T.C
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NİCEL DEĞERLİ VERİ KÜMELERİNDEN SIRALI ÖRÜNTÜLERİN ÇIKARILMASI İÇİN FP-GROWTH TABANLI BİR YÖNTEM

YÜKSEK LİSANS TEZİ

A.BAHADIR KARLİ

Anabilim Dalı: Bilgisayar Mühendisliği

Programı: Kuramsal Temeller

Tez Danışmanı: Doç. Dr. Mehmet KAYA

Tezin Enstitüye Verildiği Tarih: 08 Şubat 2010

ŞUBAT-2010

T.C
FIRAT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

NİCEL DEĞERLİ VERİ KÜMELERİNDEN SIRALI ÖRÜNTÜLERİN ÇIKARILMASI İÇİN FP-GROWTH TABANLI BİR YÖNTEM

YÜKEK LİSANS TEZİ

A. Bahadır KARLİ

07129104

Tezin Enstitüye Verildiği Tarih : 08 Şubat 2010
Tezin Savunulduğu Tarih : 25 Şubat 2010

Tez Danışmanı : Doç. Dr. Mehmet KAYA (F.Ü)
Diğer Jüri Üyeleri : Doç. Dr. İbrahim TÜRKOĞLU (F.Ü)
Yrd. Doç. Dr. Burhan ERGEN (F.Ü)

ŞUBAT- 2010

ÖNSÖZ

Bu tez çalışmam boyunca, ilgi ve yardımlarını esirgemeyen danışman hocam Doç. Dr. Mehmet KAYA hocama, teşvik ve desteklerinden dolayı eşime ve kızlarıma ve bu tez çalışması için proje desteği sağlayan FÜBAP'a teşekkürlerimi sunarım. Ayrıca Dicle Üniversitesi Tıp Fakültesinden Prof. Dr. Mustafa ALDEMİR hocama, Doç. Dr. Beran YOKUŞ hocama ve Evren GİDİCİ' ye ve tez süreci boyunca yokluğumu aratmayan ve beni destekleyen mesai arkadaşlarıma da teşekkürlerimi sunarım.

Ahmet Bahadır KARLİ
ELAZIĞ - 2010

İÇİNDEKİLER

Sayfa No

ÖZET

Sıralı örüntüler zaman damgası ile sıralanmış bir veritabanında yaygın öğeler kümesinin bir başka yaygın öğe kümesi tarafından izlenmesidir. Sıralı örüntülerin keşfedilmesi için şimdiye kadar birçok algoritma önerilmiştir. Bu algoritmalar içinde en etkili olanlardan biri de FP-Growth algoritması kullanarak bu örüntülerin elde edilmesidir. Buna karşılık FP-Growth yaklaşımı, karmaşık veri yapısı kullanması ve alt ağaçların oluşturulması için özyineli bazı işlemlere gerek duyması gibi önemli dezavantajlar da içermektedir. Dahası mevcut FP-Growth yaklaşımlarında veri tabanı hep ikili veri kümesi olarak ele alınmıştır. Halbuki gerçek hayattaki veritabanlarının büyük çoğunluğu nicel değerli öğe kümelerinden oluşmaktadır.

Bu tezde, yukarıda belirtilen problemlerin üstesinden gelebilmek için öncelikle FP-Growth algoritması gibi düşünen fakat FP-ağacı oluşturulurken özyineleme yerine öğe tabanlı aday küme üretimi kullanan yeni bir yöntem önerilmiştir. Bu yöntem daha sonra nicel değerli veritabanlarında sıralı örüntülerin bulunması için uyarlanmıştır.

Dicle Üniversitesi Tıp Fakültesi Merkez laboratuarındaki verilerden elde edilen sonuçlar yöntemin uygulanabilirliğini ve klasik FP-Growth algoritmasına olan üstünlüğünü göstermektedir.

Anahtar Kelimeler: Veri Madenciliği, Sıralı Örüntüler, FP-Growth algoritması

SUMMARY

A FP-Growth Based Method for Extraction of Sequential Patterns From Quantitative Databases

The problem of discovering sequential patterns is to find inter-transaction patterns such that the presence of a set of items is followed by another item in the time-stamp ordered transaction set. So far, many methods have been proposed to discover the sequential patterns. One of the effective approaches within them is to extract the patterns by using FP-Growth. However, FP-Growth approach contains important disadvantages; including the use of complex data structures and the requirement of some recursive processes to extract sub-trees. Moreover, in the past, many algorithms were proposed for mining sequential patterns, most of which were based on items with binary value. Transactions with quantitative values are, however, commonly seen in real-world applications.

In this thesis, in order to overcome the problems mentioned above, first, using item-based candidate generation instead of recursive processes, a FP-growth based novel approach is proposed. This method is then adapted to find sequential pattern in databases with quantitative item. The experimental results conducted on real data set of Dicle University, Faculty of Medicine, Center Laboratory show the applicability and the superiority of the proposed method.

Keywords: Data Mining, Sequential Patterns, FP-Growth algorithm

ŞEKİLLER LİSTESİ

Sayfa No

TABLOLAR LİSTESİ

Sayfa No

Sayfa No

1. GİRİŞ

Veri madenciliği ve bulanık küme teorisi, bir çok araştırmaya tabi tutulan ve iş dünyası, bilim, ekonomi, mühendislik, tıp gibi bir çok alanda uygulanmaya başlanan ve etkili ve başarılı sonuçlar elde edilen uygulamalardır [1-3].

Veri madenciliğinde anlamlı, ilginç, işe yarar kuralların ve örüntülerin ortaya çıkarılması için bir çok çalışma yapılmıştır [4-11]. Veri madenciliğinde ilişki kurallarının ortaya çıkarılması için öncelikle veritabanından bütün yaygın öğe kümelerinin ortaya çıkarılması gerekir. Yaygın öğe kümeleri, veri madenciliğinde, sıralı örüntüler, ilişki kuralları, korelasyonlar, çok boyutlu örüntüler gibi ilginç motifleri bulmak için önemli bir rol oynar [12,13]. Bu yaygın öğe kümelerinin ortaya çıkarılması için yapılan çalışmalarda temel olarak iki yaklaşım mevcuttur. Agrawal [5] tarafından geliştirilen Apriori algoritması ve Han [4,14] tarafından öne atılan FP-Growth yaklaşımıdır. Apriori algoritması, aday kümeler üretme ve test yaklaşımını benimser. Çok sayıda aday küme üretme ihtiyacı duyar ve her aday küme üretimi için de veri kümesini tarar. FP-Growth yaklaşımı ise aday küme üretmez. Yaygın örüntüleri bulmak için Fp-tree ağaç veri yapısını kullanır. Algoritma, Fp-tree ağaç veri yapısını oluşturduktan sonra, bu ağacı alt ağaçlara bölerek ve her alt ağacın kendisini de öz yineli olarak dolaşarak yaygın örüntüleri bulmaya çalışır. FP-Growth algoritması, Apriori algoritmasının dezavantajlarını ortadan kaldırmasına rağmen, bu yaklaşımın kendisi de, uygulanabilirlik açısından bazı dezavantajlara sahiptir.

Sıralı örüntüler, sıklıkla görülen ve birbiri ardına meydana gelen, ardışık olaylar veya alt dizilerdir[15,16]. Gerçek dünyada, birçok uygulama alanında olan nicel değerli veriler ve sıralı örüntüler sıklıkla karşımıza çıkmaktadır. Sıralı örüntülerde amaç bir sonraki davranışın, olayın büyük bir olasılıkla tahmin edilmesidir.

1.1. Tezin Amacı

Sıralı örüntülerin bulunması için, öncelikle yaygın örüntülerin bulunması gerekir. Yaygın örüntülerin ortaya çıkarılması için önerilen FP-Growth [4,14] yaklaşımı, Apriori algoritmasına göre, çok maliyetli olan veritabanının defalarca taranması ve çok sayıda aday üretme ve test etme sakıncalarını ortadan kaldırır. Bunun yanında Apriori algoritmasına göre sıralı örüntüleri bulurken daha hızlı çalışmasına ve daha etkili olmasına rağmen;

karmaşık bir veri yapısı kullanması, Apriori [5] ile kıyaslandığında veri yapısını oluşturmanın zor olması ve öğe koşullu alt ağaçların oluşturulup öz yineli işlemlere tabi tutulması gibi bazı dezavantajlara sahiptir. Bu tezde bu dezavantajları ortadan kaldırmak ve uygulanabilirliğini artırmak için farklı bir FP-Growth algoritması önerilmiştir. Önerilen FP-Growth algoritmasında kullanılacak olan veri yapıları veritabanı üzerinde oluşturularak, özyineleme yerine öğe tabanlı aday küme üretimi kullanılmıştır. Son aşamada sıralı örüntüler için bulanık küme teorisinden faydalanılmıştır.

Nicel değerli verileri modellemede kullanılacak olan bulanık küme teorisi, dilsel terimlerle ifade edilen üyelik fonksiyonlarını kullanır. Üyelik fonksiyonları çeşitli şekillerde olmakla beraber en çok kullanılan üçgensel üyelik fonksiyonlarıdır [17-19].

Bu maksatla, Dicle Üniversitesi Tıp Fakültesi merkez laboratuarından alınan verileri kullanarak, sıralı örüntülerin elde edilmesi ve bir sonraki olası hastalıkların belirlenmesine çalışılmıştır.

1.2. Tezin Yapısı

Bu tez çalışmasında bundan sonraki bölümler 5 ana kısma ayrılmıştır. İkinci bölümde veri madenciliğinin temel kavramları verilmiş ve veri madenciliği yöntemlerinden biri olan yaygın örüntülerin FP-Growth algoritması ile çıkarılma süreci verilmiştir. Üçüncü bölümde ise çok öğeli ve nicel değerli veritabanlarından sıralı örüntülerin çıkarılması için ilk adım olan yaygın örüntülerin ortaya çıkarılması amacıyla geliştirilen yeni bir FP-Growth algoritması önerilmiştir. Dördüncü bölümde Bulanık Küme Teorisinden faydalanılarak sıralı örüntülerin elde edilme süreci anlatılmıştır. Beşinci bölümde ise önerilen yaklaşım, Dicle Üniversitesi Tıp Fakültesi merkez laboratuarından alınan gerçek veri kümesine uygulanması anlatılmış ve elde edilen sonuçlar verilmiştir. Altıncı ve son bölümde ise sonuçlar ve sonraki çalışmalar için önerilerde bulunulmuştur.

2. VERİ MADENCİLİĞİ ve SIRALI ÖRÜNTÜLER

2.1. Veri Madenciliği

Bilgiyi madenleme işi olarak ta bilinen veri madenciliği, veritabanlarında tutulan ve veritabanları içerisinde gömülü olan geniş veri yığınları içerisinden, saçma olmayan, anlamlı ve önceden tahmin edilmeyen örüntülerin ortaya çıkarılmasıdır [6,12]. Veri madenciliği teknikleri, mühendislik, tıp, finans gibi bir çok alana efektif olarak uygulanabilmektedir [1-3, 20].

Aslında bir süreç olan bilgiyi madenleme işinde veri madenciliği bu sürecin önemli bir parçası olmakla beraber, günümüzde bu sürecin tamamı veri madenciliği olarak bilinmektedir. Veri madenciliği süreci aşağıdaki adımlardan oluşur:

- Veri temizleme (tutarsız verileri elemek için)
- Veri entegrasyonu (gerekli olması halinde birden çok kaynaktaki verilerin birleştirilmesi)
- Veri seçme (veri tabanından çekilen analiz ile ilgili veri)
- Veri dönüşüm (madencilik için verinin uygun forma dönüştürülmesi)
- Veri madenciliği (örüntülerin ortaya çıkarılması için zorunlu işlem aşaması)
- Örüntü değerlendirme
- Bilgi sunumu[6]

Genel olarak, veriden keşfedilebilen bir çok örüntü çeşidi vardır :

- İlişki kuralı
- Sıralı örüntüler
- Sınıflandırma
- Kümeleme[6]

Bu tez çalışmasında kullanılan bazı terimlerin tanımlamaları aşağıda verilmiştir.

Sık tekrarlanan örüntüler;

Bir veri kümesinde görülen ve frekans değeri kullanıcı tanımlı eşik değerinden daha düşük olmayan öğe gruplarıdır.

Sık tekrarlanan öğe grubu;

Süt ve ekmek gibi bir öğe kümesinin bir veritabanında birlikte sık sık görülmesi bir yaygın öğe grubuna örnektir.

Sıralı örüntüler;

Sıklıkla görülen birbiri ardına meydana gelen, ardışık olaylar veya alt dizilerdir. Dijital kamera alan kişinin bir ay içinde muhtemelen bir yazıcıyı alması sıralı örüntüye bir örnektir. Sıralı örüntüler, birbiri ardına gelişen olayların meydana gelme arasındaki ilişkilerdir. Sıralı örüntüler için zaman damgası her veri kümesi için önemli bir niteliktir.

Bulanık Küme teorisi;

Bulanık küme teorisi, işlemlerdeki nicel değerleri dilsel terimlere dönüştürür, daha sonra ilişki kuralı bulmak için onları eler. Bir çok uygulama alanı bulunan bulanık küme teorisi, basit ve insanoğlunun akıl yürütmesine benzeştiği için gittikçe daha fazla akıllı sistemlerde kullanılmaya başlanmıştır[17-19].

FP-tree (sık tekrarlanan örüntü ağacı) ;

FP-Growth algoritmasının kullandığı bir ağaç veri yapısıdır.

2.2. Örüntülerin Ortaya Çıkarılmasında Ele Alınan Temel Yaklaşımlar

2.2.1. Apriori Algoritması

Örüntülerin ortaya çıkarılması için yapılan bir çok çalışma -*Apriori* algoritması gibi- aday kümeler üretme ve test etme yaklaşımını benimsemiştir. Aday kümeler üretme ve test etme yaklaşımının temel ilkesi şudur: "*eğer bir öğe grubu yaygınsa o zaman bu öğe grubunun bütün alt kümeleri de yaygındır.* " [5,21] Örneğin {c,d,e} yaygın öğe grubu olsun. O zaman bu yaygın öğe grubunun alt kümeleri olan {c,d}, {c,e}, {d,e}, {c}, {d}, {e} kümeleri de yaygın olmalıdır. Bu yaygın öğe gruplarını bulmak için veri kümesi taranır ve bunların alt kümelerinin yaygın olup olmadığını anlamak için de veri kümesi defalarca taranır. Bu yaklaşım ilk önce 1 uzunluklu yaygın öğe gruplarını daha sonra 2 uzunluklu sonra 3 uzunluklu vb. yaygın öğe gruplarını bulur. Bu araştırmanın her aşaması için veri kümesi baştanbaşa taranır. Aday küme üretimi ve olası her kuralı test etmek için

veri kümesi her seferinde baştanbaşa taranır. Ancak bu algoritma üç temel noktada sıkıntı vermektedir :

- Herşeyden önce bu algoritma büyük sayıda aday üretme ihtiyacı duymaktadır.
- Defalarca veri kümesini tarar ve geniş sayıdaki aday kümelerini örüntü karşılaştırması yoluyla test eder. Bu da veritabanı için çok yüksek maliyetli işlemlerdir.
- Veritabanındaki verilerin değişmesi durumunda bu işlemlerin hepsinin baştan tekrar edilmesi demektir.

2.2.2. FP-Growth (sık tekrarlanan örüntüleri geliştirme) Algoritması

Büyük ve yoğun veri kümelerinde etkili olan, FP-tree ağaç yapısına dayanan, aday kümeler üretmeden yaygın öğeler grubunu oluşturmak için kullanılan bir algoritmadır[4,14]. Bu algoritma kısaca, böl ve yönet mantığını kullanarak, problemi alt problemlere bölerek çözmeye çalışır.

FP-tree Ağaç Yapısının Biçimsel Tanımlaması ;

- Oluşturulacak olan bir FP-Tree ağaç yapısının içeriği ;
    1. null olarak adlandırılan ve değeri null (boş) olarak atanan bir kök düğüm,
    2. bu kök düğümün çocukları olarak bir ön ekli alt ağaç öğe kümesi,
    3. ve bir frekans öğe başlık tablosundan oluşur.
- Bu ağaçta yer alan her düğüm üç bilgi alanından oluşur :
    1. Öğe adı; kayıt okunurken elde edilen ad, aynı zamanda düğüme verilen ad (düğümün etiketi),
    2. Bu öğeye ait bir sayaç; düğüm olarak atanmış ve okunan yeni kayıtta bu düğümün üzerinden her geçmek gerektiğinde bir artırılan sayaç,
    3. Düğüm bağlantı bilgisi ; FP-tree yapısında bu düğümden bir sonraki düğümü gösteren bilgi.
- Frekans öğe başlık tablosunda yer alan her kayıt iki bilgi alanından oluşur :
    1. Öğe adı,
    2. Kendi adıyla örtüşen FP-tree yapısında yer alan kendisine ait ilk düğümü gösteren bilgi.

FP-Growth Algoritması ve FP-tree Yapısının Avantajları;

Bu veri yapısı ve bu yaklaşım özellikle üç noktada bizi başarıya ulaştırır :

- Herşeyden önce FP-tree yapısı, geniş veritabanlarına göre sıkıştırılmış, özetlenmiş ve yoğunlaştırılmış daha küçük bir veri yapısı elde etmeyi sağlar. Bu yapı veritabanının tekrar tekrar taranmasını önler ve maliyeti azaltır
- FP-Tree yapısına dayanan çalışmalar parçalı örüntü geliştirmeyi benimser ki bu da geniş sayıda aday üretme maliyetlerinin sakıncalarını ortadan kaldırır
- Bu yapı ve bu yapıya dayanan metot daha sonra detayları verilecek olan böl-yönet yaklaşımını benimsediği ve öğe koşullu alt ağaçlar oluşturduğu için, sıralı örüntüleri geliştirirken yapılacak arama sınırlarını çok ciddi bir şekilde azaltır.

FP-Growth Algoritmasının Çalışması : FP-tree Ağacının Oluşturulması ve Yaygın Öğelerin Çıkarılması

Her şeyden önce bu algoritma veritabanını sadece iki kez tarar. İlk taramayı daha sonra FP-tree yapısını oluşturmak amacıyla kullanılacak olan yaygın öğeleri ve bunların frekanslarını bulmak amacıyla yapar. Frekanslarıyla birlikte bulunan bu yaygın öğeler frekanslarına göre yukarıdan aşağıya sıralanırlar. Bu sıralamadan sonra minimum destek sayısının altında kalan öğeler elenir.

İkinci taramayı FP-tree yapısını oluşturmak için yapar. FP-tree yapısını oluşturmak için ilk başta null olarak etiketlendirilen ve değeri boş olan kök düğüm oluşturulur. Bundan sonra veritabanından okunan her bir kayıt satırındaki sık tekrarlanan öğe grupları içindeki öğeler için oluşturulacak olan düğümler bu kök düğümün altında yer alacak şekilde oluşturulur. Bu düğümler oluşturulurken veritabanındaki her bir kayıt tek tek ele alınır ve her bir kayıt kendi içinde daha önce ilk taramada oluşturulan öğelerin frekans tablosundaki frekans sayısına göre yukarıdan aşağıya sıralanır. Sıralama bu şekilde yapıldığı zaman, sıralanmış kayıtlar ilk öğe olarak aynı sık tekrarlanan öğe içerecektir.

Bu yapıyı oluşturmak için temel işlem; her bir kayıt içindeki frekans sıralamasını ve gerekiyorsa her bir öğe için kök düğümün altında bir düğüm oluşturmayı gerektirir. Bu yapıda her öğe için ihtiyaç duyulmuyorsa ayrı bir düğüm oluşturmaya gerek yoktur. Eğer bir öğenin orijini daha önce oluşturulan bir öğeye dayanıyorsa bu dikkate alınır. Yani veritabanından kayıtlar okunup bu kayıtlardaki yaygın öğeler FP-tree yapısına eklendikçe ortak önekler ortaya çıkacaktır. Örneğin veritabanındaki {A, B, C} kaydını ele alalım.

Daha önce {A, B, D} kaydının ele alındığını ve bu {A, B, D} kaydı için düğümlerin oluşturulduğunu varsayalım. Bu durumda {A, B, C} kaydı ele alınırken daha önce oluşturulan {A, B, D} kaydından dolayı {A, B} ortak ön ek olacaktır. Dolayısıyla {A, B, C} kaydı için FP-tree yapısı üzerinde A ve B için ayrı bir düğüm oluşturulmayacaktır. Sadece A ve B düğümlerinin sayacı bir artırılacaktır ve aynı düzeyde C düğümü oluşturulacaktır. Bu arada şuna da dikkat etmek gerekir; bu yapı oluşturulurken minimum destek sayısı dikkate alınarak yaygın olmadığına karar verilen öğeler dikkate alınmayacaktır. Yani bu yapıda yaygın olmayan öğeler yer almayacaktır. Buna ek olarak FP-tree yapısından başka bir bağlı liste (linked list) oluşturulabilir. Bu liste, örneğin tekrar edilen A düğümünü daha önce oluşturulmuş olan bütün diğer A düğümlerine bağlamak için kullanılabilir.

Bu FP-tree yapısının oluşturulma süreci aşağıdaki gibi özetlenebilir :

- Veri tabanını bir kere tara.
    1. Yaygın öğeler kümesi F’ yi öğelerin frekanslarıyla birlikte bir araya getir.
    2. F kümesini öğelerin frekanslarına göre yukarıdan aşağıya doğru sırala
    3. Yaygın öğeler listesi L’ yi oluştur.
- FP-tree yapısının kök düğümü olan ve null olarak etiketlendirilen T’yi oluştur. Veritabanında yer alan her işlem satırı için şunları yap :
    1. L listesindeki sıralamayı baz alarak, her işlem kaydındaki öğeleri seç ve L’ ye göre sırala.
    2. Elde edilen bu sıralanmış frekans öğe listesindeki ilk öğeden başlayarak FP-tree yapısına ekle

Bu yapının inşa edilmesinden sonra FP-Growth algoritmasının bu yapıdan faydalanarak yaygın örüntüleri nasıl çıkaracağı aşağıdaki verilmiştir;

- Bir koşullu yollar kümesi türet. Bunlar FP-tree’den elde edilen sonekli örüntülerdir.
- Koşullu yollar için bir koşullu FP-tree oluştur (öğe koşullu alt ağaçlar)
- Yaygın örüntüler bulmak ve bulunan her örüntünün destek seviyesini belirlemek için koşullu ağacı özyineli olarak dolaş

Dikkat edilirse bu yapıda, yaygın olmayan öğeler bu yapı oluşturulmadan önce elendiği için, sadece minimum destek sayısını aşmış olan öğeler yer alır. Yani yaygın olmayan öğeler için boşa zaman kaybedilmez.

Bu açıklamaya ek olarak; frekansı en yüksek yani en yaygın öğeler ağaçta en tepede ya da kök düğümüne en yakın öğeler olduğundan, FP-Growth algoritması geçerli olan yaygın örüntülerin büyük çoğunluğunu aramanın en erken zamanlarında bulur. Geniş veritabanlarında çalışıldığı göz önüne alınırsa; FP-Growth algoritmasının ölçeklenebilirliği Apriori algoritmasından [5] çok daha iyi olduğundan kısa ve uzun örüntüleri kullanmak için FP-tree iyi yöntemdir. Zaten minimum eşik değeri aşağıları doğru çekildikçe bu durum daha aşikar bir hal alır.

FP-tree, Öğe Koşullu FP-Tree Oluşturulması ve Örüntü Çıkarılması

FP-tree ağacının oluşturulması, FP-Growth algoritmasının bu ağaç yapısına dayanarak oluşturduğu öğe koşullu alt ağaçlar için detaylı bir uygulama aşağıda verilmiştir:

Bir FP-tree, giriş verisinin sıkıştırılmış, yoğunlaştırılmış gösterimidir. FP-tree veri kümesinden her seferinde bir işlem okuyarak ve her işlemi bir yol şeklinde FP-tree ağacına entegre ederek kurulur. Farklı işlem satırları birçok ortak öğeye sahip olabileceğinden, bunların yolu daha önce FP-tree ağacında kurulan yolun üzerinden geçebilir.

Tablo 2.1. Örnek işlem veri kümesi [22].

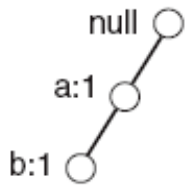
Harek et ID	Öğeler (items)
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Tablo 2.1, on işlem satırı ve beş öğeden oluşan bir işlem veri kümesini göstermektedir. Bu veri kümesi üzerinde yapılan işlemler sırasıyla aşağıdaki gibidir[22].

- Veri kümesi, destek sayılarıyla birlikte her bir öğenin belirlenmesi için bir kez taranır. Bu taramadan sonra frekanslarıyla birlikte elde edilen öğeler frekanslarına göre yukarıdan aşağıya sıralanırken yaygın olmayan öğeler elenir. Bu işlem veri

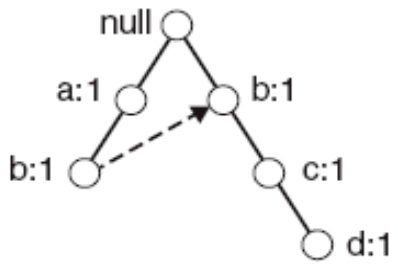
kümesine göre frekansı en yüksek, en sık tekrarlanan öğe *a*'dır. Daha sonra *a* öğesinin ardından *b,c,d* ve *e* öğesi gelir.

- Algoritma FP-tree ağacını oluşturmak için örnek veri kümesini ikince kez tarar. Başlangıçta FP-tree ağacı null ile ifade edilen bir kök düğümünü içerir. Veri kümesinin ilk satırı olan {a,b} okunduktan sonra *a* ve *b* olarak etiketlendirilen *a* ve *b* düğümleri FP-tree ağacına eklenir.Bu işlem satırının yolu şu şekilde biçimlenir : *null* → *a* → *b* FP-tree üzerinde oluşturulan bu yolun üzerindeki her bir düğümün başlangıçtaki sayaç değeri 1 olur



Şekil 2.1. Hareket ID = 1 işlem satırı okunduktan sonraki FP-tree ağacının durumu

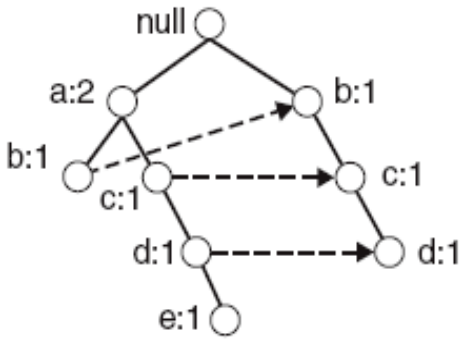
- İkinci işlem satırı olan {b,c,d} okunduktan sonra, yeni bir düğüm kümesi olan *b,c ve d* öğeleri düğümler oluşturulur. Bu yeni düğümlerin bağlanmasından sonra işlem satırının yolu şu şekilde biçimlenir : *null* → *b* → *c* → *d* bu düğümlerin sayaç değeri bu aşamada 1 dir. Dikkat edilirse her iki işlem satırında *b* ortak öğe olmasın rağmen, işlem satırlarının önekleri ortak olmadığı için her iki işlem satırının yolu da ayrı olmuştur.



Şekil 2.2. Hareket ID = 2 işlem satırı okunduktan sonraki FP-tree ağacının durumu

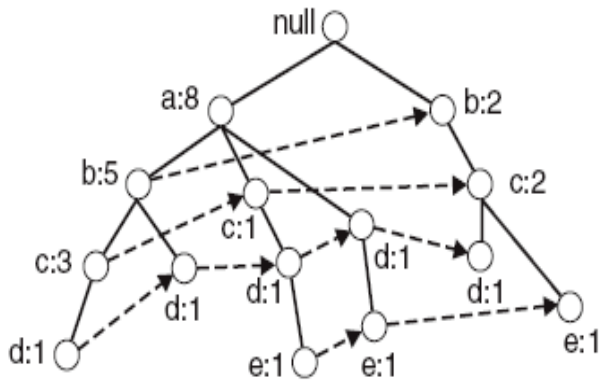
- Üçüncü işlem satırı olan {a,c,d,e}, birinci işlem satırı olan {a,b} ile ortak önek olan *a* öğesini paylaşırlar. Sonuç olarak, üçüncü işlem satırının yolu, *null* → *a* → *c* → *d* → *e*, ilk işlem satırının yolu *null* → *a* → *b*, ile örtüşür. İki işlem satırının yolu *a*

ortak öğeden dolayı örtüştüğü için, *c, d ve e* öğeleri için sayaç değeri bir olacak şekilde düğümler oluşturulurken, *a* öğesine ait olan düğümün sayaç değeri bir artarak iki olur.



Şekil 2.3. Hareket ID = 3 işlem satırı okunduktan sonraki FP-tree ağacının durumu

FP-tree ağacının oluşturulması işlemleri, veri kümesindeki bütün işlem satırları yukarıdaki gibi ele alınarak devam ettirilir ve bu işlemlerin sonucunda FP-tree ağacı oluşturulur. Örnek olarak ele alınan veri kümesinden işlemler bittikten sonra elde edilen FP-tree ağacının çizelge gösterimi şu şekilde olur.

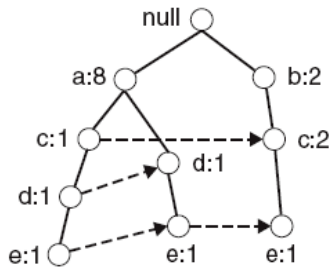


Şekil 2.4. Hareket ID = 10 işlem satırı okunduktan sonraki FP-tree ağacının durumu [22].

Bir FP-tree, aynı zamanda, ağaç üzerindeki aynı düğümlerin kendi aralarındaki bağlantıyı gösteren bir gösterge listesine de sahiptir. Bu göstergeler hem Şekil 2.4.' te kesikli çizgilerle gösterilmiştir. Bu göstergeler, ağaç içinde öğelerin bireysel olarak ele

alınıp hızlı bir şekilde kendi kendilerine erişme kolaylığı sağlama konusunda bize yardımcı olur.

FP-Growth algoritması, bir FP-tree ağacını aşağıdan yukarıya doğru bir mantıkla dolaşarak yaygın öğe gruplarını üretir. Verilmiş olan örnekte gösterilen ağaç yapısını göz önüne alırsak ; FP-Growth algoritması ilk olarak *e* öğesi ile biten, ardından *d, c, b ve* son olarak ta *a* ile biten yaygın öğe gruplarına bakar. Bu yolların açılımı aşağıdaki Şekil 2.5' te *e* ile biten ağaç gösterildiği gibi olur.



Şekil 2.5. *e* düğümü içeren yollar

e ile biten yaygın öğe grubunun bulunmasından sonra, algoritma *d* ile biten yaygın öğe grubuna bakar yani *d* ile ilişkili yollara bakar. Bu işlemler, düğüm *c*, *b*, ve en son düğüm *a* ile ilişkili bütün yollar ortaya çıkarılıncaya kadar devam eder. Bunlarla ilgili yaygın öğe gruplarının özetlenmiş durumu aşağıdaki Tablo 2.2' de gösterilmiştir.

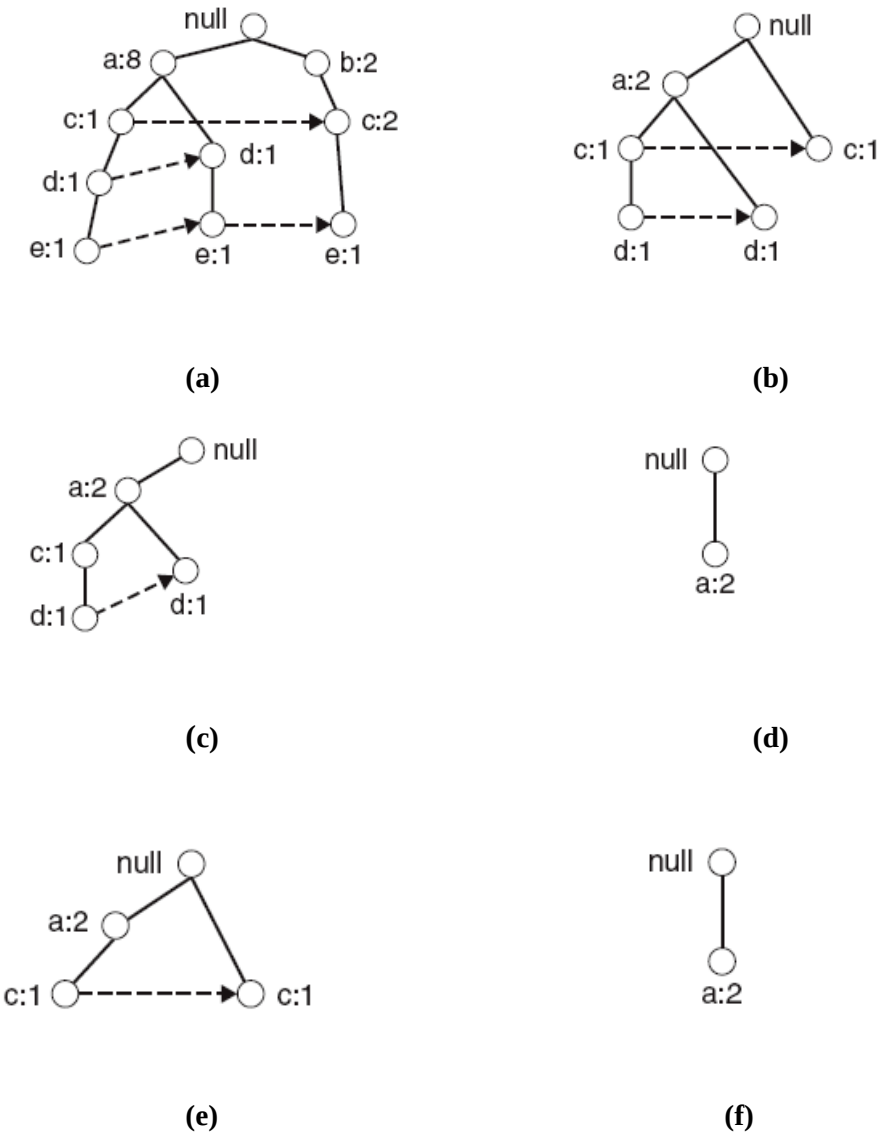
Tablo 2.2. Yaygın öğe gruplarının sıralanmış listesi [22].

Sonek	Yaygın öğe grupları
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

FP-Growth algoritması bu şekilde bütün yaygın öğe gruplarını bulurken dayandığı strateji daha önce de belirtildiği gibi böl-yönet stratejisidir. Bu stratejinin amacı problemi daha küçük alt problemlere bölmektir. Örneğin, *e* ile biten bütün yaygın öğe gruplarının bulunulmasına çalışalım. Bunu yapmak için, ilk olarak {e} öğe grubunun kendisinin yaygın olup olmadığı kontrol edilmelidir. Eğer yaygınsa, bunun alt problemlerine geçilir.

Yani sonu *de,* arkasından *ce, be* ve *ae* ile biten yaygın öğe gruplarının bulunmasına yönelik alt problemler ele alınır. Alt problemlerden elde edilen çözümlerin birleştirilmesiyle, sonu *e* ile biten bütün yaygın öğe grupları bulunabilir. Bu böl-yönet yaklaşımı FP-Growth algoritması tarafından kullanılan anahtar stratejidir.

Alt problemlerin çözümü için aşağıda bir örnek verilmiştir. *e* ile sonlanan yaygın öğe gruplarının bulunulmasına çalışılmıştır.



Şekil 2.6. FP-Growth uygulamasıyla, *e* ile sonuçlanan yaygın öğe gruplarının bulunması
(a) önek yollar, e ile sonuçlanan, **(b)** e için koşullu FP-tree,
(c) önek yollar, de ile sonuçlanan, **(d)** de için koşullu FP-tree
(e) önek yollar, ce ile sonuçlanan, **(f)** önek yollar ae ile sonuçlanan

- İlk adım, düğüm *e* içeren bütün yolların bir araya getirilmesidir. Bunlar önek yollar
- olarak adlandırılan ve Şekil 2.6. (a)‘ da gösterilen başlangıç yollardır.
- Şekil 2.6.(a)’ dan düğüm *e* ile ilişkili destek sayıları toplanarak *e* için destek sayısı bulunur. Minimum destek sayısının 2 olduğunu kabul edelim. Bu durumda, destek sayısı 3 olduğundan {e} yaygın öğe grubudur.
- {e} yaygın olduğundan, algoritma, *de, ce, be* ve *ae* ile sonuçlanan yaygın öğe gruplarının bulunmasına yönelik alt problemleri çözmek zorundadır. Bu alt problemleri çözmeden önce, algoritma, önek yolları **Koşullu FP-tree** yapısına çevirmelidir. Koşullu FP-tree, yapısal olarak FP-tree ağacına benzer, aralarındaki tek fark koşullu FP-tree ağacının belli bir sonek ile sonuçlanan yaygın öğe gruplarını bulmak için kullanılmasıdır. Bir koşullu FP-tree şu şekilde elde edilir :

    1. İlk olarak, destek sayısı önek yollar boyunca güncellenmelidir çünkü bazı sayaçlar *e* öğesini içermeyen işlem satırlarını kapsar. Örneğin, şekil 2.6. (a)’ nın en sağındaki yol, null → b:2 → c:2 → e:1, *e* öğesini içermeyen bir işlem {*b,c*} içerir. {*b,c,e*} içeren işlemin gerçek sayısını yansıtmak için önek yol boyunca sayaçlar 1 ‘e ayarlanmalıdır.
    2. Önek yollar, *e* düğümlerinin atılmasıyla budanır. Bu düğümler atılabilir çünkü sadece *e* içeren işlemleri yansıtmak amacıyla destek sayıları önek yol boyunca zaten güncellenmiş durumdadır ve *de, ce, be,* ve *ae* ile sonuçlanan yaygın öğe gruplarının bulunmasına yönelik alt problemlerin artık düğüm *e* hakkında bilgiye ihtiyacı yoktur.
    3. Önek yollar boyunca destek sayılarının güncellenmesinden sonra, artık bazı öğeler yaygın olmayacaktır. Örneğin, düğüm *b* sadece bir kez görünür ve destek sayısı 1’e eşit olur. Bunun anlamı şudur ; demek ki *b* ve *e*’ yi birlikte içeren sadece bir tane işlem vardır. Öğe *b* sonraki analizde güvenli bir şekilde ihmal edilebilir çünkü *be* ile sonuçlanan bütün öğe grupları yaygın olmamalıdır.

e için koşullu FP-tree Şekil 2.6.(b)’ de gösterilmiştir. Frekans sayısı güncellenmiş ve düğüm *b* ve *e* elenmiş olduğu için, ağaç, orijinal önek yoldan farklı görünür.

- FP-Growth, *e* ile ilgili, *de, ce* ve *be* ile sonuçlanan yaygın öğe gruplarının bulunmasına yönelik alt problemlerin çözümü için koşullu FP-tree kullanır. d*e* ile sonuçlanan yaygın öğe gruplarını bulmak için, *d* için önek yollar, *e* için oluşturulan

koşullu FP-tree kullanılarak bir araya getirilir. (Şekil 2.6.c) Düğüm *d* ile ilişkili frekans sayılarının toplanmasıyla, {d,e} için destek sayısını elde ederiz. Destek sayısı 2' ye eşit olduğundan, {d,e} bir yaygın öğe grubudur. Sonra, algoritma adım 3'te açıklanan yaklaşımı kullanarak, *de* için koşullu FP-tree yapısını inşa eder. Destek sayılarının güncellenmesinden ve yaygın olmayan öğe *c'* nin atılmasından sonra, *de* için koşullu FP-tree Şekil 2.6.(d)' de gösterildiği gibi olur. Koşullu FP-tree sadece bir öğe içerdiğinden, *a*, kendisinin destek sayısı minimum destek sayısına eşittir, algoritma, {*a,d,e*} yaygın öğe grubunu çıkarır ve *ce* yaygın öğe grubunu üretmek için diğer alt probleme yönelir. *c* için önek yollar işlendikten sonra, sadece {*c,e*} yaygın olarak bulunur. Algoritma diğer alt problemleri işledikten sonra sadece geriye kalan {*a,e*} yaygın öğe grubunu bulur.

FP-Growth algoritması, örüntüleri çıkarmak için, öğe koşullu alt ağaçlar oluşturduktan sonra, her bir öğe koşullu alt ağacı *özyineli* olarak dolaşır. Bundan dolayı her öğe koşullu alt ağacın kendisi de, kendi altında öğe koşullu alt ağaçlarını oluşturur.

3. ÇOK ÖĞELİ VE NİCEL DEĞERLİ VERİTABANINDAN SIRALI ÖRÜNTÜLERİN ÇIKARILMASI: YENİ BİR FP-GROWTH YAKLAŞIMI

Bu tez çalışmasında, ele alınan veri kümeleri çok öğeli, nicel değerli ve zaman damgalı niteliklidir. Klasik FP-Growth çalışmalarında amaç sadece sık görülen yaygın öğe gruplarını ortaya çıkarmak olduğundan nicel değerin ve zaman damgasının önemi yoktur. Sık görülen yaygın öğe gruplarını ortaya çıkarmak için hareket sırası niteliği yeterlidir. Biz bu tezde önceki çalışmalardan farklı olarak FP-Growth algoritmasına farklı bir yaklaşım getirilerek, çok-öğeli veritabanından sıralı örüntüleri çıkarmak için yeni bir yöntem sunulmuştur. Zaten gerçek dünyada veri kümeleri çoğunlukla çok öğeli işlem, nicel değerli ve zaman damgalı nitelikli verilerden oluşmaktadır. Yani bu çalışmada ele alınan işlem kayıtlarının veri yapısı şu şekildedir. Müşteri Id, işlem zamanı ve miktar şeklindedir. Eğer veri kümesinde hareket sırası (TID) yoksa bu nitelik müşteri id ve işlem zamanı dikkate alınarak oluşturulacaktır.

FP-Growth yaklaşımı, Apriori algoritmasına göre, çok maliyetli olan veritabanının defalarca taranması ve çok sayıda aday üretme ve test etme sakıncalarını ortadan kaldırmasına ve aynı zamanda Apriori algoritmasına göre sıralı örüntüleri bulurken daha hızlı çalışmasına ve daha etkili olmasına rağmen;

- Karmaşık bir veri yapısı kullanmaktadır.
- Apriori algoritmasıyla karşılaştırdığımızda bu veri yapısını oluşturmak daha zordur.
- Ayrıca öğe koşullu alt ağaçları oluşturup dolaşmak için gereken özyineli işlemler bellek kaynağını çok ciddi anlamda tüketmektedir.

Yukarıda belirtilen FP-Growth yaklaşımının dezavantajlarına şunu da eklemek gerekir: Bu karmaşık veri yapısını Linked-list v.b bir yapı kullanarak bellek üzerinde oluşturmak hiç gerçekçi değildir[25]. Bununla birlikte özyineli işlemleri de bellek üzerinde çalıştırmak yığıt taşmasına neden olacaktır. Çünkü özyineli işlemler için bellek üzerinde oluşturulan yığıt veri yapısı kullanılmaktadır.

Dolayısıyla eğer FP-Growth algoritmasının, örüntüleri ortaya çıkarmak için kullandığı FP-tree ağaç veri yapısını bellek üzerinde oluşturup, bütün bir algoritmayı çalıştırmak istersek, ele alınacak veri kümesinin büyüklüğüne çok ciddi anlamda bir limit getirmemiz

gerekecektir[28]. Bu şekilde sadece 5-10 satırlık işlemleri ele alabiliriz. Yani ciddi bir veri kümesini ele alıp bütün bir algoritmayı bellek üzerinde çalıştırmak mümkün değildir.

Bu tezde, yukarıda belirtilen nitelikteki veri kümelerinden sıralı örüntüleri elde etmek için, öncelikli ilişkisel veri tabanları üzerinde basit bir tablo veri yapısı kullanılacaktır. FP-tree ağacını oluşturmak için bir tablo yeterli olacaktır. Aşağıda bu ağacın tablo veri yapısı üzerinde oluşturulma mantığı ve algoritması verilmiştir.

Ayrıca FP-Growth algoritması, öğe koşullu alt ağaçlar oluşturmak ve bu her bir öğe koşullu alt ağaçları dolaşması için öz yineli işlemler gerektirmektedir. Bu tezde, FP-tree ağacı oluşturulduktan sonra yaygın öğeleri elde etme sürecinde gerekli olan işlemler için, FP-Growth algoritmasına yeni bir yaklaşım önerilmiştir. Bu yaklaşımda öğe koşullu alt ağaçlar oluşturulmadan, FP-tree ağacının tamamını dolaşmadan ve özyineli işlemler kullanılmadan, öğe tabanlı aday kümeler üretilerek yaygın öğeler elde edilmektedir.

Yaygın öğeler elde edildikten sonra, sıralı örüntüleri elde etmek için, veri kümesinden gerekli olmayan öğelerin tespit edilmesi ve elenmesi için bulanık küme teorisi kullanılarak sıralı örüntüler elde edilmektedir.

3.1. Veri Kümesini Hazırlama

Bilindiği üzere bir sıralı örüntü ile “A alındıktan sonra B alınmaktadır” kuralı elde edildiğinde bundan sonraki davranış eğilimi tahmin edilebilir. Dolayısıyla sıralı örüntülerde hareket sırası önemlidir. Eğer ele aldığımız veri setinde sadece müşteri ID, zaman damgası ve miktar nitelikleri varsa bu durumda bu veri setini kullanmaya başlamadan önce müşteri Id ve zaman damgasına göre hareket sırasını (TID) oluşturmak gerekir. Örneğin ele alınan veri kümesi Tablo 3.1’ deki gibi olsun.

Tablo 3.1. Örnek veri kümesi [23].

Müşteri	Hareket zamanı	Yapılan alışveriş
1	5 Haziran 2009	(B,2) (C,3)
1	17 Haziran 2009	(E, 3)
2	13 Haziran 2009	(D, 5)
2	19 Haziran 2009	(B,2) (C, 4)
2	21 Haziran 2009	(D, 9)
3	7 Haziran 2009	(A, 7) (B, 8)
3	15 Haziran 2009	(D, 7) (E, 9)
4	11 Haziran 2009	(B, 1) (C, 3)
4	16 Haziran 2009	(C, 8)
4	23 Haziran 2009	(D, 7) (E, 10)
4	30 Haziran 2009	(B, 1)
5	3 Haziran 2009	(D, 6) (E, 12)
5	13 Haziran 2009	(B, 2) (C, 4)
5	17 Haziran 2009	(C, 6)
6	19 Haziran 2009	(D, 6) (E, 10)
6	25 Haziran 2009	(B, 3) (C, 5)

Tablo 3.1’ de verilen örnek veri kümesinin veri tabanı üzerindeki tablo yerleşimi Tablo 3.2’ deki gibidir

Tablo 3.2. Tablo 3.1’ deki veri kümesinin veri tabanı tablosu üzerindeki yerleşimi

ITEM	CID	T_DATE	AMOUNT
b	1	05/07/2009	2
c	1	05/07/2009	3
e	1	17/07/2009	3
d	2	13/07/2009	5
b	2	19/07/2009	2
c	2	19/07/2009	4
d	2	21/07/2009	9
a	3	07/07/2009	7
b	3	07/07/2009	8
d	3	15/07/2009	7
e	3	15/07/2009	9
b	4	11/07/2009	1
c	4	11/07/2009	3
c	4	16/07/2009	8
d	4	23/07/2009	7

TID = 1 hareket grubu

TID = 2 hareket grubu

TID = 3 hareket grubu

TID = 4 hareket grubu

Böyle bir veri kümesini örnek olarak ele aldığımızda, yapılan hareketlerin, hareket sırasını müşteri ve zaman damgalı niteliğine bağlı olarak oluşturmamız gerekir. Çünkü her şeyden önce FP-Growth algoritması yaygın öğeleri elde etmek için FP-tree ağaç yapısını inşa ederken hareket sırasından (TID) yola çıkar. Buna ek olarak kimin neyi ne zaman aldığının sırası önemlidir. Ayrıca aynı tarihte ama farklı zamanda gerçekleşen işlemleri de birbirinden ayırmak gerekir. Örneğin müşteri kodu 1 olan kişi 5 Haziran 2009 tarihinde B ve C öğelerini aldıktan sonra aynı müşteri aynı tarihte yani 5 Haziran 2009 tarihinde farklı bir zamanda başka öğeler alırsa bu hareketi bu müşteri bazında ayırmak gerekir. Çünkü B ve C öğesini aldıktan sonra gelip aynı tarihte ancak farklı bir zamanda, farklı bir hareket olarak, X öğesini alırsa bunun sırasının bilinmesi gerekir. Yani aynı tarihteki iki farklı hareketin ayrı ayrı hareketler olarak ele alınması gerektiğinden dolayı B ve C öğesinin alınma hareketinin hareket sırası 1, X öğesinin alınma hareket sırası da 2 olarak atanmalıdır. Veri kümesindeki diğer hareketler için de bu hareket sıraları grupları ardışık olarak oluşturulur. Bu düşünceden hareketle yukarıdaki örnek tabloda gösterilen veri kümesinin işlemden sonraki durumu Tablo 3.3.' te gösterildiği gibi olur. *t* tablosu örnek veri kümesidir. Yeni duruma göre oluşan tablo *t* olarak ele alınacaktır.

Tablo 3.3. Veri kümesinin hareket sırası oluşturulduktan sonraki durumu (*t* veri kümesi)

TID	ITEM	CID	T_DATE	AMOUNT
1	b	1	05/07/2009	2
1	c	1	05/07/2009	3
2	e	1	17/07/2009	3
3	d	2	13/07/2009	5
4	b	2	19/07/2009	2
4	c	2	19/07/2009	4
5	d	2	21/07/2009	9
6	a	3	07/07/2009	7
6	b	3	07/07/2009	8
7	d	3	15/07/2009	7
7	e	3	15/07/2009	9
8	b	4	11/07/2009	1
8	c	4	11/07/2009	3
9	c	4	16/07/2009	8
10	d	4	23/07/2009	7

row(s) 1 - 15 of 26

……..

Bu sıralı hareketleri oluşturmak için aşağıdaki çalışan kod kullanabilir. Veri kümesinin yerleştiği tablo adı *t* olarak ele alınmıştır.

```
declare
  cursor cc is
   select distinct cid
   from tt order by cid ;
  cursor xx(c_id number) is
   select cid,item,t_date
   from tt
   where cid = c_id
   group by cid,item,t_date
   order by cid,t_date ;
seq number(3);
seq_date date ;
begin
  seq := 0 ;
  seq_date := '01/01/1901' ;
  for c in cc loop
    for x in xx(c.cid) loop
     if x.t_date <> seq_date then
       seq := seq + 1 ;
       seq_date := x.t_date ;
     end if ;
     insert into t values(seq,x.item,x.cid,x.t_date) ;
    end loop ;
   seq_date := '01/01/1901' ;
 end loop ;
end ;
```

3.2. FP-tree Ağacının Sql Tabanlı Oluşturulması

FP-tree oldukça yoğun ve genellikle orijinal veri kümesinden daha küçük olmasına rağmen, geniş veri tabanlarında daha öncede belirtildiği gibi bellek üzerinde oluşturmak hiç gerçekçi değildir.

Veritabanı sistemleri, veriye erişim için, filtreleme için ve veri indekslemede güçlü mekanizmalar sunar. Dolayısıyla veritabanları sistemlerinin bu avantajlarından ve SQL dilinin yeteneklerinden faydalanmak gerekir[23,24].

FP-tree ağacını, bellek üzerinde oluşturmak yerine, veritabanında basit tablo veri yapısı üzerinde oluşturmak daha gerçekçi bir yaklaşımdır[25].

FP-tree ağacını oluşturmaya başlamadan önce, veritabanı bir kere taranarak frekans tablosu (header) oluşturulur ve öğeler frekans sayısına göre yukarıdan aşağıya sıralanır. Bu tablonun adı *f* olarak kabul edilsin. *f* tablosu öğe adı (item) ve veri kümesinde tekrarlanma sayısı olan frekans (cnt) sütunlarından oluşur. Bu tablo, frekans sayısı kullanıcı tanımlı minimum eşik değerini aşan (destek sayısı) yaygın öğeleri gösterir. Yaygın olmayan öğeler bu tabloda yer almaz. Aslında *f* tablosu bir uzunluklu ve destek sayısını aşmış yaygın öğelerdir. Bu tablo aşağıda gösterilen çalışan kod ile oluşturulur.

```
insert into f
     select item,count(*) from t
     group by item
     having count(*) >= min_support
     order by count(*) desc,item ;
```

Burada *t*, veri kümesi, min_support, kullanıcı tanımlı eşik değeridir. Tablo 2.1' deki örnek veri kümesi ele alınırsa, frekans tablosundaki öğeler ve sıklık değerleri sırasıyla (a, 8), (b, 7), (c, 6), (d, 5), (e, 3) olur.

Frekans tablosunu oluşturduktan sonra, *t* veri kümesindeki yaygın olmayan öğeleri elemek için, *f* tablosuna bağlı olarak *t* veri kümesi *t2* tablosuna aktarılır. *t2* tablosu *f* tablosunda yer almayan öğelerin, *t* veri kümesinden elenmesiyle elde edilen ve sadece yaygın öğeleri içinde barındıran tablodur. Bu aktarma işlemi aşağıdaki çalışan kod ile gerçekleştirilir.

```
insert into t2
 select t.tid,t.item
 from f,t
 where f.item =t.item
 order by t.tid,f.cnt desc,t.item ;
```

t2 tablosundaki öğeler, daha fazla önek paylaşımı olabilmesi için kendi frekans değerlerine göre yukarıdan aşağıya sıralanır.

Tablo 2.1' de verilen basit örnek veri kümesini ele alalım. Kullanıcı tanımlı eşik değeri (minimum destek sayısı) 2 olsun. *t* tablosunun veri tabanı üzerindeki yerleşimi, veritabanının bir kere taranmasından sonra oluşan *f* tablosu ve *t2* tablosu, Tablo 3.4 ile gösterilmiştir.

Tablo 3.4. Örnek *t*, *f* ve *t2* tabloları, (a) örnek veri kümesi *t*, (b) *t2* tablosu, (c) *f* tablosu

TID	ITEM	CID	T_DATE	AMOUNT
1	a	-	-	-
1	b	-	-	-
2	b	-	-	-
2	c	-	-	-
2	d	-	-	-
3	a	-	-	-
3	c	-	-	-
3	d	-	-	-
3	e	-	-	-
4	a	-	-	-
4	d	-	-	-
4	e	-	-	-
5	a	-	-	-
5	b	-	-	-
5	c	-	-	-

row(s) 1 - 15 of 29

(a)

TID	ITEM	CID	T_DATE	AMOUNT
1	a	-	-	-
1	b	-	-	-
2	b	-	-	-
2	c	-	-	-
2	d	-	-	-
3	a	-	-	-
3	c	-	-	-
3	d	-	-	-
3	e	-	-	-
4	a	-	-	-
4	d	-	-	-
4	e	-	-	-
5	a	-	-	-
5	b	-	-	-
5	c	-	-	-

row(s) 1 - 15 of 29

(b)

ITEM	CNT
a	8
b	7
c	6
d	5
e	3

row(s) 1 - 5

(b)

3.2.1. Önerilen Yöntemle FP-tree Ağacını Oluşturma

Daha önce de belirtildiği gibi, FP-tree ağacı veritabanı üzerinde basit tablo veri yapısı üzerinde kurulacaktır. Bunun için sunulacak tablo üç alandan oluşmaktadır. Öğe tanımlayıcı (item), öğenin bir alt ağaçtaki içinde bulunduğu işlem sayısı olan sayaç (cnt) ve öğe önek alt ağaç yolu (path). Path alanının işlevi sadece FP-tree ağacını kurarken fayda sağlamak değildir. Bu alan aynı zamanda, FP-tree ağacından bütün yaygın örüntüleri

bulurken de fayda sağlayacaktır. FP-tree ağacı veritabanı tablosu üzerine kurulurken, path alanı, *f* tablosunda yer alan bir öğenin, FP tablosuna eklenip eklenmemesine karar veren önemi bir koşul olacaktır. Şöyle ki; bir öğenin FP tablosuna eklenme koşulu şudur [25] ; her harekette ilgili öğeler grubu ele alınacak şekilde; eğer FP öğe tablosunda yoksa veya FP tablosunda aynı öğeler var ancak eklenip eklenmemesine karar verilecek olan öğenin mevcut path bilgisi var olan aynı öğelerden farklıysa, FP tablosuna eklenir. Aksi durumda aynı path bilgisine sahip öğenin sayacı bir arttırılır. FP tablosunun path bilgi alanı her yaygın öğenin önek öğeleri belirleyen bilgiyi sakladığı için, bu öğe ile birlikte görülen öğelerin bulunmasında büyük bir kolaylık sağlar. Path alanı, FP-tree ağacı kullanılarak yaygın örüntülerin bulunması sürecinde, bu çalışmada yer alan ve FP-Growth algoritmasına yeni bir yaklaşım getiren, öğe koşullu alt ağaçların oluşturulup özyineli olarak dolaşılması yerine öğe tabanlı aday üretme aşamasında da kullanılacaktır. Özetle her yaygın öğe aşağıdaki gibi test edilir :

- Eğer FP tablosu aynı öğe ve path bilgisine sahip değilse, öğe FP tablosuna sayaç değeri 1 olarak eklenir
- Aksi durumda FP tablosu sayaç değeri 1 arttırılarak güncellenir.

Örnek olarak Tablo 2.1’ deki örnek veri kümesini ele alalım :

Tablo 3.5. *t*2 tablosu

TID	ITEM	CID	T_DATE	AMOUNT
1	a	-	-	-
1	b	-	-	-
2	b	-	-	-
2	c	-	-	-
2	d	-	-	-
3	a	-	-	-
3	c	-	-	-
3	d	-	-	-
3	e	-	-	-

TID = 1 hareket grubu

TID = 2 hareket grubu

Frekans tablosu oluşturulup, *t* veri kümesinden yaygın olmayan öğelerin elenmesinden sonra oluşan *t2* tablosu Tablo 3.5.’ te gösterilmiştir. Bu *t2* tablosuna göre FP-tree ağacını oluşturma akışı aşağıdaki gibidir :

İlk öğe *a* okunur. Okunan öğe *a* için path alanı “*null* “ olur. Sonraki öğe *b* için path alanı “ *null : a* “ olur. Üçüncü öğe *b* için path alanı “*null* “ , sonraki öğe *c* için path alanı

“ *null : b* “, *d* öğesi için path alanı “*null : b : c* “ olur. Bu şekilde her hareket sırası okunarak işlemler devam eder ve FP –tree ağacı oluşturulur.

Ancak, FP-tree ağacını oluştururken her yaygın öğe bire bir test edilmelidir. Her okunan öğe için FP tablosu, öğenin tabloya eklenip eklenmemesine karar verilmesi için test edilmek zorundadır. Bu bize gereksiz zaman kaybettirir. Bunun yerine genişletilmiş FP yaklaşımı kullanılır[25].

3.2.2. Genişletilmiş FP-tree Yaklaşımı

Genişletilmiş FP-tree, *t2* tablosunda hareketlerin içindeki yaygın öğelerin doğrudan dönüşümüyle elde edilir. Bu yaklaşımda, her hareket sırasının içinde yer alan ilk yaygın öğenin path alanı “*null*“ olarak atanır. İkinci yaygın öğe için path “*null : i_1* “, üçüncü yaygın öğe için path “*null : i_1 : i_2* “ olarak atanır ve bu şekilde her hareket sırasına ait öğeler okunarak işlemler devam eder[25]. Genişletilmiş FP-tree ağacı bize, her harekete ait yaygın öğeleri ve onların önek path bilgisinin tamamını verir. Genişletilmiş FP-tree ağacı için kullanılacak olan veri tablosu öğe (item) ve path bilgisini gösteren iki alandan oluşur. Genişletilmiş FP-tree ağacı elde edildikten sonra, aynı path bilgisi olan öğeler birleştirilerek FP-tree ağacı elde edilir. Tablo 2.1’ deki örnek veri kümesi kullanılarak elde edilen Genişletilmiş FP-tree ve bundan elde edilen FP Tablo 3.6 ile , geliştirilen kod ise aşağıda gösterilmiştir. Şekil 3.1 ise bu ağacın grafiksel gösterimidir.

```
PROCEDURE create_gFP IS

    cursor get_tid is
      select distinct tid from t2
      order by tid ;
    cursor get_path(t_id number) is
      select tid,item from t2
      where tid = t_id ;
    cur_path varchar2(255) ;
    prefix_item varchar2(10) ;

 BEGIN

  for gtid in get_tid loop
    cur_path := null ;
    for g_path in get_path(gtid.tid) loop
      if cur_path is null then
        cur_path := 'null' ;
      else
        cur_path := cur_path||':'||prefix_item ;
      end if ;

      insert into gFP values(g_path.item,cur_path) ;
```

```
     prefix_item := g_path.item ;
   end loop ;--get path
 end loop ;-- distinct tid

END create_gFP ;

--gFP den FP yi olusturma

PROCEDURE get_FP_from_gFP IS
BEGIN
  insert into FP
    select item,count(*) cnt,path from gFP
    group by item,path
    order by cnt desc ;
END get_FP_from_gFP ;
```

Tablo 3.6. Genişletilmiş FP ağacı ve bundan elde edilen FP
(a) Tablo 2.1 örnek veri kümesi için, genişletilmiş FP ağacı
(b) genişletilmiş FP ağacından elde edilen FP

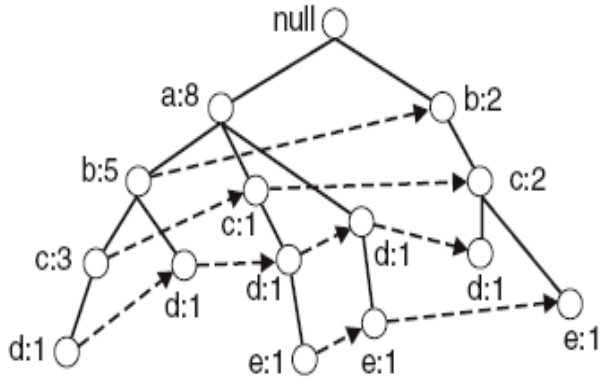
ITEM	PATH
a	null
b	null:a
b	null
c	null:b
d	null:b:c
a	null
c	null:a
d	null:a:c
e	null:a:c:d
a	null
d	null:a
e	null:a:d
a	null
b	null:a
c	null:a:b

ITEM	PATH
a	null
b	null:a
c	null:a:b
d	null:a:b:c
a	null
a	null
b	null:a
c	null:a:b
a	null
b	null:a
d	null:a:b
b	null
c	null:b
e	null:b:c

(a)

ITEM	CNT	PATH
a	8	null
b	5	null:a
c	3	null:a:b
b	2	null
c	2	null:b
d	1	null:a
d	1	null:a:b:c
e	1	null:a:d
e	1	null:a:c:d
d	1	null:a:c
d	1	null:a:b
c	1	null:a
d	1	null:b:c
e	1	null:b:c

(b)



Şekil 3.1. Örnek veri kümesi çin FP-tree ağacının grafiksel gösterimi [22].

3.2.3. FP-tree Ağacından Sık Tekrarlanan Yaygın Örüntülerin Bulunması

FP tablosunu oluşturduktan sonra, artık bu tabloyu yaygın örüntülerin kümesini elde etmek için etkili bir şekilde kullanabiliriz. Bölüm 2.2.2' de FP-Growth algoritmasının, örüntüleri ortaya çıkarırken, her yaygın öğe için öğe koşullu alt ağaçları oluşturduğunu ve bu alt ağacı özyineli olarak dolaştığını anlatmıştık. Bu bölümde FP-tree ağacından örüntüleri elde etmek için öğe koşullu alt ağaçlar oluşturup, bu alt ağacı özyineli olarak dolaşıp örüntü bulmaya çalışmayacağız. Çünkü bu işlemde her yinelemede FP-tree ağacını oluşturur gibi alt ağaçlar oluşturulur. Üstelik bu oluşturma özyineli olduğu için, her öğe koşullu alt ağacın kendisi de içinde öğe koşullu alt ağaçlar oluşturur. Bu yaklaşım hem karmaşık bir durum ortaya çıkarır hem de üçüncü bölümün başında anlatıldığı gibi, bu özyineli işlemler bellek kaynağını ciddi bir şekilde tüketir. Ayrıca bu yaklaşım geniş veritabanlarında yığıt taşmasına da sebep olacaktır. Dolayısıyla örüntü bulma işlemleri tamamlanamayacaktır. Aslında büyük veritabanlarında, öğe koşullu alt ağaçların oluşturulması ve her bir alt ağacın özyineli olarak dolaşılması pek gerçekçi olmamakla birlikte uygulanması pek mümkün görünmemektedir. Bu tez çalışmasında alt ağaçlar özyineli olarak oluşturularak, özyineli olarak dolaşılmamaktadır.

Örüntülerin bulunması için, FP-tree ağacı aşağıdan yukarıya doğru bir yaklaşımla dolaşıldı. Yani önce *e* ile biten örüntüleri, sonra sırasıyla *d*, *c*, *b* ve *a* ile biten örüntüler elde edildi. Oluşturulan örnek *f* tablosuna göre, örüntülerin ortaya çıkarılmasına frekansı en düşük olan *e* öğesinden başladık. FP tablosundaki öğeye ait path alanı, bu öğe ile birlikte görülen öğeler bilgisini verir. Bunun için oluşturulan FP tablosundan *e* öğesine ait satırları sayaç değerleriyle birlikte okunur.

Her satır için sıralı bir şekilde hareket sırası atandı (1, 2, 3, ..) Daha sonra bu satırlardaki öğeler, hareket sırası dikkate alınarak ayrıştırılıp *e* öğesine ait yeni bir lokal işlem veri kümesi oluşturuldu. Lokal işlem veri kümesi Tablo 3.7 ‘ de gösterilmiştir.

Tablo 3.7. *e* için lokal işlem veri kümesi

Hareket ID (TID)	Öğeler (items)
1	{a, d}
2	{a, c, d}
3	{b, c}

Bu lokal işlem veri kümesini oluşturmak için tmp olarak çağrılan geçici bir tablo kullanıldı. Bu tmp tablosu üç alandan oluşmaktadır. Hareket sırası (TID) , öğe tanımlayıcı (item) ve *e* ile birlikte görülen öğelerin görülme sıklığını gösteren sayaç (cnt) . Bu sayaç bilgisi FP tablosunda cnt alanında gösterilir. Örneğin FP tablosundan okunan e öğesine ait satırdaki cnt bilgisi 2 ise, bu satıra ait path alanındaki ayrıştırılacak olan ve tmp tablosuna eklenecek olan öğelerin de cnt sayısı 2 olur. Bu sayaç daha sonra lokal işlem veri kümesinde yer alan öğelerin destek sayısını aşıp aşmadıklarını anlamak için kullanılacaktır. FP tablosundan okunan her bir path satırını bir hareket sırası olarak ele aldık. Bu hareket sıraları tmp tablosundaki hareket ID alanına karşılık gelir.

Bu veri kümesini oluştururken, her bir e öğesinin path bilgisinde yer alan ve ayrıştırılan öğeleri, bir hareket sırasına göre tmp tablosuna ekledik. FP tablosundan, e öğesine ait satırların bulunması ve bu satırlardaki path bilgisi alanında yer alan öğelerin geçici olarak kullanılan tmp tablosuna eklenme işlemleri için geliştirilen çalışan kod aşağıda verilmiştir.

```
cursor for_pb(s_item varchar2) is
   select item,cnt,ltrim(path,'null:') path
   from FP where item = s_item ;

for pbi in for_pb(ff.item) loop
  insert into tmp
  select temp_tid,REGEXP_SUBSTR (pbi.path,'[^:]+', 1, level),pbi.cnt
  from dual
  connect by level <= length(regexp_replace(pbi.path ,'[^:]*'))+1 ;
  temp_tid := temp_tid + 1 ;
 end loop ; --for_pb loop cursor
```

Örnek olarak *e* ile biten örüntülerin çıkarılması ele alındığı için, burada s_item parametresinin ve *f* tablosundan ele alınan öğe olduğu için ff.item parametresinin değeri *e*

öğesidir. ff.item değeri frekans tablosundan gelen değerdir. pbi.path değeri FP tablosundan gelen path bilgisidir. FP tablosundan *e* öğesine ait üç path satırı okunur. Bu satırlar Tablo 3.8' de gösterilmiştir.

Tablo 3.8. FP tablosunun *e* ile biten öğeler için durumu

ITEM	CNT	PATH
a	8	null
b	5	null:a
c	3	null:a:b
b	2	null
c	2	null:b
d	1	null:a
d	1	null:a:b:c
e	1	null:a:d
e	1	null:a:c:d
d	1	null:a:c
d	1	null:a:b
c	1	null:a
d	1	null:b:c
e	1	null:b:c
		row(s) 1 - 14 of 14

FP tablosundan elde edilen bu satırlardaki *e* ile birlikte görülen öğelerin ayrıştırılmasından sonra, hareket sırasına göre tmp tablosuna eklenmesiyle elde edilen *e* öğesi için öğe tabanlı işlem veri kümesi Tablo 3.9'da gösterilmiştir.

Tablo 3.9. *e* için oluşturulan lokal işlem veri kümesi

TID	ITEM	CNT
1	a	1
1	d	1
2	a	1
2	c	1
2	d	1
3	b	1
3	c	1

e öğesi için, öğe tabanlı işlem veri kümesini oluşturulduktan sonra, bu öğe tabanlı işlem veri kümesi için lokal frekans tablosu oluşturuldu. Lokal frekans tablosu

oluşturulurken destek sayısını aşamayan öğeler elendi. Yani bu lokal frekans tablosundaki öğeler *e* için 1-uzunluklu yaygın öğelerdir. Bunun anlamı şudur: demek ki bu lokal frekans tablosundaki öğeler, tüm işlem veri kümesinde *e* ile birlikte en az destek sayısı kadar birlikte görünmüşlerdir. Tablo 3.9 için öğeler ve sayaç değerleri şu şekilde oluşur : (a, 2), (b, 1), (c, 2), (d, 2) destek sayısı 2 olduğu için, burada *b* öğesi destek sayısını aşamadığı için elenir. Oluşan lokal frekans tablosu Tablo 3.10' da gösterilmiştir.

Tablo 3.10. *e* için lokal frekans tablosu

ITEM	CNT
a	2
c	2
d	2

row(s) 1 - 3 of 3

Bu lokal frekans tablosu için, lokalf tablosunu kullandık. Bu tablo öğe tanımlayıcı olarak item, öğenin sıklık sayısını gösteren cnt alanlarından oluşmaktadır. lokalf tablosunu oluşturmak için geliştirilen kod aşağıda verilmiştir.

```
insert into lokalf
  select item,sum(cnt) from tmp
  group by item
  having sum(cnt) >= min_support
  order by sum(cnt) desc, item ;
```

Burada min_support, kullanıcı tanımlı eşik değeri olan destek sayısı, tmp tablosu ise *e* için oluşturulan lokal işlem veri kümesidir.

Eğer lokalf tablosunda destek sayısını aşmış hiçbir öğe yoksa demek ki *t* örnek işlem veri kümesinde *e* ile biten hiçbir örüntü yaygın değildir. Bu noktada *e* için işlemler biter ve *f* tablosunda yer alan *d* ile biten örüntülerin ortaya çıkarılması için, *e* öğesi için yapılan yukarıdaki işlemler *d* için başlar.

Eğer lokalf tablosunda en az bir öğe varsa, bundan sonra *e* ile biten yaygın örüntülerin ortaya çıkarılması işlemleri başlar. Bu noktada öğe koşullu alt ağaç oluşturup bu alt ağacı özyineli olarak dolaşmanın anlamı yoktur. Bunun yerine bu tezde önerilen, lokalf tablosunda yer alan öğelerin *e* ile birlikte görünme kombinasyonlarını düşünüp öğe tabanlı aday örüntü kümeleri üretmektir. Bu aday örüntü kümeleri FP tablosunun tamamında değil sadece öğe tabanlı, yani FP tablosunun o öğe ile ilgili satırlardaki path alanına bakılarak test edilir ve yaygın olup olmadıklarına karar verilir. *e* için oluşan lokalf tablosundaki

öğelere dayanarak üretilen bu aday örüntü kümeler Tablo 3.8' de gösterildiği gibi FP tablosundan sadece üç satırdaki path alanındaki öğeler dikkate alınarak test edilir. Öğe tabanlı aday örüntü kümelerinin üretimini aşağıdaki çalışan kod ile gerçekleştirilebilir.

```
select ltrim(replace(sys_connect_by_path(item,','),',',':'),':') comb
 from lokalf
 connect by nocycle prior item < item  ;
```

Yukarıdaki kod çalıştırıldığında lokalf tablosundaki öğelere dayanılarak Tablo 3.11' de gösterilen aday örüntüler üretilir. Bu aday örüntüler geçici aday tablosunda tutulur.

Tablo 3.11. Lokalf tablosundan elde edilen aday örüntüler

COMB
a
a:c
a:c:d
a:d
c
c:d
d

Bu aday örüntüler üretilirken, bu aşamada aynı anlama gelecek aday örüntüler sadece bir kez üretilmiştir. Örneğin *a: c* örüntüsü ile *c: a* örüntüsü bu aşamada aynı anlama geldiğinden *c: a* örüntüsünü ayrıca aday örüntü olarak oluşturulmayacaktır. Ancak sıralı örüntülerde sıra önemli olduğundan bu durum tezin sonraki aşamalarında ele alınacaktır.

Bu aday örüntüler *e* ile biten öğeler içindir. Örneğin *a:c* örüntüsünü test etmek istediğimizde aslında biz *a:c:e* örüntüsünü test etmiş oluruz. Diğer bir deyişle gerçekte *e* ile biten bir örüntüyü test etmiş oluruz. Dolayısıyla bu aday örüntünün testini yukarıda belirttiğimiz gibi FP tablosundan sadece *e* ile ilgili satırlardaki path alanında yer alan öğeler üzerinden yaparız.

Her bir örüntü adayının öğe tabanlı frekans sayısına bakılır. Yani FP tablosundan sadece ilgili satırlardaki görünme sayısına bakılır. Eğer test edilecek aday örüntü tek bir öğeden oluşuyorsa, yani bir uzunluklu bir örüntü ise, sadece *a* gibi, bu örüntü adayı FP tablosunda değil de doğrudan lokalf tablosundaki cnt alanındaki sayaç değerine bakılırak test edilir. Eğer sayaç değeri destek sayısını aşıyorsa bu örüntünün yaygın olduğuna karar verilir ve yaygın öğeler tablosuna *e* ile biten yaygın örüntü olarak eklenir. Ancak bir uzunluklu aday örüntüyü diğer bir deyişle tek bir öğeden oluşan aday örüntünün kendisini

test etmeye gerek yoktur. Bu aday örüntüler lokalf tablosundan üretilmiştir. Lokalf tablosundaki öğeler zaten destek sayısını aşmış öğeler olduğundan, bir uzunluklu aday örüntüler doğrudan yaygın öğeler tablosuna eklenir. Eğer aday örüntü 2-uzunluklu, 3-uzunluklu..ise yani aday örüntü birden fazla öğeden oluşuyorsa, örüntünün frekans sayısı için FP tablosundan sadece ilgili satırlardaki görünme sayısına bakılır. Eğer örüntünün frekans sayısı destek sayısını aşarsa bu örüntünün yaygın bir örüntü olduğuna karar verilir ve yaygın öğeler tablosuna *e* ile biten yaygın örüntü olarak eklenir.

e ile biten örüntülerin ortaya çıkarılması örneğine devam edelim. FP tablosunun *e* ile ilgili satırlarındaki path alanlarındaki öğeler Tablo 3.12' de gösterildiği gibidir.

Tablo 3.12. FP tablosunun *e* ile ilgili satırlarındaki öğeler

ITEM	CNT	PATH
e	1	null:a:d
e	1	null:a:c:d
e	1	null:b:c

Tablo 3.11' de gösterilen ve lokalf tablosundan elde edilen aday örüntülerden bir uzunluklu olan aday örüntüler, *a, c, d,* yukarıda belirtildiği gibi yaygın örüntülerdir. Yani *a : e*, *c : e* ve *d : e* örüntüleri yaygın örüntü olarak belirlenir. Diğer aday örüntülerden *a : c* örüntüsünü test edelim. *a : c* aday örüntüsünü test etmek için yukarıdaki path alanlarına bakıldığında, sadece bir kez göründüğü anlaşılır. Dolayısıyla bu aday örüntünün frekans değeri bir olur ve destek sayısı 2 olduğundan bu örüntünün yaygın olmadığına karar verilir. Yani *a : c : e* aday örüntüsü yaygın bir örüntü değildir. Şimdi de *a: d* aday örüntüyü test edelim. bu örüntü hem *null: a : d* path alanında, hem de *null : a : c : d* path alanında göründüğü için, *a : d* aday örüntünün frekans değeri 2 olur. Destek sayısı da 2 olduğu için bu aday örüntünün yaygın bir örüntü olduğuna karar verilir ve *e* ile biten yaygın örüntüler kümesine eklenir. Diğer adaylar test edildikten sonra Tablo 3.14.' de gösterildiği gibi *e* ile biten örüntüler elde edilir.

Öğe tabanlı bu test işlemi yapılırken, kolaylık olsun diye aday örüntülerin her görünmesini, göründüğü path bilgisi ile birlikte geçici bir tabloda tuttuk. Tablo 3.13' da bu tablonun durumu gösterilmektedir. Daha sonra bu geçici tablodaki aday örüntüler, örüntü bazında gruplanarak frekans değerleri elde edildi. Destek sayısını aşan örüntüler asıl örüntü tablosuna eklendi.

Tablo 3.13. Geçici tabloda tutulan aday örüntülerin durumları

PATTERN	CNT	ITEM	PATH
a	1	e	null:a:d
a	1	e	null:a:c:d
a:c	1	e	null:a:c:d
a:c:d	1	e	null:a:c:d
a:d	1	e	null:a:c:d
a:d	1	e	null:a:d
c	1	e	null:b:c
c	1	e	null:a:c:d
c:d	1	e	null:a:c:d
d	1	e	null:a:d
d	1	e	null:a:c:d

Tablo 3.14. *e* ile biten örüntüler

PATTERN	CNT	ITEM
a:d:e	2	e
a:e	2	e
c:e	2	e
d:e	2	e

Bu işlemler, *d*, *b*, *c* ve *a* ile biten örüntülerin ortaya çıkarılması için de aynı şekilde tekrarlanarak Tablo 2.1' deki örnek işlem veri kümesi için sık tekrarlanan yaygın örüntüler elde edilir. Bu tezde önerilen algoritma ile elde edilen ve Tablo 3.15 ile gösterilen bu yaygın örüntüler, klasik FP-Growth algoritmasıyla elde edilen ve Tablo 2.2 ile gösterilen örüntülerle aynıdır.

Tablo 3.15. Önerilen yöntem ile elde edilen yaygın örüntüler

PATTERN	CNT	PB_ITEM
a:d:e	2	e
a:e	2	e
c:e	2	e
d:e	2	e
a:c:d	2	d
a:b:d	2	d
c:d	3	d
b:d	3	d
a:d	4	d
b:c:d	2	d
a:c	4	c
b:c	5	c
a:b:c	3	c
a:b	5	b
A	8	a

4. SIRALI ÖRÜNTÜLER

Sık tekrarlanan yaygın örüntülerin çıkarılması için kolay anlaşılması açısından nicel değerli ve tarih damgalı olmayan Tablo 2.1 ile gösterilen örnek işlem veri kümesi kullanıldı. Ancak sıralı örüntülerin çıkarılması için Tablo 3.1 ile gösterilen nicel değerli ve tarih damgalı örnek işlem veri kümesini kullanılacaktır.

Sık tekrarlanan yaygın örüntülerin çıkarılmasından sonraki aşama bu yaygın örüntülere dayanarak sıralı örüntülerin çıkarılması aşamasıdır. Sıralı örüntülerin ortaya çıkarılması aşamasının belli bölümlerinde bulanık küme teorisinden yararlanacağız.

4.1. Bulanık Küme Kavramları

İlk olarak 1965 yılında Zadeh [18] tarafından önerilen bulanık küme teorisinin, bulanık kümeler üzerinde kullanılan temel ve genellikle kullanılan işlemleri şunlardır :

$\mu_A(x)$: üyelik fonksiyonu

- Bir bulanık küme A'nın tümlemesi ¬ A ile gösterilir ve ¬ A' nın üyelik fonksiyonu aşağıdaki gibi verilir :

  $\mu_{\neg A}(x) = 1 - \mu_A(x), \quad \forall x \in X.$
- İki bulanık küme A ve B' nin kesişimi $A \cap B$ ile gösterilir ve $A \cap B$ ' nin üyelik fonksiyonu aşağıdaki gibi verilir :

  $\mu_{A \cap B}(x) = \min \{ \mu_A(x), \mu_B(x) \}, \quad \forall x \in X.$
- İki bulanık küme A ve B' nin birleşimi $A \cup B$ ile gösterilir ve $A \cup B$' nin üyelik fonksiyonu aşağıdaki gibi verilir :

  $\mu_{A \cup B}(x) = \max \{ \mu_A(x), \mu_B(x) \}, \quad \forall x \in X.$

4.2. Sıralı Örüntülerin Çıkarılması

Sıralı örüntülerin çıkarılması için ilk aşama, bölüm.3.2' de anlatılan sık tekrarlanan yaygın örüntülerin ortaya çıkarılması aşamasıdır. Yaygın örüntülerin çıkarılması aşamasında, *t* işlem veri kümesine göre bir uzunluklu yaygın öğeleri gösteren frekans tablosu ve FP-tree ağacı, veri tabanı tablosu kullanılarak oluşturulmuş ve bu yapıya dayanılarak, öğe koşullu alt ağaçlar oluşturulmadan öğe tabanlı aday örüntü kümeleri

üretilmiş ve aday örüntü kümelerinden destek sayısını geçen örüntüler sık tekrarlanan yaygın örüntü olarak kaydedilmişti.

Bu işlemlerin uygulama sonuçları ve sonraki aşamasında da elde edilecek sıralı örüntüler için Tablo 3.1 örnek işlem veri kümesi kullanılacaktır. Bu işlem veri kümesindeki hareketler ele alınarak, bölüm 3.1' de anlatıldığı gibi hareket sırası atayarak *t* tablosu oluşturulur. Oluşan *t* tablosunun veri tabanı üzerindeki görünümü Tablo 4.1 ile gösterilmiştir.

Tablo 4.1. Tablo 3.1 veri kümesinin hareket sırası oluşturulduktan sonraki durumu (*t* veri kümesi)

TID	ITEM	CID	T_DATE	AMOUNT
1	b	1	05/07/2009	2
1	c	1	05/07/2009	3
2	e	1	17/07/2009	3
3	d	2	13/07/2009	5
4	b	2	19/07/2009	2
4	c	2	19/07/2009	4
5	d	2	21/07/2009	9
6	a	3	07/07/2009	7
6	b	3	07/07/2009	8
7	d	3	15/07/2009	7
7	e	3	15/07/2009	9
8	b	4	11/07/2009	1
8	c	4	11/07/2009	3
9	c	4	16/07/2009	8
10	d	4	23/07/2009	7

row(s) 1 - 15 of 26

TID	ITEM	CID	T_DATE	AMOUNT
10	e	4	23/05/12009	10
11	b	4	30/05/2009	1
12	d	5	03/05/2009	6
12	e	5	03/05/2009	12
13	b	5	13/05/2009	2
13	c	5	13/05/2009	4
14	c	5	17/05/2009	6
15	d	6	19/05/2009	6
15	e	6	19/05/2009	10
16	b	6	25/05/2009	3
16	c	6	25/05/2009	5

row(s) 16 - 26 of 26

t veri kümesinden elde edilen ve bir uzunluklu yaygın öğeleri gösteren *f* frekans tablosu Tablo 3.2' de gösterilmiştir. *a* öğesi destek sayısını aşamadığı için elenmiştir.

Tablo 4.2. Bir uzunluklu yaygın öğeleri gösteren f frekans tablosu

ITEM	CNT
b	7
c	7
d	6
e	5

Bu frekans tablosuna bağlı olarak *t* tablosundan yaygın olmayan öğeler elenerek *t2* tablosu elde edilir. Elde edilen *t2* tablosu, Tablo 4.3 ile gösterilmiştir.

Tablo 4.3. Destek sayısını aşamayan öğelerin elenmesi sonucu elde edilen *t2* tablosu

TID	ITEM	CID	T_DATE	AMOUNT
1	b	1	05/05/2009	2
1	c	1	05/05/2009	3
2	e	1	17/05/2009	3
3	d	2	13/05/2009	5
4	b	2	19/05/2009	2
4	c	2	19/05/2009	4
5	d	2	21/05/2009	9
6	b	3	07/05/2009	8
7	d	3	15/05/2009	7
7	e	3	15/05/2009	9
8	b	4	11/05/2009	1
8	c	4	11/05/2009	3
9	c	4	16/05/2009	8
10	d	4	23/05/2009	7
10	e	4	23/05/2009	10
11	b	4	30/05/2009	1
12	d	5	03/05/2009	6
12	e	5	03/05/2009	12
13	b	5	13/05/2009	2
13	c	5	13/05/2009	4
14	c	5	17/05/2009	6
15	d	6	19/05/2009	6
15	e	6	19/05/2009	10
16	b	6	25/05/2009	3
16	c	6	25/05/2009	5

Bu *t2* tablosundan hareketle önce genişletilmiş FP sonra FP-tree ağacı oluşturulur. Oluşturulan FP-tree ağacı Tablo 4.4 ile gösterilmiştir.

Tablo 4.4. Fp-tree ağacı

ITEM	CNT	PATH
b	7	null
d	6	null
c	5	null:b
e	4	null:d
c	2	null
e	1	null

Bu yapıya dayanılarak, öğe koşullu alt ağaçlar oluşturulmadan öğe tabanlı aday örüntü kümeleri üretilir ve aday örüntü kümelerinden destek sayısını geçen örüntüler sık tekrarlanan yaygın örüntü olarak kaydedilir. Kaydedilen sık tekrarlanan yaygın örüntüler Tablo 4.5' te gösterilmiştir.

Tablo 4.5. Sık tekrarlanan yaygın örüntüler

PATTERN	CNT	ITEM
d:e	4	e
b:c	5	c

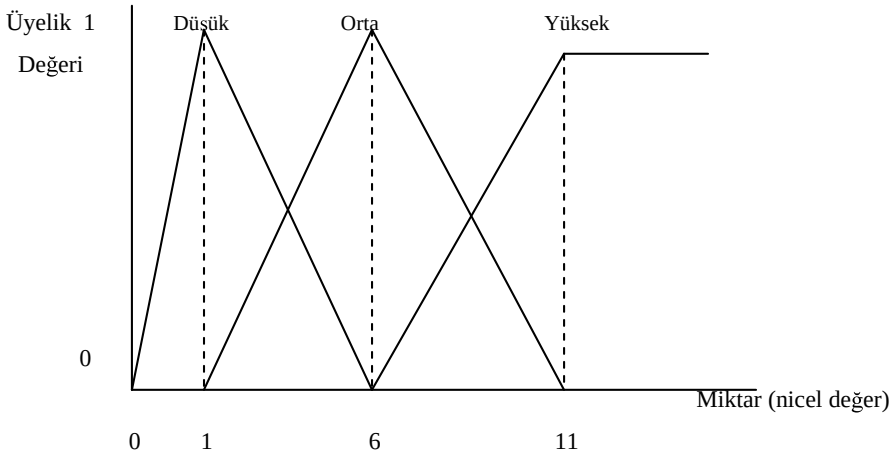
Bunları nasıl elde edildiğini anlatan işlem adımları bölüm 3.2.4' te detaylı bir şekilde anlatılmıştır.

Tablo.3.1 işlem veri kümesine göre sık tekrarlanan yaygın örüntüler bulunur. Bundan sonraki adım, sıralı örüntüleri ortaya çıkarmaktır. Öncelikle elde edilen sık tekrarlanan yaygın örüntüler ile, bir uzunluklu yaygın öğeleri gösteren *f* frekans tablosundaki öğeler birleştirilerek geniş öğe küme listesi oluşturulur. Bu geniş öğe listesi new_header diye adlandırılan yeni bir tabloda tutulur. İlerleyen aşamalarda aday örüntüler oluşturup test edilmesi amacıyla bu geniş öğe listesi kullanılacaktır. Elde edilen bu geniş öğe listesi Tablo 4.6' da gösterilmiştir.

Tablo 4.6. Geniş öğe listesi

ITEMSETS	CODES
b	1
c	2
d	3
e	4
d:e	5
b:c	6

Bu aşamadan sonra sıralı örüntüleri ortaya çıkarmak için bulanık küme teorisinden yararlanılacaktır. Bunun için ilk adım olarak işlem veri kümesine uygun olarak bir üyelik fonksiyonu kullanılır. Üyelik fonksiyonu işlem veri kümesindeki hareketlerde geçen nicel değerlere göre üç bulanık bölgeye ayrılır: düşük, orta ve yüksek. Bu üyelik fonksiyonuna göre her hareket için üç bulanık üyelik değeri üretilir. Sıralı örüntüleri ortaya çıkarmada kullanılacak Tablo.3.1' deki işlem veri kümesi için örnek olarak Şekil.4.1' de [26] gösterilen bir üçgen üyelik fonksiyonu kullanılacaktır.



Şekil 4.1. Örnek olarak kullanılan üçgen üyelik fonksiyonu[26].

Bundan sonraki işlem adımları aşağıdaki gibi olur :

- Her hareketin içinde yer alan nicel değerler bulanık kümelere dönüştürülür. Örneğin *t2* tablosunda müşteri kodu 1 olan müşterinin almış olduğu *E* öğesini ele alalım. *E* öğesinin 3 olan nicel değeri Şekil 4.1' deki üyelik fonksiyonuna göre ; (0.6 / E.Düşük + 0.4 / E.Orta) şeklinde dönüştürülür[26].
- Her hareket sırasında yer alan her öğenin sayısal önemi hesaplanarak, bulanık bölge sayıları elde edilir. Burada bir müşterinin hareketi içindeki öğenin aynı bölgesi için bulanık küme birleşimindeki maksimum işlemi kullanılır. Örneğin *D* öğesinin Orta bölgesini alalım. Bunun sayısal önemi aşağıdaki Tablo 4.7' ye göre = (0.0 + max (0.8, 0.4) + 0.8 + 0.8 + 1.0 + 1.0) = 4.4 olur[26].
- Her yaygın öğenin, olası üç bölgeden en yüksek sayılı bulanık bölgesi bulunur. Bulunan bu bölge artık bu öğe için referans bölgesi olarak alınır. Önerilen yaklaşımda, bu bölge, öğenin işlem veri kümesinden elenip elenmeyeceğine karar verilmesi açısından önemli olacaktır.

Bu noktadan sonra, yaygın öğe için bu en yüksek sayılı bulanık bölge dikkate alınarak, nicel değeri bu bölgeye uygun olmayan öğeler *t2* tablosundan elenir. Eldeki yeni işlem veri kümesi dikkate alınarak, geniş öğe kümeleri listesinden üretilen aday sıralı örüntüler *t2* tablosu üzerinden test edilerek, her aday örüntünün destek sayısı bulunur ve destek sayısını aşan örüntüler sıralı örüntüler olarak kaydedilir.

İşlem adımlarını uygulamaya başlamadan önce, *t2* tablosunu, düşük, orta ve yüksek bulanık bölgeleri temsilen üç yeni alan eklendi. Bir uzunluklu yaygın öğelerin frekanslarını gösteren *f* tablosuna da, en yüksek bulanık bölgeyi temsilen yeni bir alan eklendi.

Üyelik fonksiyonu dikkate alınarak işlem adımları uygulanmaya başlansın. *t2* tablosunda yer alan nicel değerleri, bulanık kümelere dönüştürüldü ve *t2* tablosuna eklenen üç alana da bu dönüşen değerler işlendi. *t2* tablosunun yeni durumu Tablo 4.8' de gösterilmiştir.

Tablo 4.7. Nicel değerlere karşılık gelen Bulanık Küme

CID	Bulanık Küme
1	[(0.8 / B.Low + 0.2 / B.Middle) (0.6 / C.Low + 0.4 / C.Middle)] [(0.6 / E.Low + 0.4 / E.Middle)]
2	[(0.2 / D.Low + 0.8 / D.Middle)] [(0.8 / B.Low + 0.2 / B.Middle) (0.4 / C.Low + 0.6 / C.Middle)] [(0.4 / D.Middle + 0.6 / D.High)]
3	[(0.6 / B.Middle + 0.4 / B.High)] [(0.8 / D.Middle + 0.2 / D.High) (0.4 / E.Middle + 0.6 / E.High)]
4	[(1.0 / B.Low) (0.6 / C.Low + 0.4 / C.Middle)] [(0.6 / C.Middle + 0.4 / C.High)] [(0.8 / D.Middle + 0.2 / D.High) (0.2 / E.Middle + 0.8 / E.high)] [(1.0 / B.Low)]
5	[(1.0 / D.Middle) (1.0 / E.High)] [(0.8 / B.Low + 0.2 / B.Middle) (0.4 / C.Low + 0.6 / C.Middle)] [(1.0 / C.Middle)]
6	[(1.0 / D.Middle) (0.2 / E.Middle + 0.8 / E.High)] [(0.6 / B.Low + 0.4 / B.Middle) (0.2 / C.low + 0.8 / C.Middle)]

Tablo 4.8. *t2* tablosunun bulanık değerlerden sonraki yeni durumu

TID	ITEM	CID	T_DATE	AMOUNT	L	M	H
1	b	1	05/05/2009	2	,8	,2	-
1	c	1	05/05/2009	3	,6	,4	-
2	e	1	17/05/2009	3	,6	,4	-
3	d	2	13/05/2009	5	,2	,8	-
4	b	2	19/05/2009	2	,8	,2	-
4	c	2	19/05/2009	4	,4	,6	-
5	d	2	21/05/2009	9	-	,4	,6
6	b	3	07/05/2009	8	-	,6	,4
7	d	3	15/05/2009	7	-	,8	,2
7	e	3	15/05/2009	9	-	,4	,6
8	b	4	11/05/2009	1	1	-	-
8	c	4	11/05/2009	3	,6	,4	-
9	c	4	16/05/2009	8	-	,6	,4
10	d	4	23/05/2009	7	-	,8	,2
10	e	4	23/05/2009	10	-	,2	,8
11	b	4	30/05/2009	1	1	-	-
12	d	5	03/05/2009	6	-	1	-
12	e	5	03/05/2009	12	-	-	1
13	b	5	13/05/2009	2	,8	,2	-
13	c	5	13/05/2009	4	,4	,6	-
14	c	5	17/05/2009	6	-	1	-
15	d	6	19/05/2009	6	-	1	-
15	e	6	19/05/2009	10	-	,2	,8
16	b	6	25/05/2009	3	,6	,4	-
16	c	6	25/05/2009	5	,2	,8	-

Bulanık kümeler bulunduktan sonra, her hareket sırası grubunda yer alan her öğenin sayısal önemi hesaplanarak, bulanık bölge sayıları elde edilir ve bu değerler FG olarak adlandırılan yeni bir tabloya kaydedilir. FG tablosu, öğeyi temsilen öğe alanı (item) ve düşük, orta, yüksek bölgeler için L, M ve H alanlarından oluşur. Burada bir müşterinin hareketi içindeki öğenin aynı bölgesi için bulanık küme birleşimindeki maksimum işlemi kullanılır. Sayısal önemlerin hesaplanması ve bulanık bölge sayılarının elde edilmesi için aşağıdaki kod kullanılır.

```
INSERT INTO FG
   SELECT item, SUM(max_l) sum_max_l,
                SUM(max_m) sum_max_m,
                SUM(max_h) sum_max_h
     FROM (SELECT item,
                cid,
                MAX(l) max_l,
                MAX(m) max_m,
                MAX(h) max_h
           FROM t2
           GROUP BY item, cid)
    GROUP BY item
    ORDER BY item ;
```

Bu işlemin sonucunda oluşan FG tablosunun veri tabanı üzerindeki görünümü Tablo 4.9' da gösterildiği olur.

Tablo 4.9. FG tablosu

ITEM	L	M	H
b	4	1,6	,4
c	2,2	3,4	,4
d	,2	4,4	1
e	,6	1,2	3,2

FG tablosunu elde ettikten sonra, her yaygın öğenin referans olarak kullanılacağı en yüksek bulanık bölgesini tayin edip bu bölgeyi f frekans tablosuna kaydederiz. Bunun için aşağıdaki çalışan kod kullanılır.

```
procedure set_region as
   cursor x is
     SELECT ITEM,L,M,H
     FROM FG
     ORDER BY ITEM ;
 region_flag varchar2(1) ;
 begin
    for xx in x loop
      if greatest(xx.L,xx.M,xx.H) = xx.L then
         region_flag := 'L' ;
      elsif  greatest(xx.L,xx.M,xx.H) = xx.M then
         region_flag := 'M' ;
      elsif greatest(xx.L,xx.M,xx.H) = xx.H then
         region_flag := 'H' ;
      end if ;
      UPDATE F
      SET region = region_flag
      where item = xx.item ;
    end loop ;
 end set_region ;
```

Her yaygın öğe için referans olan bulanık bölgenin tespitinden sonra *f* frekans tablosunun durumu Tablo 4.10' daki gibi olur.

Tablo 4.10. Bulanık bölge tespitinden sonra *f* tablosunun görünümü

ITEM	CNT	REGION
b	7	L
c	7	M
d	6	M
e	5	H

Bulanık bölge tespitinden sonra, *t2* işlem veri kümesindeki nicel değerli öğelerin işlem veri kümesinden elenip elenmeyeceğine karar verilir. *t2* işlem veri kümesindeki öğe düşük, orta ve yüksek bulanık değerler olan L, M ve H değerleriyle birlikte okunur ve bu öğe için *f* tablosundaki referans olarak kabul edilen bulanık bölgeye bakılır. Eğer bu öğeye ait nicel değer referans bulanık bölge aralığına düşmüyorsa, bu işlem satırındaki öğenin, bu işlem satırından elenmesine ve sıralı örüntüleri ortaya çıkarırken dikkate alınmaması gerektiğine karar verilir. Öğeye ait nicel değerin, referans bulanık bölgeye düşüp düşmediğine aşağıdaki şekilde karar verilir.

L,M ve H bulanık değerlerinden herhangi biri boş ise (null); demek ki bu işlem satırında, bulanık değer hesaplanırken ele alınan öğenin nicel değeri, bu bölgenin çok uzağında kalmıştır.

t2 tablosundan L, M ve H değerleriyle birlikte okunan öğenin ;

- Tablo.15'te gösterilen F tablosundaki referans bölgesi Düşük ise ;

Bu öğenin bu işlem satırındaki L alanındaki değerine bakılır ;

Eğer L değeri boş (null) ise veya L değeri M değerinden küçük ise bu öğe bu işlem satırından atılacak demektir. Yani bu öğenin bu işlem satırındaki nicel değeri kendi referans bölgesinin dışında kalmıştır.

- F tablosundaki referans bölgesi Orta ise ;

Bu öğenin bu işlem satırındaki M alanındaki değerine bakılır ; Orta bölge için öğenin hem sağ (Yüksek – H) bölgesindeki hem de sol bölgesindeki (Düşük – L) değer önemlidir. Yani her iki tarafında kontrol edilmesi gerekir.

1. Eğer M değeri boş (null) ise öğenin nicel değeri bölge dışıdır, elenir.
2. Eğer M değeri H değerinden küçük ve L değeri boş (null) ise öğenin nicel değeri bölge dışıdır, elenir

3. Eğer M değeri L değerinden küçük ise ve H değeri boş (null) ise öğenin nicel değeri bölge dışıdır, elenir.

Örneğin CID = 1 olan satırdaki (C, 3) söz konusu olsun. C öğesinin *f* tablosundaki referans bölgesi M yani Ortadır. *t2* tablosundan bu satır okunurken bu öğeye ait L, M ve H değerleri sırasıyla 0.6, 0.4 ve null gelir. M değeri L değerinden küçük olduğu için cid = 1 olan işlem satırındaki c öğesi işlem veri kümesinden atılır. Diğer bir deyişle *t2* tablosundan atılır.

- F tablosundaki referans bölgesi Yüksek (H) ise ;

Bu ögenin bu işlem satırındaki H alanındaki değerine bakılır;

Eğer H değeri boş (null) ise veya M değeri H değerinden büyük ise öğenin nicel değeri bölge dışıdır, elenir.

Örneğin, CID = 1 olan satırdaki (E, 3) öğesine bakalım. E öğesinin *f* tablosundaki referans bölgesi H yani Yüksektir. T2 tablosundan bu öğeye ait L, M ve H değerleri sırasıyla 0.6, 0.4 ve null gelir. H değeri boş (null) olduğu için bu öğe bu işlem satırından atılır.

Bu elemeyi yapan çalışan kod aşağıda verilmiştir.

```
procedure eliminate_outregion_items as
   cursor y is
     SELECT t2.item,L,M,H,tid,cid,region,silinecek
     FROM T2,F
     WHERE t2.item = f.item ;
    outregion varchar2(1) ;
 begin
  for xx in y loop
     outregion := null ;
    if xx.region = 'L' then
      if (xx.L is null) OR (xx.L < nvl(xx.M,0)) then
        outregion := 'Y' ;
      end if ;
    end if ;
    if xx.region = 'M' then
       if xx.M is null then
         outregion := 'Y' ;
       elsif xx.M = 1 then
         outregion := null ;
       elsif xx.L is null and xx.M < xx.H then
         outregion := 'Y' ;
       elsif xx.H is null and xx.M < xx.L then
         outregion := 'Y' ;
       end if ;
    end if ;
    if xx.region = 'H' then
       if (xx.H is null) OR (xx.H < nvl(xx.M,0)) then
         outregion := 'Y' ;
       end if ;
```

```
    end if ;
    if outregion = 'Y' then
       UPDATE T2
        SET SILINECEK = 'Y'
    WHERE t2.cid = xx.cid and t2.tid = xx.tid and t2.item =
xx.item ;
    end if ;
  end loop ;
```

Burada *t2* tablosunda SILINECEK alanı Y olanlar, *t2* tablosundan atılacak demektir. Bu işlemin sonucunda oluşan *t2* tablosu Tablo 4.11'de gösterilmiştir.

Tablo 4.11. Referans bölgesinin dışında kalan işlem satırlarını gösteren *t2* tablosu

TID	ITEM	CID	T_DATE	AMOUNT	L	M	H	SILINECEK
1	b	1	05/05/2009	2	,8	,2	-	-
1	c	1	05/05/2009	3	,6	,4	-	Y
2	e	1	17/05/2009	3	,6	,4	-	Y
3	d	2	13/05/2009	5	,2	,8	-	-
4	b	2	19/05/2009	2	,8	,2	-	-
4	c	2	19/05/2009	4	,4	,6	-	-
5	d	2	21/05/2009	9	-	,4	,6	Y
6	b	3	07/05/2009	8	-	,6	,4	Y
7	d	3	15/05/2009	7	-	,8	,2	-
7	e	3	15/05/2009	9	-	,4	,6	-
8	b	4	11/05/2009	1	1	-	-	-
8	c	4	11/05/2009	3	,6	,4	-	Y
9	c	4	16/05/2009	8	-	,6	,4	-
10	d	4	23/05/2009	7	-	,8	,2	-
10	e	4	23/05/2009	10	-	,2	,8	-
11	b	4	30/05/2009	1	1	-	-	-
12	d	5	03/05/2009	6	-	1	-	-
12	e	5	03/05/2009	12	-	-	1	-
13	b	5	13/05/2009	2	,8	,2	-	-
13	c	5	13/05/2009	4	,4	,6	-	-
14	c	5	17/05/2009	6	-	1	-	-
15	d	6	19/05/2009	6	-	1	-	-
15	e	6	19/05/2009	10	-	,2	,8	-
16	b	6	25/05/2009	3	,6	,4	-	-

t2 tablosu yukarıdaki şekilde sadeleştirildikten sonra, artık *t2* tablosundan sıralı örüntüler çıkarılmaya başlanabilir.

Bunun için daha önce de belirtildiği gibi yeni başlangıç tablosu olan geniş öğe listesi ele alınacaktır. Bölüm 3.2’ de anlatılan FP ağacından sık tekrarlan yaygın örüntüler ile *f* frekans tablosundaki bir uzunluklu yaygın öğelerin birleştirilmesi sonucu elde edilen ve aşağıda gösterilen Geniş öğe listesi Tablo 4.6 ile daha önce verilmişti.

ITEMSETS	CODES
b	1
c	2
d	3
e	4
d:e	5
b:c	6

Elde kalan yeni işlem veri kümesi dikkate alınarak, geniş öğe kümeleri listesinden üretilen aday sıralı örüntüler *t2* tablosu üzerinden test edilerek, her aday örüntünün destek sayısı bulunur ve destek sayısını aşan örüntüler sıralı örüntüler olarak kaydedilir. Geniş öğe kümeleri listesi oluşturulurken, tablo üzerinde yer alan her öğe grubunu temsilen sıralı bir şekilde kod sıra numarası atanır. Aday örüntüler bu kod üzerinden üretilerek test edilir. Üretilen aday örüntüler; (1 : 1), (1 : 2), (1 : 3)…(2 : 1)…(2 : 6)……(6 : 6) şeklinde olacaktır.

Sıralı örüntülerin çıkarılmaya başlanmasındaki ilk adım olan, geniş öğe listesinden elde edilen aday örüntüler aşağıda verilen kod ile üretilir ve comb diye adlandırılan bir veri tabanı tablosu üzerinde tutulur.

```
INSERT INTO comb
SELECT ltrim(replace(sys_connect_by_path(codes,','),',',':'),':')
comb,level
      FROM new_header
      WHERE level > 1
      CONNECT BY    level =2 ;
```

aday örüntüler için üretilen kodların veri tabanı tablosu üzerindeki kısmi gösterimi Tablo 4.12’ de verilmiştir.

Tablo 4.12. Geniş öğe listesinden üretilen aday örüntüler

RESULT
1:1
1:2
1:3
1:4
1:5
1:6
2:1
2:2
2:3
2:4
2:5
2:6

Kod bazında üretilen bu aday kodların geniş öğe listesinde karşılık gelen öğe grupları sıralı ele alınarak sıralı bir şekilde *t2* tablosu üzerinde test edilir. Destek sayısını aşan örüntüler, sıralı örüntü olarak kaydedilir. Elde edilen sıralı örüntülerden ikinci bir geniş öğe listesi oluşturulur. Oluşturulan bu yeni geniş öğe listesinden yukarıda anlatıldığı gibi yeniden aday örüntüler üretilir. Üretilen bu ikinci aşamadaki aday örüntüler de tekrar test edilir. Test sonucunda destek sayısını aşan örüntüler varsa aynı işlemler üçüncü aşama için yapılır. Bu işlemler, herhangi bir aşamada testi geçen aday örüntü olmadığı zaman sona erer.

Destek sayısı 2 olarak kabul edilmişti. Buna göre kod bazında üretilen aday örüntülerin, çözümlenmesi, test edilmesi ve destek sayısını aşan örüntülerin kaydedilmesi için aşağıda kod kullanılır.

```
cursor c is
  select result from comb;

 cursor fc(res varchar2) is
     select to_number(REGEXP_SUBSTR (res,'[^:]+', 1, level)) r
      from dual
      connect by level <= length(regexp_replace(res ,'[^:]*'))+1 ;

grp number(3) ;
cnt number(3) ;
itemset1 varchar(255) ;
itemset2 varchar2(255) ;
```

```
PROCEDURE CHECK_IT AS
cursor m is
 WITH    later    AS
 (
      SELECT cid,  t_date
      FROM    t2
      WHERE   item IN (select item from k2) and silinecek is null
      GROUP BY  cid, t_date
      HAVING COUNT (DISTINCT item) = (select count(item) from k2)
  )
SELECT  e.cid,
        e.t_date  AS e_t_date,
        l.t_date  AS l_t_date
FROM  t2          e
JOIN  later       l     ON    e.cid = l.cid
                        AND   e.t_date    < l.t_date
WHERE   e.item    IN (select item from k1) and silinecek is null
GROUP BY e.cid,e.t_date,l.t_date
HAVING COUNT( DISTINCT item) = (select count(item) from k1) ;

BEGIN
  cnt := 0 ;
  for mm in m loop
     cnt := cnt + 1 ;
  end loop ;
END CHECK_IT ;
BEGIN
for cc in c loop
   grp := 0 ;
   delete from k1 ; delete from k2;
  for ffc in fc(cc.result) loop
    grp:= grp + 1 ;
    if grp = 1 then
     select itemsets into itemset1 from new_header where codes = ffc.r ;
     insert into k1
       select REGEXP_SUBSTR (itemset1,'[^:]+', 1, level)
        from dual
        connect by level <= length(regexp_replace(itemset1 ,'[^:]*'))+1 ;
     end if ;
     if grp = 2 then
      select itemsets into itemset2 from new_header where codes = ffc.r ;
       insert into k2
       select REGEXP_SUBSTR (itemset2,'[^:]+', 1, level)
        from dual
        connect by level <= length(regexp_replace(itemset2 ,'[^:]*'))+1 ;
     end if ;
   end loop ;
 cnt := 0 ;
 CHECK_IT ;
 if cnt > 2 then -- DESTEK SAYISINI AŞAN SIRALI ÖRÜNTÜ SEÇİMİ
   INSERT INTO SEQUENCES VALUES(itemset1,itemset2,cc.result) ;
  commit ;
 end if ;
 end loop ;
END ;
```

Sıralı örüntülerde sıranın önemli olduğu daha belirtilmişti. Bundan dolayı yukarıda verilen kodda görüldüğü gibi, bir müşteri tarafından alınan öğe grubu k1 tablosuna, aynı

müşteri tarafından farklı bir zamanda alınan öğe grubu ise k2 tablosuna atandıktan sonra test yapılır. Bu işlemler sonucunda elde edilen sıralı örüntüler Tablo 4.13' te gösterildiği gibi olur.

Tablo 4.13. Sıralı örüntüler

S1	S2	SEQ
d	b:c	3:6
e	b	4:1
e	c	4:2
d:e	b	5:1
d:e	c	5:2
d	b	3:1
d	c	3:2

Tablo.4.13' te görüldüğü gibi bazı sıralı örüntüler, diğer sıralı örüntüleri kapsamaktadır. Bundan dolayı sadeleştirme yapmak gerekiyor. Örneğin 3:1 örüntüsü, 5:1 örüntüsüne ait olduğu için, 3:1 örüntüsü bu tablodan çıkarılacaktır. Çünkü 3:1 örüntüsünde yer alan "*d* alındıktan sonra *b* alınmaktadır" kuralı, 5:1 örüntüsündeki "*d* ve *e* alındıktan sonra *b* alınmaktadır" kuralın içinde mevcuttur. Bu şekilde sadeleştirme yapıldıktan sonra Tablo.4.14' te gösterilen sıralı örüntüler elde edilir.

Tablo 4.14. Sadeleştirme yapıldıktan sonra elde edilen sıralı örüntüler

S1	S2	SEQ
d	b:c	3:6
d:e	b	5:1
d:e	c	5:2

5. UYGULAMA SONUÇLARI

Bu tez çalışmasında nicel değerli verilerden sıralı örüntüleri elde etmek için Dicle Üniversitesi Tıp Fakültesi Araştırma Hastanesinin bir aylık Merkez Laboratuar hasta tahlil verileri kullanılmıştır. Toplam 6580 farklı hastaya, 156099 adet tahlil yapılmıştır. Laboratuarda kanların tahlili için kullanılan 40'ın üzerinde tahlil grubu vardır. Bu gruplardan bazıları Tablo 5.1 ile gösterilmiştir. Hastalardan alınan kanların tahlilleri hastalık durumuna göre çok değişik gruplarda yapılabilmektedir.

Tablo 5.1. Tahlil grupları

Biyokimya
İdrar (Rutin)
Hematoloji (Rutin)
Hormon
TROMBOSİT FONKSİYON TESTLERİ
Hematoloji (Acil Lab.)
Kan Gazları (Acil Lab.)
Sedimantasyon
Bakteriyoloji

Bu tez çalışmasında sadece Rutin Biyokimya tahlil verileri kullanılmıştır. Biyokimya grubu altında çalışılan yaklaşık 66 parametre vardır. Bu parametreler bazıları Tablo 5.2 ile gösterilmiştir.

Tablo 5.2. Biyokimya tahlil parametreleri

FOSFOR
HDL-C
ÜRIK ASIT
GLUKOZ
Na
ALP
DEMIR BAGLAMA K.
INDIREKT BILIRUBIN
CK-MB
KREATININ(Idrar)
KALSIYUM(Idrar)
GLUKOZ (Bos)
KALSIYUM (24H-IDRARDA)
DIREKT BILiRUBIN

Bu parametreler ile Karaciğer, Pankreas, Böbrek ve Kalp gibi hasar görmüş organlardan kana geçen Ck, Fosfor, AST, ALT v.b. kimyasal maddelerin ölçümü yapılır [29]. Bu testler ile hasar tespiti yapılabildiği gibi aynı zamanda tedavi görmüş bir hastanın, tedaviye yanıt verip vermediğini anlamak için de yapılır [29].

Kullanılan örnek Biyokimya tahlil verilerinin bazı nitelikleri Tablo 5.3 ile gösterilmiştir.

Tablo 5.3. Örnek Biyokimya tahlil verileri

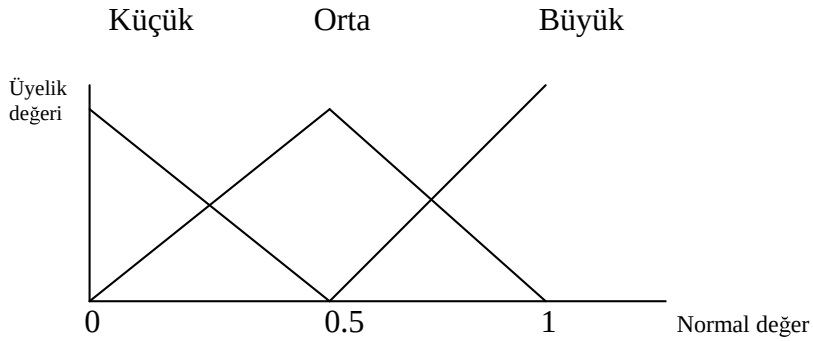
T_SICIL	CINSIYET	DOGTAR	ISLEMTAR	TEST_ADI	R_DEGER	A_LIM	U_LIM
2009004844	E	12/09/1996	22/01/2009	FOSFOR	5,2	0	0
2009004844	E	12/09/1996	22/01/2009	TOTAL BILIRUBIN	,7	,2	1
2009004844	E	12/09/1996	22/01/2009	DIREKT BILiRUBIN	,3	0	0
2009004844	E	12/09/1996	22/01/2009	AST	28	10	40
2009004844	E	12/09/1996	22/01/2009	ALT	12	10	35
2009004844	E	12/09/1996	22/01/2009	T.PROTEIN	7,7	6,4	8,3
2009004844	E	12/09/1996	22/01/2009	ALBUMIN	4,5	3,5	5
2009004844	E	12/09/1996	22/01/2009	GLOBULIN	3,2	2,6	3,7
2009004844	E	12/09/1996	22/01/2009	TRIGLISERID	74	50	180
2009004844	E	12/09/1996	22/01/2009	KOLESTEROL	170	112	200
2009004844	E	12/09/1996	22/01/2009	HDL-C	62	0	0
2009004844	E	12/09/1996	22/01/2009	LDL-C	93,2	0	0
2009004844	E	12/09/1996	22/01/2009	ÜRIK ASIT	3,1	0	0
2001048803	K	01/01/1973	22/01/2009	ÜRE (24H-IDRARDA)	447	-100000	100000
2001048803	K	01/01/1973	22/01/2009	KREATININ (24H-IDRARDA)	35,91	-100000	100000

Bu tablodan işlem veri kümesi oluşturmak için, T_SICIL, ISLEMTAR, TEST_ADI ve R_DEGER nitelikleri kullanılacaktır. TEST_ADI Biyokimya altında yapılan parametrelerdir. R_DEGER niteliği tahlil sonucunu göstermektedir. Bu tahlil sonucu nicel değer olarak ele alınacaktır. Ancak tabloda görüldüğü gibi test için çalışılan parametrelerin alt ve üst limitleri aynı aralıkta değildir. Dolayısıyla tahlil sonuçlarının değer aralıkları birbirlerinden çok farklı olabilmektedir..Bundan dolayı bulanık değerleri elde etmek için tahlil sonuçlarının, alt ve üst limit değerleri kullanılarak normalize edilmesi gerekir. Normalize etmek için kullanılacak yöntem şu şekilde olacaktır:

X, test parametresi olsun. Bir hastanın X parametre tahlil sonucu aşağıdaki formülle normalize edilebilir :

Normalize edilen X = (Xgercek-Xmin) / (Xmax-Xmin)

Burada Xgercek hastanın bu teste ait tahlil sonucu, Xmax testin üst limiti, Xmin ise testin alt limitidir. Örneğin bir X test parametresinin üst limit 319, alt limiti de 0,3 olsun. Bir hastanın da X tahlilin sonucu da 0,3 olsun. Üyelik fonksiyonu olarak da Şekil.5.1 alınacaktır.



Şekil 5.1. Uygulamada kullanılacak üyelik fonksiyonu

Hastanın 0.3 olan tahlil sonucu Xgerçek değeri olur. Normalize edilen X değer :

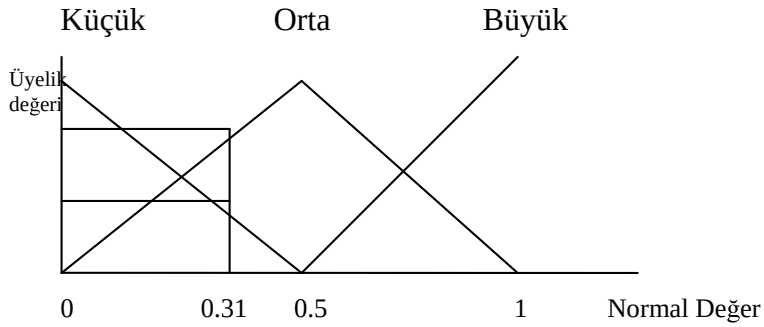
X = (0.3-0.3) / (319-0.3) = 0 olur. Bu değer bulanık kümelerde 0 sıfır değerine normalize edildi.

Diğer bir hastanın X değeri yani Xgercek=319 olsun. O zaman;

Normalize edilen X= (319- 0.3) / (319 – 0.3) = 1. Yani veritabanındaki en büyük değer olan 319 sayısı bulanık kümelerde en büyük değer olan 1'e normalize edildi.

Son örnek başka bir hastanın X değeri yani Xgercek=100 olsun. O zaman;

Normalize edilen X = (100 – 0.3) / (319 – 0.3) = 0.31'e karsılık gelir. 0.31'i yukarıdaki şekilde yerine koyarsak, sonuç Şekil 5.2' de görüldüğü gibi olur.;



Şekil 5.2. Normalize edilen değerin gösterimi

Küçük üyelik fonksiyonunu = 0.4 oranında, Orta üyelik fonksiyonunu da 0,6 oranında kesecektir.

Uygulamada veritabanı olarak Oracle 10gXE veritabanı kullanılmıştır. Uygulama dili olarak PL/SQL ve SQL kullanılmıştır. Uygulama için Pentium4 3.00 Ghz, 2GB Ram olan bir bilgisayar kullanılmıştır.

Sıralı örüntülerin çıkarılmasına başlanmadan evvel, işlem veri kümesi olan *t* tablosunun oluşturulması gerekir. Bunun için Tablo 5.3 ile gösterilen Biyokimya test verilerinin hareket sıraları oluşturulduktan sonra, nicel değer olarak kabul edilen test sonuçları yukarıda anlatıldığı şekilde normalize edilir. Bu işlemlerden sonra oluşan *t* tablosu Tablo 5.4 ile gösterildiği gibi olur.

Tablo 5.4. Biyokimya test verileri baz alınarak oluşturulan *t* tablosu

TID	ITEM	CID	T_DATE	AMOUNT	NORMAL	ALIMIT	ULIMIT
3764	GLU	2007080497	12/01/2009	82	,1	0	800
3764	IBCT	2007080497	12/01/2009	292	,44	0	660
3764	KREA	2007080497	12/01/2009	,7	,02	0	37
3764	LDH	2007080497	12/01/2009	244	,05	0	4732
3764	Na	2007080497	12/01/2009	136	,72	0	189
3764	PHOS	2007080497	12/01/2009	3,6	,1	0	37
3764	PROT	2007080497	12/01/2009	6,6	0	0	2041
3764	URE	2007080497	12/01/2009	42	,11	0	381
3765	ALP	2007080570	30/01/2009	282	,14	40	1713
3765	ALT	2007080570	30/01/2009	11	0	10	2778
3765	AST	2007080570	30/01/2009	22	0	10	3683
3765	GLU	2007080570	30/01/2009	85	,02	70	800
3765	K	2007080570	30/01/2009	3,4	0	3,5	37
3765	KREA	2007080570	30/01/2009	,7	0	,6	37
3765	LDH	2007080570	30/01/2009	244	,03	125	4732

Yukarıdaki veriler dikkate alınarak, çalışmalar farklı destek sayıları seçilerek yapıldı. Destek sayısının 1000 seçilmesi halinde 9448 adet yaygın öğeler grubu elde edildi. Elde edilen 9448 kuralın bazıları Tablo 5.5 ile gösterilmiştir.

Tablo 5.5. Destek sayısının 1000 seçilmesi halinde elde edilen yaygın öğeler grubu

PATTERN	CNT	PB_ITEM
GLB:GLU:URE:LDH	1080	LDH
GLB:KREA:Na:LDH	1080	LDH
GLB:KREA:Na:PROT:LDH	1080	LDH
GLB:KREA:Na:PROT:URE:LDH	1080	LDH
GLB:KREA:Na:URE:LDH	1080	LDH
GLB:Na:PROT:URE:LDH	1080	LDH
ALB:ALT:AST:GLU:KREA:Na:LDH	1079	LDH
ALB:ALT:AST:GLU:KREA:Na:URE:LDH	1079	LDH
ALB:ALT:AST:GLU:Na:LDH	1079	LDH
ALB:ALT:AST:GLU:Na:URE:LDH	1079	LDH
KREA:URE:CA	1350	CA
Na:CA	1315	CA
Na:URE:CA	1313	CA
KREA:Na:CA	1312	CA
KREA:Na:URE:CA	1312	CA
K:CA	1310	CA
K:Na:CA	1308	CA
K:URE:CA	1307	CA
K:Na:URE:CA	1306	CA
K:KREA:CA	1305	CA
ALT:AST:GLU:K:Na:URE:LDH	1166	LDH
ALT:AST:GLU:K:URE:LDH	1166	LDH
AST:GLU:K:KREA:LDH	1166	LDH
AST:GLU:K:KREA:Na:LDH	1166	LDH
AST:GLU:K:KREA:Na:URE:LDH	1166	LDH
AST:GLU:K:KREA:URE:LDH	1166	LDH
AST:GLU:K:LDH	1166	LDH
AST:GLU:K:Na:LDH	1166	LDH
AST:GLU:K:Na:URE:LDH	1166	LDH

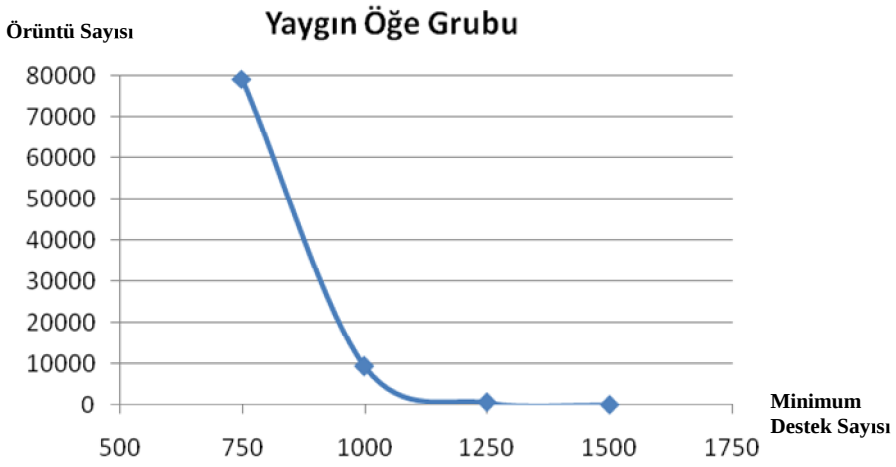
Sıralı örüntüler için elde edilen geniş öğe listesi Tablo 5.6 ile gösterilmiştir. Bu tezin sonraki uygulamasında farklı minimum destek değerleri için elde edilen yaygın öğe grupları Şekil 5.3'de verilmiştir. Şekilde de gösterildiği gibi minimum destek değeri artıkça yaygın öğe grubu sayısı hızlı bir şekilde azalmaktadır.

Diğer bir uygulama da ise yine farklı minimum destek değerleri için önerilen yöntemin ve FP-Growth algoritmasının yürütme zamanlarının karşılaştırılması yapılmıştır. Şekil 5.4'de

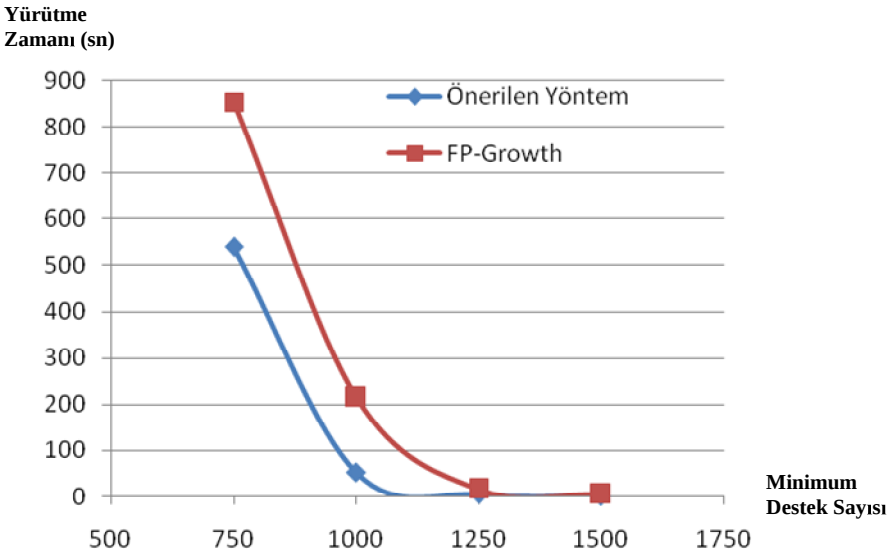
gösterildiği gibi önerilen yöntemin yürütme zamanı FP-Growth algoritmasınınkine göre daha düşüktür.

Tablo 5.6. Destek sayısının 1000 olması durumunda elde edilen geniş öğe listesi

ITEMSETS	CODES
ALB:AST:GLU:KREA:Na:LDH	500
ALB:ALT:PROT:LDH	255
ALB:Na:LDH	256
ALT:AST:GLB:LDH	257
ALT:AST:GLB:PROT:LDH	258
ALT:GLB:LDH	259
ALT:GLB:PROT:LDH	260
ALB:ALT:AST:GLB:LDH	261
ALB:ALT:AST:GLB:PROT:LDH	262
ALB:ALT:GLB:LDH	263
ALB:ALT:GLB:PROT:LDH	264
ALB:AST:Na:LDH	265
ALB:KREA:Na:LDH	266
ALB:KREA:Na:URE:LDH	267
ALB:Na:URE:LDH	268



Şekil 5.3. Farklı minimum destek değerlerinde elde edilen yaygın öğe grubu sayıları



Şekil 5.4. Farklı minimum destek değerlerinde önerilen yöntemin ve FP-Growth algoritmasının yürütme zamanları

Bu tez çalışmasında Dicle Üniversitesi Araştırma Hastanesine gelen kişilerin Merkez laboratuar Rutin Biyokimya sonuçları kullanılmıştır. A testini isteyenler beraberinde B testini de istiyorlarmış şeklinde bir yaklaşımla beraber, sıralı örüntülerde hastaların bir sonraki gelişlerinde yaptırdıkları testlerde belirlenmiştir.

Uygulama sonuçları değerlendirildiğinde öne çıkan sonuçlar şunlardır, Karaciğer fonksiyon testleri istenen hastalarda aynı zamanda böbrek fonksiyon testleri de istenmiştir. Böbrek fonksiyon testleri istenen hastalar, ikinci gelişlerinde tekrar böbrek fonksiyon testi yaptırmıştır. Albümin, globülin ve total proteinin üçü bir arada istenmiştir.

6. SONUÇ VE DEĞERLENDİRME

Sıralı örüntülerin keşfedilmesi için daha önce önerilen yöntemlerden biri ve belki de en etkilisi FP-Growth algoritması ile olanıdır. Ancak bu yöntemde bile bazı sorunlar yaşanmaktadır. Bunlardan biri bu algoritma için çok karmaşık veri yapısı oluşturmak zorunda kalınması diğeri ise yaygın örüntülerin çıkarılması için gerekli olan alt ağaçların oluşturulmasında ve dolaşılmasında bile öz yineli işlemlere gerek duyulmasıdır. Ayrıca klasik FP-Growth algoritmasıyla şimdiye kadar hep ikili değer alan veritabanlarından yaygın öğelerin bulunmasına çalışılmıştır. Fakat günümüzdeki veritabanların çoğunluğunun nicel değer içeren niteliklerden oluştuğu göz önüne alınırsa bu algoritmanın ilgili veritabanlarına uyarılma zorunluluğu da kolayca ortaya çıkar.

Bu tezde yukarıdaki problemlere çözüm üretmek için FP-Growth tabanlı bir algoritma önerilmiştir. Algoritmanın üstünlüğü özyinelemeli yerine öğe tabanlı aday küme üretme ve daha basit veri yapısı kullanmasıdır. Önerilen yöntem daha sonra nicel değerli veri tabanlardan sıralı örüntülerin bulunması için kullanılmıştır. Veritabanı olarak Dicle Üniversitesi Tıp Fakültesi Merkez Laboratuarındaki hastalık veri kümeleri alınmıştır.

Deneysel tecrübelerden elde edilen sonuçlarda önerilen yöntemin FP-Growth ile aynı sayıda yaygın öğe kümesi bulurken, FP-Growth'a göre daha az bellek ve zaman harcadığı görülmüştür.

Bu tez çalışmasında hastane laboratuarlarında veri madenciliği uygulamalarının temel kullanım alanı, analiz karakteristiklerini elde ederek laboratuar verilerinin karar verme sürecinde laboratuar stratejilerinin belirlenmesi ve nihai olarak performansın artırılmasıdır.

Gelecekte yapılacak çalışmalarda, rutin biyokimya testleri haricinde merkez laboratuarının tümünde yapılan testler değerlendirilerek daha geniş daha etkin sonuçlara ulaşılması mümkündür. Ayrıca daha sonraki çalışmalarda sıralı örüntülerin çevrim-içi keşfedilmesi için yeni yöntemler geliştirilebilir. Çünkü önerilen yöntem bu yönde de ilerleme için yeni veri yapıları geliştirmiştir.

KAYNAKLAR

[1] **Wu Y. H. and.Chen, A.L.P**, 2002. Prediction Of Web Page Accesses By Proxy Server Log, *In World Wide Web: Internet And Web Information Systems*, **5[1]:**67-88.

[2] **Hatonen, K., Klemettinen, M., Mannila, H., Ronkainen, P. And Toivonen, H**., 1996. Knowledge Discovery From Telecommunication Network Alarm Databases, *In12the Intl. Conf. Data Engineering.*

[3] **Bonfield, J.K. and Staden, R.**, 2002. Ztr: A New Format For Dna Sequence Trace Data, *Bioinformatics*, **18(1):**3-10.

[4] **Han. J., Pei, J., Yin Y**., 2000. Mining Frequent Patterns Without Gandidate Generation, *Proc. of Acm Int. Conf on Management of Data*, 1-12.

[5] **Agrawal R, Imielinski T, Swami A.,** 1993. Mining Association Rules Between Sets of Items In Large Databases. *In Proceedings of the 1993 ACM- Sigmod international Conference on Management of Data*, Washington DC, S. 207–216.

[6] **Han, J., and Kamber, M.,** 2006. Data Mining: Concepts And Techniques, Second Edition.

[7] **Bayardo, R.,** 1998. Efficiently Mining Long Patterns From Databases. *Proc. of' Acm Int. Conf. on Management of Data*, Sayfa 85-93.

[8] **Yang,J., Wang, W., Yu, P.S., Han, J.**, 2002. Mining Long Sequential Patterns In Noisy Environment. *Sigmod Conference* 2002, Sayfa 406-417.

[9] **Savasere, A., Omiecinski, E., Navathe, S.,** 1995. An Efficient Algorithm For Mining Association Rules In Large Databases. *In.Vldb'95*, Sayfa. 432-443.

[10] **Chen, Y.L., Huang, C.K.,** 2006. A New Approach For Discovering Fuzzy Quantitative Sequential Patterns in Sequence Databases, *Fuzzy Sets and Systems*, **157:12**, Pages: 1641-1661.

[11] **Adnan, M., Alhajj, R., and Barker, K.,** 2006. Constructing Complete Fp-Tree For Incremental Mining Of Frequent Patterns In Dynamic Databases, *LNCS*, Volume 4031, 363-372.

[12] **Pei, J.,** 2002 Pattern-Growth Methods For Frequent Pattern Mining, *Phd thesis*, Simon Fraser University.

[13] **Agrawal, R. and Srikant, R.,** 1994. Fast Algorithms For Mining Association Rules. *In Proc. Int. Conf. Very Large Data Base*, Sayfa 487–499.

[14] **Han, J., Pei, J., Yin, Y. and Mao, R.,** 2004. Mining Frequent Patterns Without Candidate Generation: A Frequent-Pattern Tree Approach, *Data Mining And Knowledge Discovery*, **8**, 53-87.

[15] **Agrawal R.and Srikant R.,** 1995. Mining Sequential Patterns, *IEEE International Conference on Data Engineering* 3-14.

[16] **Srikant, R., and Agrawal, R.,** 1996. Mining Sequential Patterns: Generalizations and Performance Improvements, *EDBT*.

[17] **Hong, T. P. and Lee, C.Y.,** 1996. Induction of Fizzy Rules And Membership Functions from Training Examples, *Fuzzy Sets And Systems*, Vol. **84**, 33-47.

[18] **Zimmermann, H.J.,** 1991. Fuzzy Set Theory and Its Applications, Kluwer Academic Publisher.

[19] **Hong T.P. and Chen, J.B.,** 1999. Finding Relevant Attributes and Membership Functions, *Fuzzy Sets And Systems*, Vol. **103**, No**. 3**, 389-404.

[20] **Eker, H.,** 2006. Veri Madenciliği veya Bilgi Keşfi, İnsan Kaynakları Akademisi.

[21] **Tan, P.N., Steinbach, M., and Kumar, V.,** 2006. Introduction To Data Mining: Association Analysis:Basic Concepts And Algorithms, Chapter 6.

[22] **Verhein, F.,** 2008. Frequent Pattern Growth (Fp-Growth) Algorithm Outline, *Lecture Notes*, University of Sydney.

[23] Oracle Database Sql Reference, 10g Release 1 (10.1), Part No. B10759-01.

[24] Oracle PL/SQL User's Guide And Reference, 10g Release 1 (10.1), Part No. B10807-01.

[25] **Shang, X., Sattler, K.U., Geist, I.,** 2005. Sql Based Frequent Pattern Mining With Fp-Growth, *LNCS*, Volume 3392, 32-46.

[26] **Hong, T.Z., Lin, K.Y., and Wang, S.L.,** 2001. Mining Fuzzy Sequential Patterns From Multiple-Item Transactions, *IFSA World Congress and 20th NAFIPS Int'l con]erence*, Vol. **3**, pp. 1317-1321.

[27] **Lin, C.W., Hong, T.P., Lu, W. H.,** 2009. Linguistic Data Mining with Fuzzy FP-Trees, *Expert Systems with Applications*, 10.1016/J.Eswa.2009.12.052.

[28] **Shang, X.,** 2004. SQL based frequent pattern mining without candidate generation, *Proceedings of the 2004 ACM symposium on Applied computing*, Pages: 618 – 619.

[29] **Erbil, M. K. ve Saruhan, E.,** 2005. Laboratuar El Kitabı, Ankara.

ÖZGEÇMİŞ

14.03.1969 Diyarbakır doğumluyum. Bir yıl Dicle Üniversitesi Fen-Edebiyat Fakültesi Matematik bölümünde, bir yıl da İ.T.Ü. Endüstri Mühendisliğinde okuduktan sonra 1991–92 öğretim yılında Hacettepe Üniversitesi Bilgisayar Mühendisliğine girdim. 1997 yılında bu bölümden mezun oldum. 2000 yılında, evlendikten sonra Dicle Üniversitesi Tıp Fakültesinde Okutman olarak göreve başladım. Bu görevim halen sürmektedir.