizelgeleri, İzmir Büyükşehir Belediyesi Coğrafi Bilgi Sistemi ve TÜİK'den satın aldığımız mahalle nüfus bilgilerini birleştirerek iki gerçek hayat örneği hazırladık. Ayrıca proje boyunca kullanmak üzere küçük ve büyük boyutlu rassal örnekler oluşturduk. Bu örneklerin gerçek hayat problemi ile benzer özellikleri taşımasına özen gösterdik.

Genel amaçlı çözücü küçük boyutlu problemlerin sadece bir bölümünde en iyi sonucu bulabildi. Bu sebeple, probleme özgü kesin çözüm yöntemi olarak dal-sınır ve dal-fiyat algoritmaları önerdik ve bu algoritmaların çeşitli türevlerini oluşturduk. Özellikle dal-fiyat algoritmasının gelişmiş türevi ile genel çözücüden daha iyi sonuçlar elde ettik.

Ancak kesin çözüm yöntemlerinin büyük boyutlu ve gerçek hayat problemlerini makul sürelerde çözememesi sebebi ile sezgisel algoritmalar geliştirdik. Ayrıca probleme özgü alt sınır algoritmaları önerdik. Tabu arama ve değişken komşuluk arama yöntemlerinin çeşitli türevlerini oluşturduk ve tüm problemlere olurlu çözümler bulduk. Özellikle değişken komşuluk arama algoritmasının en iyi türevi ile alt sınıra çok yakın sonuçlar elde ettik.

Projenin son aşamasında, problemi iki amaçlı olarak inceledik. Bu problemin tüm etkin sonuçlarını bulmak üzere matematiksel model ve sezgisel yöntemler önerdik.

Bu projenin devamında,

- sonuçların gerçek hayata uygulanması,
- dal-fiyat algoritmasının daha büyük boyutlu problemleri çözmek üzere geliştirilmesi
- eczane nöbet çizelgeleme ve eczane bölgelerinin güncellenmesi problemlerinin bir arada çözülmesi

- müşteri taleplerinin daha iyi tahmin edilmesi (örneğin hastane gibi özel talep içeren bölgelerin ve mahallelerdeki yaş aralıklarının dikkate alınması) ve
- diğer illerde benzer çalışmaların yapılması

konuları üzerinde yeni araştırma olanakları mevcuttur.

KAYNAKLAR

Abedzadeh, M., Mazinani, M., Moradinasab, N. ve Roghanian, E. 2013. “Parallel variable neighborhood search for solving fuzzy multi-objective dynamic facility layout problem”, Int. J. Adv. Manuf. Technology, 65, 197–211.

Adenso-Diaz, B. ve Rodriguez, F. 1997. “A Simple Search Heuristic for the MCLP: Application to the Location of Ambulance Bases in a Rural Region”, Omega, 25, 181-187.

Adibi, M. A., Zandieh, M. ve Amiri, M. 2010. “Multi-objective scheduling of dynamic job shop using variable neighborhood search”, Expert Systems with Applications, 37, 282–287.

Ağlamaz, E. 2011. “Pharmacy Duty Scheduling Problem with an Application to İzmir”. Yüksek Lisans Tezi, İzmir Ekonomi Üniversitesi, Lojistik Yönetimi Bölümü.

Ağlamaz, E. Özpeynirci, Ö. 2011. “Pharmacy Duty Scheduling Problems”. Teknik Rapor, İzmir Ekonomi Üniversitesi, Lojistik Yönetimi Bölümü.

Alguwaizani, A., Hansen, P., Mladenovic, N. ve Ngai, E. 2011. “Variable neighborhood search for harmonic means clustering”, Applied Mathematical Modelling, 35, 2688-2694.

Alp, O., Erkut, E. ve Drezner, D. 2003. “An efficient genetic algorithm for the p -median problem”, Annals of Operations Research, 122, 2142.

Al-Sultan, K. S., Al-Fawzan, M. A. 1999. “A Tabu Search Approach to the Uncapacitated Facility Location Problem”, Annals of Operations Research, 86 91-103.

Arroyo, J. E. C., Ottoni, R. S. ve Oliveira, A. P. 2011. “Multi-objective Variable Neighborhood Search Algorithms for a Single Machine Scheduling Problem with Distinct due Windows, Electronic Notes in Theoretical Computer Science”, 281, 5–19.

Başar, A., Çatay, B., Ünlüyurt, T. 2011. “A Multi-period Double Coverage Approach for Locating the Emergency Medical Service Stations in Istanbul”, Journal of the Operational Research Society 62, 627-637.

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H. 1998. “Branch-and-Price: Column Generation for Solving Huge Integer Programs”, Operations Research, 46, 3.

Beasley, J.E. 1985. “A Note on Solving Large p-Median Problems”, European Journal of Operational Research, 21, 270-273.

Bilgin, S., Azizoğlu, M. 2009. “Operation assignment and capacity allocation problem in automated manufacturing systems”, Computers & Industrial Engineering, 56 662–676.

Brimberg, J., Urosevic, D. ve Mladenovic, N. 2006. “Variable Neighborhood Search for the Vertex Weighted K-Cardinality Tree Problem”, European Journal of Operational Research, 171, 74-84.

Carreas, M., Serra, D. 1999. “On Optimal Location with Threshold Requirements”, Socio-Economic Planning Sciences, 32(2), 91-103.

Çatay, B., Başar, A., Ünlüyurt, T. 2008. “İstanbul’da Acil Yardım İstasyonlarının Yerlerinin Planlanması”, Endüstri Mühendisliği Dergisi, 19(4), 20-35.

Crainic, T. G., Gendreau, M., Hansen, P., Mladenovic, N. 2004. “Cooperative Parallel Variable Neighborhood Search for the p-Median”, Journal of Heuristics, 10 293–314.

Dehnokhalaji, A., Korhonen, P. J., Köksalan, M., Nasrabadi, N. ve Wallenius, J. 2011. “Convex cone-based partial order for multiple criteria alternatives”, Decision Support Systems, 51, 256–261.

Doerner, K.F., Gutjahr, W.J., Hartl, R.F., Karall, M., Reimann, M. 2005. “Heuristic Solution of an Extended Double-Coverage Ambulance Location Problem for Austria”, Central European Journal of Operations Research, 13, 325-340.

Dupont, L. 2008. “Branch and bound algorithm for a facility location problem with concave site dependent costs”, Int. J. Production Economics 112, 245–254.

Eskandarpour, M., Zegordi, S. H. ve Nikbakhsh, E. 2013. “A parallel variable neighborhood search for the multi-objective sustainable post-sales network design problem”, International Journal of Production Economics, 145, 117–131.

Eskandarpour, M., Nikbakhsh, E. ve Zegordi, S. H. 2013a. “Variable neighborhood search for the bi-objective post-sales network design problem: A fitness landscape analysis approach”, Computers and Operations Research, http://dx.doi.org/10.1016/j.cor.2013.06.002

Fayed, H.A. ve Atiya, A.F. 2013. “A mixed breadth-depth first strategy for the branch and bound tree of Euclidean k-center problems”, Computational Optimization and Applications, 54, 675-703, (2013).

Fleszar, K. ve Hindi, K. S. 2008. “An effective VNS for the capacitated p–median problem”, European Journal of Operational Research, 191, 612-622.

Gagne, C., Gravel, M. ve Price, W. L. 2005. “Using metaheuristic compromise programming for the solution of multiple-objective scheduling problems”, Journal of the Operational Research Society, 56, 687-698.

Garcia-Lopez, F., Melian-Batista, B., Moreno-Perez, J. A. ve Moreno-Vega, M. 2002. “The parallel variable neighborhod search for the p-median problem”, Journal of Heuristics, 8, 375-388.

Gendreau, M., Laporte, G., Semet, F. 1997. “Solving an Ambulance Location Model by Tabu Search, Location Science”, 5, 75-88.

Gendreau,M., Laporte, G., Semet, F. 2001. “A Dynamic Model and Parallel Tabu Search Heuristic for Real Time Ambulance Relocation”, Parallel Computing, 27, 1641-1653.

Glover, F. 1989. “Tabu Search–Part I”, ORSA Journal on Computing, 1(3) 190-206.

Glover, F. 1990. “Tabu Search–Part II”, ORSA Journal on Computing, 2(1) 4-32.

Glover, F., Laguna, M. 1997. “Tabu Search”, Norwell, MA: Kluwer Academic Publishers.

Hale, T.S., Moberg, C.R. 2003. “Location Science Research: A Review”, Annals of Operations Research, 123, 21-35.

Hanafi, S., Lazic, Mladenovic, N., Wilbaut, C., Crevits, I. 2010. “Hybrid Variable Neighborhood Decomposition Search for 0-1 Mixed Integer Programming Problem”, Electronic Notes in Discrete Mathematics, 36, 883–890.

Hansen, P. ve Jaumard, B. 1997. “Cluster analysis and mathematical programming. Mathematical Programming”, 79, 191-215.

Hansen, P., Mladenovıc, N. 1997. “Variable Neighborhood Search for the p-Median”, Location Science, 5, 4, 207-226.

Hansen, P. ve Mladenovic, N. 2001a. “Variable Neighborhood Decomposition Search”, Journal of Heuristics, 7, 335-350.

Hansen, P. ve Mladenovic, N. 2001b. “Variable Neighborhood Search: Principles and Applications”, European Journal of Operational Research, 130, 449-467.

Hansen, P., Mladenovic, N. ve Urosevic, D. 2006. “Variable neighborhood search and local branching”, Computers and Operations Research, 33, 3034-3045.

Hansen, P., Oğuz, C. ve Mladenovic, N. 2008. “Variable neighborhood search for minimum cost berth allocation”, European Journal of Operational Research, 191, 636-649.

Harewood, S.I. 2002. “Emergency Ambulance Deployment in Barbados: A Multi-Objective Approach”, Journal of the Operational Research Society, 53, 185-192.

Holmberg, K., Rönnqvist, M. ve Yuan D. 1999. “An exact algorithm for the capacitated facility location problems with single sourcing”, European Journal of Operational Research 113, 544 – 559.

Jarboui, B., Derbel, H., Hanafi, S. ve Mladenovic, N. 2013. “Variable neighborhood search for location routing”, Computers and Operations Research, 40, 47-57.

Kariv, O. ve Hakimi, S.L. 1969. “An algorithmic approach to network location problems”, SIAM Journal on Applied Mathematics, 37, 539-560.

Karsu, Ö. ve Azizoğlu, M. 2012. “The multi-resource agent bottleneck generalised assignment problem”, International Journal of Production Research, 50, 2, 309-324.

Kochetov, Y., Alekseeva, E., Levanova, T. ve Loresh, N. 2005. “Large neighborhood search for the p-median problem”, Yugoslav Journal of Operations Research, 15 (1), 53-63.

Lazic, J., Hanafi, S., Mladenovic, N. ve Urosevic, D. 2010. “Variable neighbourhood decomposition search for 0-1 mixed integer programs”, Computers and Operations Research, 37, 1055-1067.

Levanova, T. ve Loresh, M.A. 2004. “Algorithms of ant system and simulated annealing for the p -median problem”, Automation and Remote Control, 65, 431-438.

Li, J-Q., Pan, Q-K. ve Xie, S-X. 2010. “A Hybrid Variable Neighborhood Search Algorithm for Solving Multi-Objective Flexible Job Shop Problems”, ComSIS, 7, 4.

Liang, Y-C. ve Lo, M-H. 2010. “Multi-objective redundancy allocation optimization using a variable neighborhood search algorithm”, Journal of Heuristics, 16, 511-535.

Liang, Y-C., Hsiao, Y-M. ve Tien, C-Y. 2013. “Metaheuristics for drilling operation scheduling in Taiwan PCB industries”, International Journal of Production Economics, 141, 189–198.

Liang, Y-C. ve Chuang, C-Y. 2013. “Variable neighborhood search for multi-objective resource allocation problems”, Robotics and Computer-Integrated Manufacturing, 29, 73–78.

Mladenovic, N., Hansen, P. 1997. “Variable Neighborhood Search”, Computers Operations Research, 24, 11, 1097-1100.

Mladenovic, N., Brimberg, J., Hansen, P. ve Moreno-Perez, J. A. 2007. “The p-median problem: A survey of metaheuristic approaches”, European Journal of Operational Research, 179, 927-939.

Mladenovic, N., Drazic, M., Kovacevic-Vujcic, V. ve Cangalovic, M. 2008. “General variable neighborhood search for the continuous optimization”, Computers and Operations Research, 191, 753-770.

Mladenovic, N., Urosevic, D., Perez-Brito, D. ve Garcia-Gonzalez, C. G. 2010. “Variable neighborhood search for bandwith reduction”, European Journal of Operational Research, 200, 14-27.

Nadirler D. ve Karasakal E. 2008. “Mixed integer programming-based solution procedure for single-facility location with maximin of rectilinear distance”, Journal of the Operational Research Society, 59, 563–570.

Ng, B.T. ve Han, J. 1994. “Efficient and effective clustering methods for spatial data mining”. 20th International Conference on Very Large Data Bases. Bocca, J. vd., Morgan, Kaufmann.

Onety, R. E., Tadei, R., Neto, O. M. ve Takahashi, R. H. C. 2013. “Multiobjective optimization of MPLS-IP networks with a variable neighborhood genetic algorithm”, Applied Soft Computing, 13, 4403–4412.

Özpeynirci, S. ve Azizoğlu, M. 2009. “Bounding approaches for operation assignment and capacity allocation problem in flexible manufacturing systems”, Computers & Operations Research 36, 2531 – 2540.

Özpeynirci, Ö. ve Köksalan, M. 2010. “An Exact Algorithm for Finding Extreme Supported Nondominated Points of Multiobjective Mixed Integer Programs”, Management Science, 56, 12, 2302–2315.

Perez, J. A. M., Moreno-Vega, J. M. ve Martin, I. R., 2003. “Variable neighborhood tabu search and its application to the median cycle problem”, European Journal of Operational Research, 151, 365-378.

Rajagopalan H.K., Saydam, C., Xiao, J. 2008. “A Multiperiod Set Covering Location Model for Dynamic Redeployment of Ambulances”, Computers and Operations Research, 35(3), 814-826.

Reinelt, G. 1991. “TSPLIB: A Traveling Salesman Problem Library”, ORSA Journal on Computing, 3, 376-384.

Resende, M. ve Werneck, R.F. 2003. “On the implementation of a swap-based local search procedure for the p-median problem”, 119-127. Proceedings of the 5th Workshop on Algorithm Engineering and Experiments. Ladner, R. E. SIAM, Philadelphia.

Revelle, C.S., Eiselt, H.A. ve Daskin, M.S. 2008. “A bibliography for some fundamental problem categories in discrete location science”, European Journal of Operational Research, 184, 817-848.

Ripon, K. S. N., Glette, K., Khan, K. N., Hovin, M. ve Torresen, J. 2013. “Adaptive variable neighborhood search for solving multi-objective facility layout problems with unequal area facilities”, Swarm and Evolutionary Computation, 8, 1–12.

Rolland, E., Schilling, D. A., Current, J. R. 1996. “An efficient tabu search procedure for the p-Median problem”, European Journal of Operational Research, 96 329-342.

Sevkli, M., Aydın, M. E. 2006. “Variable Neighborhood Search for Job Shop Scheduling Problems”, Journal of Software, 1, 34-39.

Solyali, O., Özpeynirci, Ö. 2009. “Operational Fixed Job Scheduling Problem Under Spread Time Constraints: A Branch-and-Price Algorithm”, International Journal of Production Research, 47:7,1887-1893.

Urosevic, D., Brimberg, J., Mladenoviç, N. 2004. “Variable Neighborhood Decomposition Search for the Edge Weighted k-Cardinality Tree Problem”, Computers and Operations Research, 31, 1205–1213.

Wang, Q., Batta, R., Joyendu, B., Rump, C. M. 2003. “Budget Constrained Location Problem with Opening and Closing of Facilities”, Computers and Operations Research, 30, 2047-2069.

Zhao, Q., Xie, C. ve Xiao, Y. 2012. “A variable neighborhood decomposition search algorithm for multilevel capacitated lot-sizing problems”, Electronic Notes in Discrete Mathematics, 39, 129-135.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evol. Comput.7(2), 117–132 (2003).

EK: Yayınlanmak Üzere Kabul Edilmiş Makale

Variable Neighborhood Search for the Pharmacy Duty Scheduling Problem

Fatih Kocatürk[a], Özgür Özpeynirci[b,*]

[a] *Department of Mathematics, İzmir University of Economics, İzmir, Turkey*
[b] *Department of Logistics Management, İzmir University of Economics, İzmir, Turkey*

Abstract

In this paper, we study on the Pharmacy Duty Scheduling (PDS) problem, where a subset of pharmacies should be on duty on national holidays, at weekends and at nights in order to be able to satisfy the emergency drug needs of the society. PDS problem is a multi-period p-median problem with special side constraints and it is an NP-Hard problem. We propose four Variable Neighborhood Search (VNS) heuristics. The first one is the basic version, BVNS. The latter two, Variable Neighborhood Decomposition Search (VNDS) and Variable Neighborhood Restricted Search (VNRS) aim to obtain better results in less computing time by decomposing or restricting the search space. The last one, Reduced VNS (RVNS), is for obtaining good initial solutions rapidly for BVNS, VNDS and VNRS. We test BVNS, VNRS and VNDS heuristics on randomly generated instances and report the computational test results. We also use VNS heuristics on real data for the pharmacies in central İzmir, and obtain significant improvements.

Keywords: pharmacy duty scheduling, variable neighborhood search, variable neighborhood restricted search, variable neighborhood decomposition search, PDS, VNS, VNRS, VNDS

1. Introduction

The Pharmacy Duty Scheduling (PDS) problem arises from the society's need for emergency drugs. The goal of the PDS problem is to assign duties

*Corresponding Author
Email address: ozgur.ozpeynirci@ieu.edu.tr (Özgür Özpeynirci)

to the pharmacies on national holidays, at weekends and weekday nights. Ağlamaz and Özpeynirci (2011) defined PDS problem as a multi-duty p-median problem and showed that PDS problem is an NP-Hard problem by reducing to the p-median problem. In this study, we develop four Variable Neighborhood Search (VNS) based heuristics for the PDS problem; (i) basic, (ii) initial solution provider, (iii) decomposed, and (iv) with restricted local search.

PDS problem is a society related real life problem and its solution affects many people. There are many studies related with healthcare problems, such as ambulance location, nurse scheduling and emergency facility location.

The Chamber of pharmacists is responsible for the duty scheduling of the pharmacies, which is currently prepared manually in a decentralized approach. In the current system, the pharmacies are grouped into regions based on their geographical locations and there is a representative pharmacist in each region. These representative pharmacists manually prepare duty schedules for their regions independently. The current decentralized system may involve excessive customer travel in order to reach the closest on duty pharmacy. On the other hand, PDS problem considers all regions simultaneously. The mathematical model and algorithms developed in this paper propose central duty schedules.

The rest of the paper is planned as follows. We provide a literature review in Section 2. We introduce the mathematical model of the problem in Section 3. We present the VNS heuristics in Section 4 and the computational results in Section 5. Finally, we conclude the paper in Section 6.

2. Literature Review

We present an extensive literature review of Variable Neighborhood Search (VNS) heuristic and its extensions in this section. We also discuss the facility location problem, the p-median problem and the emergency facility location problem.

Hale and Moberg (2003) define facility location problems as exploring where to physically locate a set of facilities so as to minimize the cost of satisfying some set of customer nodes, while satisfying some set of constraints. They also emphasize that facility location models can differ in their objective function, the distance metric applied, the number and the size of the facilities to locate and several other decision indices. ReVelle et al. (2008) classified the developments in location modeling into four categories: First, analytic

models based on a large number of simplifying assumptions; secondly continuous models assuming facilities can be located anywhere, while demands are often at discrete locations; thirdly network models assuming the location problem is embedded in network composed of links and nodes; and finally discrete location models in which demands and locations are discrete, as in the Pharmacy Duty Scheduling (PDS) problem case.

The p-median problem is a commonly used model in location science. Kariv and Hakimi (1979) classified the p-median problem as an NP-hard problem. Researchers use the p-median problem to model many real life problems regarding the location of industrial plants, warehouses, public facilities (Mladenović et al., 2007). Researchers have also applied the p-median problem to other research areas, such as cluster analysis (Hansen and Jaumard, 1997) and data mining (Ng and Han, 1994).

The computational complexity of the p-median problem has led the researchers to develop heuristics. Mladenović et al. (2007) identified two groups of heuristics for solving the p-median problem: classical heuristics and metaheuristics. Classical heuristics contained greedy, stingy, dual ascent, composite, alternate, interchange, dynamic programming, lagrangian relaxation and aggregation heuristics. Metaheuristics consisted of tabu search (TS), variable neighborhood search (VNS), genetic algorithm (GA), scatter search, simulated annealing (SA), heuristic concentration, ant colony optimization (ACO), neural networks, decomposition heuristics and hybrid heuristics (HH).

Alp et al. (2003) applied an efficient GA to the p-median problem and tested the algorithm on 80 test problems ranging 100 to 1000 nodes. Levanova and Loresh (2004) proposed SA algorithm and completely solved 17 out of the first 20 (among 40) OR-library test instances (Beasley, 1985). Resende and Werneck (2003) and Kochetov et al. (2005) suggested two different improvements of ACO and were able to solve all 20 OR-library instances.

Mladenović and Hansen (1997) introduced a new metaheuristic, VNS, an approach that avoids entrapment in local optima. They defined this simple and effective metaheuristic as a systematic change of neighborhood within a local search algorithm. Instead of following a normal trajectory, VNS searches in increasing distant neighborhoods of the current incumbent solution and jumps to a new solution if and only if an improvement was obtained.

Hansen and Mladenović (1997) proposed VNS heuristic for solving p-median problem and compared VNS with TS heuristics. They obtained better results against TS heuristics, but VNS heuristic took several hours for the large scale problems. In order to decrease computation time of the VNS

heuristic, Hansen et al. (2001) developed Variable Neighborhood Decomposition Search (VNDS). VNDS heuristic decomposes the search space and applies local search in a relatively smaller space. They illustrated VNDS on the p-median problem and tested on instances of 1400, 3038 and 5934 users from the TSP library (Reinelt, 1991). They showed that VNDS improved notably upon VNS in less computing time.

Hansen and Mladenović (2001) proposed several extensions of VNS for solving large scale problem instances: Reduced VNS (RVNS) heuristic discards the local search step of basic VNS heuristic in order to decrease the computation time for the large scale problems in which local search step takes long time. This allows a jump to a solution worse than the incumbent, with some probability, and allows easy exploration of valleys far from the incumbent solution. It also provides an efficient Skewed VNS (SVNS) heuristic, and these extensions of VNS were applied (by the researchers) to some classical optimization problems. Later, those researchers used the extensions of VNS to solve 0-1 mixed integer programming (MIP) problems with hybrid algorithms (see for example Lazić et al., 2010; Hanafi et al., 2010; Zhao et al., 2012).

Combining VNS with other heuristic methods generates new variants of VNS. Pérez et al. (2003) developed a new method, the variable neighborhood tabu search (VNTS), by combining VNS and TS method. García-López et al. (2002) developed the parallelization of VNS metaheuristic, parallel Variable Neighborhood Search (PVNS), which allowed high efficient exploration of space through the distribution of the steps of the algorithm among the available processors and then they tested PVNS on large scale instances of the p-median problem from TSP library. Crainic et al. (2004) defined Cooperative Parallel VNS (CPVNS) which cooperates several independent VNS metaheuristics by the asynchronous exchange of information about the most recent best solution. They experimented this method by using the classical TSP library problem instances with up to 11948 customers and 1000 medians. The results showed that cooperative method outperforms the recent methods in terms of computation time with no loss in solution quality.

In the literature, VNS heuristic and its extensions are applied to several optimization problems. Şevkli and Aydın (2006) proposed VNS heuristic to solve job shop scheduling problems, with better results than other proposed methods. Other problems solved by the VNS heuristics include minimum berth allocation problem (Hansen et al., 2008), location routing problem (Jarboui et al., 2013), bandwidth reduction problem (Mladenović et

al., 2010), constrained and unconstrained continuous optimization problems (Mladenović et al., 2008), harmonic means clustering problem (Alguwaizani et al., 2011), local branching problem (Hansen et al., 2006), k-cardinality tree problems (Brimberg et al., 2006; Urošević et al., 2004) and the capacitated p-median problem (Fleszar and Hindi, 2008).

Fire brigades, ambulances, police stations and hospitals are examples of important emergency service stations. The emergency service station (ESS) location problem is a very important problem that received great attention in the literature. Başar et al. (2012) propose a taxonomy for ESS location problems. Adenso-Díaz and Rodríguez (1997) suggested tabu search (TS) heuristic for locating ambulances in the Spanish province of León. Çatay et al. (2008) analyzed single and multi-period emergency station location problem for İstanbul, Turkey and propose heuristics to solve the large scale instances. Harewood (2002) proposed a multi-objective programming problem to locate ambulances on the island of Barbados. Carreras and Serra (1999) study on locating new pharmacies in the rural areas of Spain considering the service threshold levels and propose a TS heuristic. The PDS problem is also a special ESS problem. However, in PDS the locations of the pharmacies are known, and we deal with assigning duties to pharmacies in order to provide good quality service to emergency drug needs of the society.

3. Mathematical Model

Ağlamaz and Özpeynirci (2011) introduced the Pharmacy Duty Scheduling (PDS) problem and developed a mixed integer mathematical programming model for the PDS problem in cooperation with the Chamber of Pharmacists. The chamber is responsible for planning the duty schedules of pharmacies, which are clustered (for this purpose) into regions based on geographical locations. The number of pharmacies in regions may differ based on several factors such as the population density, hospital locations and preferences of the pharmacists.

Ağlamaz and Özpeynirci (2011) have made assumptions regarding the demand points and the distances between demand points and the pharmacies. They assume an aggregate customer node where a group of people living in the same neighborhood will be represented. They also assume that the customer nodes are located at the centers of the districts and the demand at each customer is proportional to the population. The distance between the customer nodes and pharmacy locations are known. The length of the planning

horizon (T) and the total number of duties which each of the pharmacies must take on during the planning horizon are inputs for the mathematical model.

Indices and Sets :
i: customer nodes (demand points), $i = 1, \ldots, I$
j: pharmacies (facility sites), $j = 1, \ldots, J$
t: time periods (days), $t = 1, \ldots, T$
k: regions of the pharmacies (facility sites), $k = 1, \ldots, K$
J_k: set of pharmacies in region k

Parameters:
h_i: demand at customer node i
d_{ij}: distance between customer node i and pharmacy j
n_j: total number of duties of pharmacy j in the planning horizon

Decision Variables:

$$y_{jt} = \begin{cases} 1, & \text{if pharmacy } j \text{ is on duty on day } t; \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{ijt} = \begin{cases} 1, & \text{if customer } i \text{ is served by pharmacy } j \text{ on day } t; \\ 0, & \text{otherwise.} \end{cases}$$

Mathematical Model:

$$Min\ F = \sum_{i=1}^{I} \sum_{j=1}^{J} \sum_{t=1}^{T} h_i d_{ij} x_{ijt} \tag{1}$$

Subject to

$$x_{ijt} \leq y_{jt} \quad \forall i, j, t \tag{2}$$

$$\sum_{t=1}^{T} y_{jt} = n_j \quad \forall j \tag{3}$$

$$\sum_{j \in J_k} y_{jt} = 1 \quad \forall k, t \tag{4}$$

$$\sum_{j=1}^{J} x_{ijt} = 1 \quad \forall i, t \tag{5}$$

$$y_{jt} \in \{0, 1\} \quad \forall j, t \tag{6}$$

$$x_{ijt} \in \{0, 1\} \quad \forall i, j, t \tag{7}$$

In this model, the objective function (1) minimizes the total demand weighted distance traveled by all customers during the planning horizon. Constraint set (2) states that in order to assign customer i to pharmacy j on day t, the pharmacy should be open on day t. Constraint set (3) show the exact number of duties to be assigned to pharmacy j during the planning horizon. Constraint set (4) states that only one pharmacy can be opened from each region on each day of the planning horizon. Constraint set (5) states that each customer should be assigned to one pharmacy on each day of the planning horizon. Finally, constraint sets (6) and (7) define the binary decision variables.

We can relax the binary decision variable x_{ijt} as continuous since there is no capacity constraint for pharmacies and each customer will be assigned to the closest open pharmacy. Moreover, the equality of the constraint set (4) can be replaced by less than or equal to constraint and that of the constraint set (5) by greater than or equal to constraint due to the minimization type objective function.

For a fair distribution of duties, n_j values satisfy the following inequality:

$$\max_{j \in J_k}\{n_j\} - \min_{j \in J_k}\{n_j\} \leq 1 \ \forall k. \tag{8}$$

This inequality ensures that the number of duties differ at most one among the pharmacies in the same region. The current system and the randomly generated instances satisfy inequality (8).

Note that, some pharmacies may be on duty on consecutive days in the optimal solution of the above model. However, we do not add constraints to

avoid such schedules. Instead, we suggest a post processing of the optimal solution. For each day of the planning horizon, the model suggests a list of pharmacies, one from each region, to be on duty on the same day. There are roughly $O(T!)$ different ways of assigning these lists to corresponding calendar days. Given T daily lists representing the optimal solution, one can pick the lists that pharmacy j is on duty and assign them to the first n_j days of the planning horizon. In this case, pharmacy j will be on duty for n_j consecutive days. Alternatively, one can develop a model that assigns daily lists to calendar days while minimizing the total number of consecutive duties.

4. Variable Neighborhood Search Algorithms

The Pharmacy Duty Scheduling (PDS) problem has similarities with the p-median problem in terms of the objective function and the uncapacitated candidate facilities. Moreover, Ağlamaz and Özpeynirci (2011) showed that PDS is NP-Hard by a reduction to the p-median. Hansen and Mladenović (1997) applied Variable Neighborhood Search (VNS) heuristic method to the p-median problem and they showed that VNS gave consistently better results compared to Greedy plus Interchange and two recent Tabu search heuristics on average scale instances from ORLIB library and larger scale instances derived from the TSPLIB library. Hansen et al. (2001) showed that Variable Neighborhood Decomposition Search (VNDS) can be very useful for large scale problems, although compared to VNS, results for medium scale instances are not always satisfactory.

In this section, we develop a Basic VNS (BVNS) heuristic for PDS problem. We define the neighborhood structure, local search methodology and the stopping conditions. In order to obtain good quality initial solutions, we propose a Reduced VNS (RVNS) that omits the local search step. We develop two variants of BVNS for solving large scale instances. The first variant is Variable Neighborhood Decomposition Search (VNDS) that solves relatively smaller decomposed problems. The second variant is Variable Neighborhood Restricted Search (VNRS) that applies local search in a restricted search space.

4.1. Basic Variable Neighborhood Search

We denote a feasible solution z as a $K \times T$ matrix, $z \in \mathbb{N}^{K \times T}$, and the entry z_{kt} of z represents the index of the on duty pharmacy in region k on

Algorithm 1 BVNS Algorithm

Initialization: Generate an initial solution (schedule) z, set the neighborhood structures $N_k,\ k = \{1, \ldots, k_{max}\}$ and define a stopping condition rule.

Main Step: Repeat the following steps until the stopping condition is satisfied,

(1) $k = 1$,

(2) Until $k = k_{max}$ repeat the following steps,

(a) *Shaking*: Generate randomly a solution, z', from the k^{th} neighborhood of z, $(z' \in N_k(z))$,

(b) *Local Search*: Find the local minimum, z'', around the solution z' using the swap algorithm,

(c) *Move*: If $f(z'') < f(z)$ then change the incumbent solution $(z = z'')$ and return $N_1, (k = 1)$ and continue to search from there. Otherwise, increase the neighborhood $(k = k + 1)$.

day t. Note that, $z_{kt} = \sum_{j \in J_k} j \times y_{jt}$, where J_k is the set of pharmacies in region k. The set of feasible solutions is Z, $Z = \{z^1, z^2, \ldots\}$.

We present the BVNS algorithm for the PDS problem in Algorithm 1. In the initialization step, we generate a feasible initial solution as follows: For each pharmacy j, we assign n_j consecutive duties starting from the earliest possible day in its region. Then, we use RVNS heuristic (will be discussed in 4.2) to improve the initial solution before starting the main step.

We use the swap algorithm to move from one feasible solution to another. Consider a solution $z \in Z$. In a swap, we pick a region, and two days such that different pharmacies are on duty, say region k and days t_1 and t_2 such that $z_{kt_1} \neq z_{kt_2}$. We swap the values of the entries z_{kt_1} and z_{kt_2} and obtain a new solution, $z' \in Z$. In case of multiple swaps, we pick a different region for each swap.

We measure the distance between two feasible solutions z^1 and z^2 by the number of swaps applied. In order to count this, we define a new operator $\ominus$ as follows:

$$z^1 \ominus z^2 = \frac{\sum_{t=1}^{T} |OD_t^1 \backslash OD_t^2|}{2},$$

where OD_t^s is the set of on duty pharmacies on day t in solution $z^s, s = 1, 2$. We divide by two in order to avoid double counting since one swap

changes two columns of a solution. We construct the neighborhood structures, $N_k,\ k = \{1, \ldots, k_{max}\}$, by using the distance metric ρ which is defined as,

$$\rho(z^1, z^2) = z^1 \ominus z^2 = z^2 \ominus z^1, \forall z^1, z^2 \in Z$$

where z^1 and z^2 are two feasible solutions in Z.

The main step of the algorithm starts from $k = 1$ and it iterates until $k = k_{max}$, where k_{max} is a parameter satisfying $k_{max} \leq K_2$ and K_2 is the number of regions in which there are two or more pharmacies. We use K_2 instead of K since swapping in a region requires at least two different pharmacies in that region. We define three parameter levels for k_{max} and test these levels.

We generate random solutions in the shaking step 2(a) according to algorithm 2. In order to obtain z' from z, we first pick k regions. Then in each region, we pick two different pharmacies. Lastly, if $n_j > 1$ for a selected pharmacy j, we select one of the days that pharmacy j is on duty. We then apply k swaps, one in each region to obtain z'.

We use a controlled random procedure for the above mentioned selection of regions, pharmacies and days. We give greater importance (i) to the regions with a high number of pharmacies, (ii) to the pharmacies with higher n_j values, and (iii) to the days that increase objective function more. Note that, both rules (i) and (ii) aim to provide a larger search space. The former rule increases the number of possible swaps and the latter one increases the number of candidate days for rule (iii). We assign random values for each region, pharmacy and day. Let these values be

- $rand\_region_k = U(0,1) * |J_k|$ for regions, where J_k is the set of pharmacies in region k and $U(0,1)$ is a uniform random variable between 0 and 1,
- $rand\_pharmacy_j = U(0,1) * n_j$ for pharmacies
- $rand\_day_t = U(0,1) * (f_t - f_{min} + 1)$ for days, where f_t is the objective function value for day t, $f_{min} = \min_t\{f_t\}$ and $F = \sum_t f_t$.

Whenever necessary, we pick the highest valued regions, pharmacies and days. In order to consider different parts of the search space, Shaking Algorithm gives priority to the unvisited regions first. For this purpose, the algorithm sets $rand\_region_k = -1$ for the selected regions. Let us define k^-

as the number of regions with $rand\_region_k = -1$. At an iteration, the algorithm selects k regions. If $K_2 - k^- < k$, then the algorithm concludes that there are not enough regions with nonnegative $rand\_region_k$ values and generates new $rand\_region_k$ values for all regions. On the other hand, Shaking Algorithm generates new $rand\_pharmacy_j$ and $rand\_day_t$ values in every iteration.

Algorithm 2 Shaking Algorithm

(1) **Sorting:**
(a) Sort regions in decreasing order of $rand\_region_k$ values,
(b) Sort pharmacies in decreasing order of $rand\_pharmacy_j$ values,
(c) Sort days in decreasing order of $rand\_day_t$ values,
(2) Select the first k regions and let K^* be the set of selected regions,
(3) For each $k \in K^*$, select the first two pharmacies j^*_{1k} and j^*_{2k} such that $j^*_{1k}, j^*_{2k} \in J_k$,
(4) For each $k \in K^*$ and $j^*_{hk} \in J_k,\ h = 1, 2$, if $n_{j^*_{hk}} > 1$ then select the first day that j^*_{hk} is on duty,
(5) For each $k \in K^*$, swap the duties of pharmacies j^*_{1k} and j^*_{2k} on the selected days,
obtain z' from z,
(6) **Update:**
(a) Set $rand\_region_k$ values to -1 for $k \in K^*$,
(b) Let k^- be the number of regions for which $rand\_region_k$ values equal to -1, if $k > K_2 - k^-$ then reassign $rand\_region_k$ values,
(c) Randomly generate $rand\_pharmacy_j$ and $rand\_day_t$ values,

In step 2(b) of the algorithm, we apply the local search around the randomly generated solution, z'. The local search method used in BVNS is independent of the neighborhood structure, N_k, and always uses $k = 1$. The local search starts from the first region and applies all possible duty swap combinations until the last region. We apply the first improvement strategy during the local search until we get the local optimum solution, i.e. if we detect an improvement, we apply the swap to the solution and update the local minimum. We restart the search from the updated solution.

After reaching the local optimum, denoted z'', we compare the objective function values of z'' and the incumbent solution z in the move step 2(c). If $f(z'') < f(z)$ then we update the incumbent solution ($z = z''$) and return

to the first neighborhood $N_1, (k = 1)$ and continue to search from there. Otherwise, we increase the neighborhood $(k = k + 1)$.

We define three types of stopping condition rules: (i) Maximum CPU time, (ii) Maximum global iteration number and (iii) Maximum outer iteration number that counts the iterations that the algorithm returns to the main step using the best known solution as an initial solution. We use maximum CPU time allowed condition simultaneously with the other stopping conditions. We use the same maximum CPU time allowed condition for BVNS, VNDS and VNRS heuristics to compare them accurately. We defined and tested two parameter levels for both maximum global iteration and maximum outer iteration conditions.

4.2. Reduced Variable Neighborhood Search

Hansen et al. (2001) emphasized that getting a good initial solution quickly is important for solving large scale problem instances. They developed Reduced VNS (RVNS) heuristic, which omits the local search step in BVNS algorithm, resulting in a significant decrease in computational time. Grujičić and Stanimirović (2012) used RVNS to get an improved initial solution before starting the basic VNS algorithm to optimize the emergency service network of police special forces units.

RVNS follows all steps of BVNS algorithm except local search in 2(b). In each step, RVNS generates a random solution from the k^{th} neighborhood of the incumbent solution and moves to this solution if it is better than the incumbent. We use the maximum non-improvement number as the stopping condition and determine it as $100 * K$. We obtain considerable good initial solutions rapidly by this method.

We use the same initial solution generation method for BVNS, VNDS and VNRS. We first generate an initial solution as described in Section 4.1 and improve this solution using RVNS.

4.3. Variable Neighborhood Decomposition Search

In the literature, Variable Neighborhood Decomposition Search (VNDS) algorithm is proposed to obtain good quality solutions in less computing time especially for the large scale problems. We aim to achieve this by decomposing the solution space and applying the local search in a relatively smaller space.

VNDS randomly selects k regions and considers the customers that can be affected by duty changes in the selected regions. Note that these customers is a subset of the customers set. VNDS applies a heuristic search in the decomposed solution space and stops with some stopping condition. It then combines the obtained solution for the decomposed space and the rest of the solution for the overall problem.

We present the steps of VNDS in Algorithm 3. In initialization step, we generate an initial feasible solution, define the neigborhood structures, and the stopping condition. Also we define I_k, the subset of customers that should be considered if region k is selected, for all regions.

Let $I_k = \{i : \alpha_i^k = 1, i \in I\}$ where $d_i^k = \min_{k_2 \in K \setminus \{k\}} \max_{j \in J_{k_2}} d_{ij}$ and

$$\alpha_i^k = \begin{cases} 1, & \text{if there exists a } j \in J_k \text{ such that } d_{ij} \leq d_i^k; \\ 0, & \text{otherwise.} \end{cases}$$

The parameter d_i^k is the worst case distance for customer i when there are no on duty pharmacies in region k.

In step 1, VNDS starts with $k = 2$. We omit $k = 1$ since every schedule leads to the same objective function value for the decomposed problem in a single region.

In step 2, VNDS sets the neighborhood structure to N_k and follows the shaking, local search and move steps. In step 2(a), VNDS has to select k regions. It first selects the region with the highest $rand\_region_k$ value. Then, it iteratively adds the closest region to the already selected regions by considering the average distance between regions until k regions are selected. The $rand\_region_k$ value is set as -1 for the selected regions. VNDS assigns new $rand\_region_k$ values when all $rand\_region_k$ values are -1. Note that a selected region can not be the first region in the next iterations but still can be selected. Let $K^\star$ be the set of selected regions in that iteration and, $I^\star = \bigcup_{k \in K^\star} I_k$ be the set of customers that may be affected by the swaps in the selected regions.

Algorithm 3 VNDS Algorithm

Initialization: Generate an initial solution (schedule) z, set the neighborhood structures $N_k,\ k = \{2, \ldots, k_{max}\}$, set customer subsets $I_k, k \in K$ and define a stopping condition rule.

Main Step: Repeat the following steps until the stopping condition is satisfied,

(1) $k = 2$,

(2) Until $k = k_{max}$ repeat the following steps,

(a) *Shaking*: Generate randomly a solution, z', from the k^{th} neighborhood of z, $(z' \in N_k(z))$, let w be the solution formed by randomly selected regions $k \in K^\star$ of z' and customers $I^\star = \bigcup_{k \in K^\star} I_k$ such that $w \in \mathbb{N}^{k \times T}$,

(b) *Local Search*: Find the local minimum, w', in the space of w using the swap algorithm, and denote the corresponding local minimum with z'' in the whole space Z, $(z'' = (z' \backslash w) \cup w')$,

(c) *Move*: If $f(z'') < f(z)$ then change the incumbent solution $(z = z'')$ and return $N_2, (k = 2)$ and continue to search from there. Otherwise, increase the neighborhood $(k = k + 1)$.

Consider a case where the selected regions are far away from each other. In this case, each customer will be affected by the swaps in only one region and the objective function value does not change. This is a similar case to $k = 1$ case as defined above. Hence, we aim to deal with k regions that are close to each other so that for some customers there are candidate pharmacies in different regions and swaps may change the objective function value.

We know the distances between the pharmacies and the customers for a given instance. We define the distance between regions k_1 and k_2 as $\bar{d}_{k_1 k_2} = \frac{\sum_{j_1 \in k_1} \sum_{j_2 \in k_2} d'_{j_1 j_2}}{|J_{k_1}| \times |J_{k_2}|}$, where $d'_{j_1 j_2} = \min_i \{d_{i j_1} + d_{i j_2}\}$.

Similar to BVNS, VNDS applies one swap in each selected region and obtains z' from z. VNDS generates decomposed solution $w \in \mathbb{N}^{k \times T}$ by considering regions in $K^\star$, customers in $I^\star$ and whole planning horizon of T days.

In step 2(b), VNDS applies a heuristic search on w and obtains w' as defined in Hansen and Mladenović (2001) and Hansen et al. (2001). Note that any heuristic method is applicable for this purpose. Similar to BVNS, VNDS uses the swap algorithm in this step. The main difference is that BVNS applies the swap algorithm to the whole problem whereas VNDS applies it

to the decomposed problem with less number of regions, pharmacies and customers.

VNDS obtains w' for the decomposed problem. Then it combines this solution with the remaining of the solution z' ($z' \backslash w$), and obtain the solution z'' ($z'' = (z' \backslash w) \cup w'$).

In step 2(c), we compare the objective function values of z'' and the incumbent solution z. If $f(z'') < f(z)$ then we update the incumbent solution ($z = z''$) and return to the second neighborhood $N_2, (k = 2)$ and continue to search from there. Otherwise, we increase the neighborhood ($k = k + 1$).

4.4. Variable Neighborhood Restricted Search

We propose Variable Neighborhood Restricted Search (VNRS) algorithm to obtain good quality solutions in short computing times, especially for the large scale problems. We aim to achieve this by applying the local search in a relatively smaller space, however without decomposing the problem.

There are three main differences between VNDS and VNRS algorithms: (i) region selection strategy and, (ii) local search, (iii) minimum k value. VNDS selects regions close to each other and only the first region to be selected must have a nonnegative $rand\_region_k$ value. Whereas VNRS selects regions randomly without considering proximities among them and each selected region must have nonnegative $rand\_region_k$ value. VNRS and BVNS follow the same strategy to update random values for regions, days and pharmacies.

In the local search step, both VNDS and VNRS apply swap algorithm only in selected regions. VNDS evaluates the saving of a swap by considering the customers in $I^\star$ and regions in $K^\star$ where as VNRS evaluates by considering all customers and regions.

In VNDS, k value varies between 2 and k_{max}, since $k = 1$ does not make sense for VNDS. However, in VNRS, a swap in a single region may change the objective function value since we consider all customers and regions. Hence VNRS allows $k = 1$ as well.

We conduct a detailed analysis on the local search procedures of BVNS, VNDS and VNRS. We analyze the cardinality of the neighborhood and complexity of computing the objective function value of a solution after performing one move. We present the results in Table 1. For BVNS, there are $T!$ possible swaps for a region. Hence, the cardinality of the neighborhood of the local search is $O((T!)^K)$. VNDS and VNRS apply local search in k regions. The neighborhood is $O(T!)$, since k is a parameter for the algorithms.

After performing one move, the schedules of only two days change and it is enough to compute the new objective function values for these days. BVNS and VNRS consider the whole customer set. Both algorithms detect the nearest on duty pharmacy out of K regions for each customer and the computational complexity of computing new objective function value is $O(IK)$. On the other hand, VNDS considers only a subset of the customers ($i \in I^{\star}$). A swap in any of the regions in K^*, where $|K^*| = k$, may change the pharmacy assignments of these customers. For VNDS, the computational complexity of computing the new objective function value is $O(I)$.

The local search starts from the region with the smallest index and analyzes the regions in increasing order of region index. We believe alternative policies that uses different orders of analyzing regions may be a possible future research direction.

Table 1: Complexity Analysis of Local Search

	BVNS	**VNDS**	**VNRS**
Cardinality of Neighborhood	$O((T!)^K)$	$O(T!)$	$O(T!)$
Complexity of Computing Obj. Func.	$O(IK)$	$O(I)$	$O(IK)$

5. Computational Experiments

We coded the algorithms in C environment and run on an Amd Phenom II X4 955 3.2 GHz Processor with 4GB memory.

We tested the algorithms on randomly generated instances by Ağlamaz and Özpeynirci (2011). There are three groups of test problems; small, large and real life. In the first group, there are 9 problem sizes and 10 instances for each problem size. In the second group, there are 3 problem sizes and 5 instances for each problem size. In the real life group, there are two instances.

We applied preliminary experiments for parameter k_{max} and the stopping conditions for BVNS, VNDS and VNRS algorithms. We determined three levels for the parameter k_{max}: $4, \lceil K/2 \rceil$ and $\min(K, 9)$. The preliminary experiments suggested the k_{max} parameter level $\lceil K/2 \rceil$ for BVNS, VNDS and VNRS.

We determined four levels for the stopping conditions: two levels for the maximum global iteration number ($25,000$ and $100{*}T$), and two levels for the maximum outer iteration number (750 and $10 * T$). We selected the stopping

condition as $10 * T$ for BVNS, VNDS and VNRS algorithms. We also used maximum CPU time as a stopping condition and set it as 2.5 hours.

Ağlamaz and Özpeynirci (2011) developed a lower bound for the PDS problem. This lower bound considers each customer separately and computes the least possible distance that each should traverse in a duty schedule. For a given customer i, the lower bound detects the closest pharmacy, say pharmacy j, assigns the customer to this pharmacy for n_j days and proceeds to the next closest pharmacy until the end of the planning horizon.

We run BVNS, VNDS and VNRS algorithms once for each instance. We measure the performance of the algorithms according to two criteria: the gap between the upper and the lower bounds, and the CPU time. The gap is $Gap = \frac{UB-LB}{LB}$, where UB is the objective function value of the analyzed solution and LB is the lower bound computed by the algorithm of Ağlamaz and Özpeynirci (2011). We also analyze the gaps using the optimal solutions as LB for a subset of small scale problems.

In the current system, each region prepares its schedule independently and without any information from the other regions. We believe that a random strategy would simulate the current system well. Hence, we generate 10 random solutions for each instance in order to have a base performance measure. For each day and region, we randomly assign an eligible pharmacy and update the corresponding n_j value until we obtain a solution. We report the average gap of these random solutions. With this information, we can assess the incremental benefits obtained by the proposed methods over the current system. It is interesting to observe that the performance of the random strategy and the current system are close for two real life instances (see Table 7). We do not report the CPU times for random strategy since all are less than one second.

We compare the results of BVNS, VNDS and VNRS algorithms as well as the random strategy. As we mentioned in Section 4.2, BVNS, VNDS and VNRS algorithms start with the same initial solution for a fair comparison. We test the differences between the means of gaps and CPU times by considering every pair of algorithms. We use paired t-test since we have two observations for each instance, one for each algorithm under consideration. We do not test the CPU time differences for the random strategy since they are very small. We do not test the CPU time differences for large scale instance because all algorithms hit the CPU time limit.

Let $A^i \in \{$ BVNS, VNDS, VNRS, Random $\}$ denote an algorithm, $i = 1, 2$ and A^i_o be the value of the o^{th} observation for the considered criterion (gap

or CPU time) for algorithm A^i. For comparing algorithms A^1 and A^2, where A^1 is the algorithm with the smaller mean for the considered criterion (gap or CPU time), we design the following hypothesis:

$$H_0 : \mu_D \leq 0$$

$$H_a : \mu_D > 0$$

with a $1-\alpha = 0.95$ confidence where $D_o = A^1_o - A^2_o$, and D is the mean of D_o values. There are 90 and 15 instances in small and large scale problems, respectively. We report results of the paired t-tests in Table 2 for each criteria, problem scale and pair of algorithms. The critical t statistic values are $t_{0.05,89} = 1.66$ and $t_{0.05,14} = 1.76$ for small and large scale problems, respectively. The absolute values of test results are larger than the corresponding critical values. Hence we reject H_0, and conclude that all tested differences are statistically significant. For the sake of simplicity, we use the significance term for the statistical significance in the remainder of the section.

Table 2: t-Statistic Table

Criteria	**Problem Scale**	**# of observations**	**t-statistics**				
					A^2		
					VNRS	**VNDS**	**Random**
Gap (%)	Small	90	A^1	**BVNS**	-3.87	-7.67	-22.14
				VNRS		-6.90	-22.19
				VNDS			-23.32
					A^2		
					BVNS	**VNDS**	**Random**
	Large	15	A^1	**VNRS**	-2.47	-4.98	-22.25
				BVNS		-8.73	-30.05
				VNDS			-31.98
					A^2		
					VNRS	**BVNS**	
CPU (secs)	Small	90	A^1	**VNDS**	-6.73	-6.26	
				VNRS		-6.09	

In Table 3, we present the computational test results for small scale instances. The first four columns represent the problem size; numbers of customers (I), pharmacies (J), planning periods (T) and regions (K). The fifth column shows the average gaps obtained by randomly generated 10 schedules for each problem size. The next three columns report the average gaps (of 10 instances in each problem size) obtained by BVNS, VNDS and VNRS algorithms. The remaining columns report the average CPU times.

Table 3: Test Results for Small Scale Problems

I	J	T	K	Gap (%)				CPU (Secs)		
				Random	BVNS	VNDS	VNRS	BVNS	VNDS	VNRS
20	10	5	4	3.2	0.5	0.5	0.5	0.0	0.0	0.0
40	20	5	4	4.6	0.6	0.6	0.6	0.1	0.0	0.0
20	20	10	4	5.5	0.4	0.6	0.4	0.2	0.0	0.1
60	30	5	9	3.6	0.3	0.4	0.3	0.7	0.0	0.2
20	30	15	9	5.0	0.2	0.3	0.2	5.2	0.2	1.7
40	40	10	9	5.8	0.5	0.7	0.6	5.4	0.2	1.7
60	60	10	9	7.9	0.7	1.2	0.8	13.8	0.9	4.4
40	60	15	9	7.8	0.7	1.4	0.7	29.9	2.7	9.5
60	90	15	9	8.9	0.7	1.7	0.8	62.9	5.8	18.2

For the random strategy, the average gap increases (with a decreasing rate) as the problem size increases and all BVNS, VNDS and VNRS algorithms generate high quality solutions, significantly better than the random strategy. The problem parameters affect the solution quality of random strategy, the gap increases as the planning horizon length increases. The solution quality of random strategy is not bad. We believe that this is due to the distribution of the regions, and the fact that each region has one on duty pharmacy everyday. This increases the probability that customers will find an on duty pharmacy near them.

According to the experiment results on small scale problems, we discuss the performances of the algorithms. In terms of solution quality, BVNS is significantly better than VNRS and VNRS is significantly better than VNDS. For the CPU time, we observe exactly the reverse order, VNDS outperforms VNRS significantly and VNRS outperforms BVNS significantly.

In Table 4, we present detailed test results for BVNS, VNDS and VNRS algorithms. We report the minimum, average, maximum and standard deviation values of solution quality and CPU time. The gaps vary between 0% and 2.4%. We can not observe any strict trends. However, as the problem size increases the minimum and average gap values tend to increase whereas the maximum and standard deviation values tend to decrease. The gap values corresponding to BVNS are less than or equal to those of VNRS and the gap values corresponding to VNRS are less than VNDS. All CPU time related values corresponding BVNS are greater than or equal to those of VNRS and CPU time values corresponding VNRS are greater than those of VNDS. CPU time values increase rapidly as the problem size increases. Considering the coefficient of variation values, the algorithms do not dominate each other in solution quality and CPU time measures.

Table 4: Detailed Test Results of BVNS, VNDS and VNRS for Small Scale Problems

I	J	T	K	Heuristic Type	Gap (%)				CPU (Secs)			
					Min	Average	Max	Std. Dev.	Min	Average	Max	Std. Dev.
20	10	5	4	BVNS	0.0	0.5	1.6	0.6	0.0	0.0	0.0	0.0
				VNDS	0.0	0.5	1.6	0.5	0.0	0.0	0.0	0.0
				VNRS	0.0	0.5	1.6	0.6	0.0	0.0	0.0	0.0
40	20	5	4	BVNS	0.2	0.6	1.5	0.4	0.0	0.1	0.1	0.0
				VNDS	0.2	0.6	1.5	0.4	0.0	0.0	0.0	0.0
				VNRS	0.2	0.6	1.4	0.4	0.0	0.0	0.0	0.0
20	20	10	4	BVNS	0.0	0.4	1.0	0.4	0.1	0.2	0.3	0.1
				VNDS	0.0	0.6	1.7	0.5	0.0	0.0	0.1	0.0
				VNRS	0.0	0.4	1.1	0.4	0.0	0.1	0.1	0.0
60	30	5	9	BVNS	0.0	0.3	0.7	0.2	0.3	0.7	1.1	0.2
				VNDS	0.1	0.4	0.8	0.2	0.0	0.0	0.1	0.0
				VNRS	0.1	0.3	0.7	0.2	0.1	0.2	0.4	0.1
20	30	15	9	BVNS	0.1	0.2	0.5	0.2	1.3	5.2	9.2	2.3
				VNDS	0.1	0.3	0.7	0.2	0.0	0.2	0.7	0.2
				VNRS	0.0	0.2	0.5	0.2	0.4	1.7	3.2	0.8
40	40	10	9	BVNS	0.1	0.5	1.3	0.4	1.8	5.4	8.5	2.2
				VNDS	0.3	0.7	1.5	0.4	0.0	0.2	0.5	0.2
				VNRS	0.1	0.6	1.4	0.4	0.6	1.7	2.7	0.7
60	60	10	9	BVNS	0.5	0.7	1.0	0.2	12.6	13.8	15.6	1.1
				VNDS	0.8	1.2	1.7	0.3	0.5	0.9	1.1	0.2
				VNRS	0.5	0.8	1.1	0.2	3.5	4.4	5.7	0.6
40	60	15	9	BVNS	0.3	0.7	1.1	0.3	23.1	29.9	34.9	3.9
				VNDS	0.8	1.4	2.1	0.4	1.9	2.7	3.4	0.5
				VNRS	0.4	0.7	1.1	0.3	8.1	9.5	11.3	1.0
60	90	15	9	BVNS	0.3	0.7	1.1	0.2	57.1	62.9	74.7	4.9
				VNDS	0.8	1.7	2.4	0.5	4.6	5.8	6.5	0.7
				VNRS	0.4	0.8	1.4	0.3	16.6	18.2	20.6	1.2

Ceyhan and Özpeynirci (2013) study on an exact algorithm for PDS and solve the small scale problems. The authors reported that only for four problem sizes, all 10 instances can be solved to optimality in one hour CPU time limit. These problem sizes are I20 J10 T5 K4, I40 J20 T5 K4, I60 J30 T5 K9 and I20 J30 T15 K9. We compute the relative gaps using the optimal solutions instead of the lower bound of Ağlamaz and Özpeynirci (2011) for BVNS. The average relative gaps are 0.00%, 0.01%, 0.05% and 0.02% respectively. The maximum relative gap is 0.19%. In Table 3, we reported these gaps as 0.5%, 0.6%, 0.3% and 0.2%. Using the optimal solutions as lower bounds improves the perceived quality of the solutions proposed by the heuristic algorithms. However, for medium and large scale problems, determining the optimal solution is not practical. The perceived solution quality depends on the lower bound quality and developing a stronger lower bound is a future research direction.

Table 5: Test Results for Large Scale Problems

I	J	T	K	Gap (%)				CPU (Secs)		
				Random	BVNS	VNDS	VNRS	BVNS	VNDS	VNRS
350	250	25	49	9.1	1.0	1.6	1.2	9002.1	9000.1	9000.5
350	250	25	49	7.9	1.1	1.3	1.1	9000.5	9000.0	9000.1
350	250	25	49	8.2	1.1	1.6	1.2	9000.4	9000.0	9000.8
350	250	25	49	7.4	0.9	1.2	1.0	9000.0	9000.1	9000.5
350	250	25	49	8.3	1.0	1.3	1.2	9001.5	9000.1	9001.4
350	500	50	49	10.2	1.3	2.1	1.1	9007.0	9000.1	9003.2
350	500	50	49	10.6	1.2	2.0	1.1	9004.2	9001.0	9000.5
350	500	50	49	11.0	1.2	1.9	1.2	9006.4	9000.2	9002.1
350	500	50	49	10.6	1.0	1.6	1.0	9001.4	9001.0	9003.4
350	500	50	49	10.2	1.2	2.1	1.1	9002.8	9000.1	9000.5
350	1000	100	49	11.4	2.6	3.4	0.9	9001.6	9000.3	9000.8
350	1000	100	49	11.6	2.4	3.4	0.9	9001.4	9000.8	9004.2
350	1000	100	49	10.6	2.3	2.8	0.8	9004.3	9000.1	9000.7
350	1000	100	49	12.0	2.5	2.7	0.9	9001.7	9000.1	9000.3
350	1000	100	49	11.5	2.2	2.9	0.9	9003.6	9000.2	9000.7

We present the computational test results for large scale instances in Table 5. All of BVNS, VNDS and VNRS are significantly better than the random strategy. In terms of solution quality, VNRS is significantly better than BVNS and BVNS is significantly better than VNDS. However, we do not compare CPU times, since all algorithms terminate after 2.5 hours CPU time limit.

5.1. İzmir Application

We tested the proposed VNS algorithms on the Pharmacy Duty Scheduling problem in province İzmir of Turkey. We present the district numbers, the pharmacy numbers, the length of the planning horizons of the real data and the region numbers for two periods for İzmir in Table 6.

We tested these two problems with BVNS, VNRS and VNDS algorithms by using maximum CPU time stopping condition (2.5 hrs). Table 7 shows the results of the algorithms. According to the results, current solutions are far from the lower bounds by approximately 6%. We also generated 10 random schedules for both planning periods, and found that the respective average costs are very close to those of the current solutions. Moreover, VNRS algorithm gives better results than BVNS and VNDS algorithms for both problems and decreases the gap close to 0.5%.

Table 6: İzmir Application

Period	**I**	**J**	$\boldsymbol{T}$	$\boldsymbol{K}$
2010 / 3	335	1046	97	45
2011 / 1	335	1053	101	45

Table 7: Comparison of the Results

Planning Period	**Comparison Type**	**Random**	**Current**	**BVNS**	**VNRS**	**VNDS**	**LB**
2010/3	**Cost (Million km)**	309.27	309.01	294.76	293.46	295.34	291.89
	Gap (%)	5.96	5.87	0.98	0.54	1.18	-
2011/1	**Cost (Million km)**	321.41	321.12	306.28	304.82	307.55	303.28
	Gap (%)	5.98	5.88	0.99	0.51	1.41	-

6. Conclusion

In this study, we developed BVNS, VNDS and VNRS algorithms to solve the NP-Hard PDS problem. We aimed to improve on the results of BVNS in terms of solution quality and computational time with VNRS by restricting the local search and with VNDS by decomposing the solution space for large scale problem instances. We developed RVNS for a quick initial solution generator for BVNS, VNDS and VNRS.

We tested the algorithms on randomly generated test instances and obtained significant results for the PDS problem. The test results showed that VNDS and VNRS algorithms do not give as good solutions as BVNS for small scale instances, but they decrease the computational time significantly. However, VNRS gives better results than BVNS in the same computational time for large instances. For both problem sizes, BVNS and VNRS are significantly better than VNDS in terms of solution quality, although VNDS requires less computation time for small scale instances.

We applied the proposed VNS algorithms on real data for pharmacies in central İzmir. We observed that the costs of the current solutions of the real problems are very close to the average costs of the random solutions. We obtained significant improvements using BVNS, VNDS and VNRS algorithms, and VNRS algorithm outperformed BVNS and VNDS by decreasing the gap close to 0.5% for both problems.

This study focused on the PDS problem for customer utility maximization. Future studies can focus simultaneously on pharmacy utility and customer utility maximization. Moreover, other improvements, such as enhancing the lower bound algorithm will enable more effectively testing of the solutions.

7. Acknowledgement

The authors acknowledge the support of the Scientific and Technological Research Council of Turkey (TÜBİTAK), grant number 3501 - 111M107.

8. References

Adenso-Díaz, B. and Rodríguez, F., A Simple Search Heuristic for the MCLP: Application to the Location of Ambulance Bases in a Rural Region, *Omega*, 1997, 25, 181-187.

Ağlamaz, E. and Özpeynirci, Ö., Pharmacy Duty Scheduling Problem with an Application to İzmir, *Technical Report*, 2011, İzmir University of Economics, Department of Logistics Management.

Alguwaizani, A., Hansen, P., Mladenović, N. and Ngai, E., Variable neighborhood search for harmonic means clustering, *Applied Mathematical Modelling*, 2011, 35, 2688-2694.

Alp, O., Erkut, E. and Drezner, D., An efficient genetic algorithm for the p-median problem, *Annals of Operations Research*, 2003, 122, 21-42.

Başar, A., Çatay, B. and Ünlüyurt, T., A taxonomy for emergency service station location problem, *Optimization Letters*, 2012, 6, 1147-1160.

Beasley, J.E., A Note on Solving Large p-Median Problems, *European Journal of Operational Research*, 1985, 21, 270-273.

Brimberg, J., Urošević, D. and Mladenović, N., Variable Neighborhood Search for the Vertex Weighted K-Cardinality Tree Problem, *European Journal of Operational Research*, 2006, 171, 74-84.

Carreras, M. and Serra, D., On Optimal Location with Threshold Requirements, *Socio-Economic Planning Sciences*, 1999, 32 (2), 91-103.

Ceyhan, G. and Özpeynirci, Ö., A Branch and Bound Algorithm for the Pharmacy Duty Scheduling Problem, *Technical Report*, 2013, İzmir University of Economics, Department of Logistics Management.

Crainic, T. G., Gendreau, M., Hansen, P. and Mladenović, N., Cooperative Parallel Variable Neighborhood Search for the p-median, *Journal of Heuristics*, 2004, 10, 293-314.

Çatay, B., Başar, A. and Ünlüyurt, T., İstanbul' da Acil Yardım İstasyonlarının Yerlerinin Planlanması, *Endüstri Mühendisliği Dergisi*, 2008, 19 (4), 20-35.

Fleszar, K. and Hindi, K. S., An effective VNS for the capacitated p-median problem, *European Journal of Operational Research*, 2008, 191, 612-622.

García-López, F., Melián-Batista, B., Moreno-Pérez, J. A. and Moreno-Vega, M., The parallel variable neighborhood search for the p-median problem, *Journal of Heuristics*, 2002, 8, 375-388.

Grujičić, I. and Stanimirović, Z., Variable neighborhood search method for optimizing the emergency service network of police special forces units, *Electronic Notes in Discrete Mathematics*, 2012, 39, 185-192.

Hale, S. T. and Moberg C. R., Location Science Research: A Review, *Annals of Operations Research*, 2003, 123, 21-35.

Hanafi, S., Lazić, J., Mladenović, N., Wilbaut, C. and Crévits, I., Hybrid Variable Neighborhood Decomposition Search for 0-1 Mixed Integer Programming Problem, *Electronic Notes in Discrete Mathematics*, 2010, 36, 883-890.

Hansen, P. and Jaumard, B., Cluster analysis and mathematical programming, *Mathematical Programming*, 1997, 79, 191-215.

Hansen, P. and Mladenović, N., Variable Neighborhood Search for the p-median, *Location Science*, 1997, 5 (4), 207-236.

Hansen, P., Mladenović, N. and Perez-Brito, D., Variable Neighborhood Decomposition Search, *Journal of Heuristics*, 2001, 7, 335-350.

Hansen, P. and Mladenović, N., Variable Neighborhood Search: Principles and Applicatins, *European Journal of Operational Research*, 2001, 130, 449-467.

Hansen, P., Mladenović, N. and Urošević, D., Variable neighborhood search and local branching, *Computers and Operations Research*, 2006, 33, 3034-3045.

Hansen, P., Oğuz, C. and Mladenović, N., Variable neighborhood search for minimum cost berth allocation, *European Journal of Operational Research*, 2008, 191, 636-649.

Harewood, S.I., Emergency Ambulance Deployment in Barbados: A Multi-Objective Approach, *Journal of the Operational Research Society*, 2002, 53, 185-192.

Jarboui, B., Derbel, H., Hanafi, S. and Mladenović, N., Variable neighborhood search for location routing, *Computers and Operations Research*, 2013, 40, 47-57.

Kariv, O. and Hakimi, S.L., An algorithmic approach to network location problems: Part 2. The p-medians, *SIAM, Journal on Applied Mathematics*, 1979, 37, 539-560.

Kochetov, Y., Alekseeva, E., Levanova, T. and Loresh, N., Large neighborhood search for the p-median problem, *Yugoslav Journal of Operations Research*, 2005, 15 (1), 53-63.

Lazić, J., Hanafi, S., Mladenović, N. and Urošević, D., Variable neighbourhood decomposition search for 0-1 mixed integer programs, *Computers and Operations Research*, 2010, 37, 1055-1067.

Levanova, T. and Loresh, M.A., Algorithms of ant system and simulated annealing for the p-median problem, *Automation and Remote Control*, 2004, 65, 431-438.

Mladenović, N. and Hansen, P., Variable Neighborhood Search, *Computers and Operations Research*, 1997, 24 (11), 1097-1100.

Mladenović, N., Brimberg, J., Hansen, P. and Moreno-Pérez, J. A., The p-median problem: A survey of metaheuristic approaches, *European Journal of Operational Research*, 2007, 179, 927-939.

Mladenović, N., Dražić, M., Kovačevic-Vujčić, V. and Ćangalović, M., General variable neighborhood search for the continuous optimization, *Computers and Operations Research*, 2008, 191, 753-770.

Mladenović, N., Urošević, D., Pérez-Brito, D. and García-González, C. G., Variable neighborhood search for bandwith reduction, *European Journal of Operational Research*, 2010, 200, 14-27.

Ng, B.T. and Han, J., Efficient and effective clustering methods for spatial data mining, In: Bocca, J. et al. (Eds.), Morgan, Kaufmann, *20th International Conference on Very Large Data Bases*, 1994, 144-155.

Pérez, J. A. M., Moreno-Vega, J. M. and Martín, I. R., Variable neighborhood tabu search and its application to the median cycle problem, *European Journal of Operational Research*, 2003, 151, 365-378.

Reinelt, G., TSPLIB: A Traveling Salesman Problem Library, *ORSA Journal on Computing*, 1991, 3, 376-384, http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

Resende, M. and Werneck, R.F., On the implementation of a swap-based local search procedure for the p-median problem, In: Ladner, Richard E. (Ed.), SIAM, Philadelphia, *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*, 2003, 119-127.

ReVelle, C.S., Eiselt, H.A. and Daskin, M.S., A bibliography for some fundamental problem categories in discrete location science, *European Journal of Operational Research*, 2008, 184, 817-848.

Şevkli, M. and Aydın, M. E., Variable Neighborhood Search for Job Shop Scheduling Problems, *Journal of Software*, 2006, 1 (2), 34-39.

Urošević, D., Brimberg, J. and Mladenović, N., Variable Neighborhood Decomposition Search for the Edge Weighted K-Cardinality Tree Problem, *Computers and Operations Research*, 2004, 31, 1205-1213.

Zhao, Q., Xie, C. and Xiao, Y., A variable neighborhood decomposition search algorithm for multilevel capacitated lot-sizing problems, *Electronic Notes in Discrete Mathematics*, 2012, 39, 129-135.

TÜBİTAK
PROJE ÖZET BİLGİ FORMU

Proje Yürütücüsü:	Yrd. Doç. Dr. NAİL ÖZGÜR ÖZPEYNİRCİ
Proje No:	111M107
Proje Başlığı:	Nöbetçi Eczane Çizelgeleme Problemleri
Proje Türü:	Kariyer
Proje Süresi:	30
Araştırmacılar:	
Danışmanlar:	
Projenin Yürütüldüğü Kuruluş ve Adresi:	İZMİR EKONOMİ Ü. İKTİSADİ VE İDARİ BİLİMLER F. LOJİSTİK YÖNETİMİ B.
Projenin Başlangıç ve Bitiş Tarihleri:	01/11/2011 - 01/05/2014
Onaylanan Bütçe:	100850.0
Harcanan Bütçe:	74486.5
Öz:	Bu proje kapsamında Eczane Nöbet Çizelgeleme problemleri üzerinde çalıştık. Problemin çeşitli türevlerini tanımladık ve her birisi için matematiksel programlama modellerini ortaya koyduk. Gerçek hayat probleminin hesaplama karmaşıklığını NP-Zor olarak belirledik. Probleme özgü kesin ve sezgisel çözüm yöntemleri geliştirdik. Kesin çözüm yöntemi olarak dal-sınır ve dal-fiyat algoritmaları, sezgisel yöntem olarak tabu arama ve değişken komşuluk arama algoritmaları önerdik. Probleme özgü alt sınır algoritmaları, alt problemlere özgü sezgiseller tasarladık. Problem boyutunu azaltmak için çeşitli yöntemler önerdik. Geliştirdiğimiz algoritmaların performanslarını test etmek amacıyla küçük ve büyük boyutlu rassal örnekler oluşturduk. Ayrıca İzmir iline ait gerçek verileri derleyerek, iki büyük boyutlu gerçek hayat örneği elde ettik. IBM ILOG CPLEX genel çözücüsü küçük boyutlu örneklerin bir kısmını çözebilmektedir. Dal-sınır ve dal-fiyat algoritmalarının temel hallerinin ve çeşitli türevlerinin performanslarını rassal örnekler üzerinde test ettik. Dal-sınır algoritması genel çözücü ile benzer performans göstermektedir. Dal-fiyat algoritması ise genel çözücüden daha iyi performans göstermekte ve onun çözemediği örnekleri çözebilmektedir. Büyük boyutlu örnekler ve gerçek hayat örnekleri kesin çözüm yöntemleri için fazla büyüktür. Bu sebeple, sezgisel yöntemler önerdik. Tabu arama ve değişken komşuluk arama yöntemleri ile kısa sürede olurlu çözümler elde ettik. Gerçekçi problemler için değişken komşuluk arama yöntemi ile alt sınır değerine %0,5?den daha yakın sonuçlar elde etmeyi başardık. Problemin ilk amacı olan talep ağırlıklı kat edilen yol miktarını en azlamaya ek olarak en az iş yükü oranına sahip eczanenin iş yükünü en çoklama amacını tanımladık. Bu iki amaçlı problemin tüm etkin sonuçlarını bulabilmek için kesin ve sezgisel yöntemler önerdik. Küçük boyutlu örneklerde matematiksel programlama ile tüm etkin sonuçları bulduk. Büyük boyutlu örnekler için değişken komşuluk arama yöntemi geliştirdik. Sezgisel yöntemin performansını süreye ek olarak etkin sonuçlar kümesini bulma başarısı ile değerlendirdik.
Anahtar Kelimeler:	Eczane nöbet çizelgeleme, dal-sınır, dal-fiyat, tabu arama, değişken komşuluk arama
Fikri Ürün Bildirim Formu Sunuldu Mu?:	Hayır

Projeden Yapılan Yayınlar:	1- Eczane Nöbet Çizelgeleme Problemi için Matematik Modeller (Bildiri), 2- Mathematical Models for Pharmacy Duty Scheduling Problems (Bildiri), 3- A Branch and Bound Algorithm for Pharmacy Duty Scheduling Problem (Bildiri), 4- Biobjective Pharmacy Duty Scheduling Problem (Bildiri), 5- A branch and bound algorithm for pharmacy duty scheduling problem (Bildiri)1- Eczane Nöbet Çizelgeleme Problemi için bir Değişken Komşuluk Arama Sezgiseli (Bildiri - Uluslararası Bildiri - Sözlü Sunum), 2- A Branch and Bound Algorithm for Pharmacy Duty Scheduling Problem (Bildiri - Uluslararası Bildiri - Sözlü Sunum), 3- A branch and bound algorithm for pharmacy duty scheduling problem (Bildiri - Uluslararası Bildiri - Sözlü Sunum), 4- Variable Neighborhood Search for the Pharmacy Duty Scheduling Problem (Makale - Diğer Hakemli Makale),