

YILDIZ TEKNİK ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ

Veri Mad. Apr. Tem. İli. Kur,
Alg. Uyg. ve Karş.

YÜKSEK LİSANS TEZİ

Filiz Koyuncu

2004

2 568 102

**YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

**VERİ MADENCİLİĞİNDE APİRİORİ TEMELLİ
İLİŞKİLENDİRME KURALI ALGORİTMALARININ
UYGULAMA VE KARŞILAŞTIRMASI**

Bilgisayar Müh. Filiz KOYUNCU

**FBE Bilgisayar Mühendisliği Anabilim Dalında
Hazırlanan**

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Prof. Dr. Oya KALIPSIZ

İSTANBUL, 2004

YILDIZ TEKNİK ÜNİVERSİTESİ
KÜTÜPHANE VE DOKÜMANTASYON
DAİRE BAŞKANLIĞI

Yer No (DDC) : R/368/102

Kayıt No : 2159
Geldiği Yer : Fen Bilimleri Enzt.
Tarih : 19/10/2004
Fiyat : 5.650.000
Fatura No :
Ayniyat No : 1/13
Ek :



XIV, 113

**YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ**

**VERİ MADENCİLİĞİNDE APRİORİ TEMELLİ
İLİŞKİLENDİRME KURALI ALGORİTMALARININ
UYGULAMA VE KARŞILAŞTIRMASI**

Bilgisayar Müh. Filiz KOYUNCU

FBE Bilgisayar Mühendisliği Anabilim Dalında
Hazırlanan

YÜKSEK LİSANS TEZİ

Tez Danışmanı: Prof. Dr. Oya KALIPSIZ

İSTANBUL, 2004

Prof. Dr. Mustafa Uyş.

A. C. Sonner.

Prof. Dr. Oya Kalipsiz

İÇİNDEKİLER

	Sayfa
SİMGE LİSTESİ	v
KISALTMA LİSTESİ	vi
TERİMLER SÖZLÜĞÜ	vii
ŞEKİL LİSTESİ	ix
ÇİZELGE LİSTESİ	xi
ÖNSÖZ	xii
ÖZET	xiii
ABSTRACT	xiv
1. GİRİŞ	1
2. VERİ MADENCİLİĞİ	2
2.1 Veri Madenciliği Nedir?	2
2.2 Veri Madenciliği Teknikleri	4
2.3 Kullanılan Teknikler	5
2.4 Veri Ambarı	6
2.4.1 Bir Veri Ambarı Yapısı	8
2.5 OLAP	10
2.5.1 OLAP Sisteminin İşlevsel Gereksinimleri	11
2.5.2 Temel Çok Boyutlu OLAP Özellikleri	11
2.6 Metodoloji	12
3. İLİŞKİSEL KURAL MADENCİLİĞİ	13
3.1 Temel Kavramlar	13
3.2 Problem Tanımı	18
3.3 Arama Alanı	19
3.3.1 Arama alanınının Üstünden Geçme (Traverse)	19
3.3.2 Kalem Kümelerinin Desteklerinin Belirlenmesi	20
3.3.3 Sistematikleştirme	21
3.3.4 BFS ve Oluşum Sayma	21
3.3.5 BFS ve TID-Listesi Kesişimleri	22
3.3.6 DFS ve Oluşumları Sayma	22
3.3.7 DFS ve TID-Liste Kesişimleri	23
3.4 Sepet Analizi	23
3.5 İlişkisel Kuralların Türleri	24
3.5.1 Hiyerarşik İlişkisel Kurallar	24

3.5.2	Çok Seviyeli İlişkisel Kural	26
3.5.3	Negatif İlişkisel Kural	27
3.5.4	Nicel (Quantitative) İlişkisel Kural	29
4.	İLİŞKİSEL KURAL MADENCİLİĞİ ALGORİTMALARI.....	31
4.1	Apriori Temelli Algoritmalar	31
4.2	Diğer Algoritmalar	34
4.2.1	FP-Growth	34
4.2.1.1	FP-Ağaç Yapısı	34
4.2.1.2	FP-Ağaç Yapısının Madenlenmesi Adımları	36
4.2.2	CHARM	39
4.2.2.1	CHARM Algoritma Tasarımı.....	40
4.2.2.2	Pseudo-Kod Tanımı.....	41
4.2.3	Eclat.....	43
4.2.4	Closet.....	44
4.2.4.1	Kapalı Sık Kalem Kümesi	45
4.2.4.2	Kapalı Sık Kalem Kümeleri Üzerinde İlişkisel Kural.....	46
4.2.4.3	Kapalı Sık Kalem Kümelerinin Madenciliği.....	48
4.2.4.4	CLOSET Algoritması.....	51
4.2.5	Partitioning	52
4.2.6	Paralel İlişkisel Kural Madenciliği Algoritmaları	54
4.3	Algoritmaların Deneysel Olarak Karşılaştırılması Örnekleri.....	55
5.	APRIORI TEMELLİ ALGORİTMALARIN GERÇEKLEŞTİRİLEN UYGULAMALARA GÖRE KARŞILAŞTIRILMASI	62
5.1	Apriori Temelli Algoritmaların Yapısının İncelenmesi	62
5.1.1	AIS.....	63
5.1.2	SETM	63
5.1.3	Apriori Algoritması	64
5.1.3.1	Sık Kalem Kümelerinin Madenlenmesi (Temel Adım)	64
5.1.3.2	Aday Kalem Kümeleri Oluşturma Adımı	65
5.1.3.3	Aday Kalem Kümesi Oluşturma Örnekleri	69
5.1.3.4	Sık Kalem Kümelerinden İlişkisel Kurallar Üretme	71
5.1.4	En İyileleme (Optimizasyon)	71
5.1.4.1	AprioriTid Algoritması.....	71
5.1.4.2	AprioriHybrid Algoritması.....	74
5.1.5	Doğrudan HDA ve Kısaltma (DHP)	75
5.1.5.1	Adayları Süzme	76
5.1.5.2	Veri tabanını Kısaltma (Trimming).....	77
5.1.5.3	DHP Algoritması:.....	78
5.2	Veri Hazırlama	81
5.2.1	Tez Çalışmasında Kullanılan Veri Ambarları	81
5.2.1.1	Akköprü Verisi	81
5.2.1.2	BMS-WebView-1 ve BMS-WebView-2 Verileri	83
5.3	Apriori Temelli Algoritmaların Karşılaştırılması.....	85
5.3.1	Yöntem Farklılıkları	86
5.3.2	Performans Farklılıklarına Göre Karşılaştırma	86
5.3.2.1	Akköprü Verisi Üzerinde Performans Farklılıklarına Göre Karşılaştırma	87
5.3.2.2	BMS-WebView-1 Verisi Üzerinde Performans Farklılıklarına Göre Karşılaştırma	94
5.3.2.3	BMS-WebView-2 Verisi Üzerinde Performans Farklılıklarına Göre Karşılaştırma	100

5.3.3	Sık Kalem Kümelerine Göre Karşılaştırma.....	103
5.3.4	Eşik Destek Değerlerine Göre Karşılaştırma	103
5.3.5	Farklı Veri Kümeleri Üzerinde Karşılaştırma	103
5.4	Sık Kalem Kümelerinden İlişkisel Kural Üretme	104
5.5	Uygulanan Algoritmaların Karşılaştırma Sonuçlarının Değerlendirilmesi.....	108
6.	SONUÇ VE ÖNERİLER	110
KAYNAKLAR.....		112
ÖZGEÇMİŞ		113

SİMGE LİSTESİ

C	Hash tablosu
L	Sık kalem kümesi
D	Veri tabanı
s	Kalem Kümesi
$\lambda(s)$	s 'nin Destek Değeri
σ	Kullanıcı Tanımlı Destek Eşik Değeri
γ	Kullanıcı Tanımlı Güven Eşik Değeri
2^A	Mantıksal Kafes
HDA	Hesaba dayalı adresleme

KISALTMA LİSTESİ

Supp	Support
Minsup	Minimum Support
Conf	Confidence
Minconf	Minimum Confidence
Cas	Canonical Attribute Sequences
BFS	Breadth-First Search
DFS	Depth-First Search
DHP	Direct Hashing and Pruning
OLAP	Online Analytical Processing

Root table	Root table
One frequent itemset	One frequent itemset
Dimensionality reduction	Dimensionality reduction
Episode	Episode
Generalized association	Generalized association
Cross-marketing	Cross-marketing
Online Analytical Processing	Online Analytical Processing
Multivariate analysis	Multivariate analysis
Behavioral expert	Behavioral expert
Support	Support
Derinlik öncelikli arama	Derinlik öncelikli arama
Text mining	Text mining
Noise	Noise
Box	Box
Options menu	Options menu
K-nearest neighbor	K-nearest neighbor
Naïve	Naïve
Bayrak	Bayrak
Poster	Poster
Confidence	Confidence
Cardinality	Cardinality
Association dependency	Association dependency
Canonical attribute sequences	Canonical attribute sequences
Unification	Unification
Association	Association
Item	Item
Query	Query
Rule	Rule
Rule extraction	Rule extraction
Clustering	Clustering
Frequency of frequent itemset	Frequency of frequent itemset
Itemset	Itemset
Rule	Rule
Instance-based methods, case-based reasoning	Instance-based methods, case-based reasoning
Market basket analysis	Market basket analysis
Flattening	Flattening
Image detection	Image detection
Counter	Counter
Market analysis	Market analysis
Frequent item set	Frequent item set

TERİMLER SÖZLÜĞÜ

Hashing	Hesaba dayalı adresleme
Aday	Candidate
AIS	Agrawal, Imielinski, Swami
Alt Cas	Subcas
Ana	Parent
Azaltmak	Pruning
Aşağı yönde kapanma	Downward closure
Başvuru skorlama	Application scoring
Bellek tabanlı	Memory-based
Birinci derinlikteki tablo	Root table
Birli sık kalem kümeleri	One frequent itemsets
Boyut azaltma	Dimensionality reduction
Bölüm	Episode
Breadth-First Search	Genişlik öncelikli arama
Çapraz-pazarlama	Cross-marketing
Çevrim içi çözümsel işlem	Online Analytical Processing
Çokboyutlu analiz	Multivariate analysis
Davranış skoru	Behavioral score
Destek	Support
Depth-First Search	Derinlik öncelikli arama
Döküman madenciliği	Text mining
Düğüm	Node
Empty	Boş
En İyileme	Optimizasyon
En yakın k komşu	k-nearest neighbor
Fare	Mouse
Flag	Bayrak
Gösterge	Pointer
Güven	Confidence
Aday kalem kümesi	Candidate itemset
İlişki	Association, dependency
İlkesel özellikler dizisi	Canonical attribute sequences
İşlem	Transaction
K-kalem kümesi	K-item set
Kalem	İtem
Kapak	Cover
Kök	Root
Kural çıkarma	Rule extraction
Kümeleme	Clustering
Olası σ -sık kalem kümesi	Possibly σ -Frequent item set
Nicel	Quantitative
Önek	Prefix
Örnek tabanlı yöntemler	Instance-based methods, case-based reasoning
Pazar sepeti analizi	Market basket analysis
Rastgele hale getirme	Hashing
Sahtekarlıkların saptanmasını	Fraud detection
Sayaç	Counter
Sepet analizi	Basket analysis
Sık kalem kümesi	Frequent item set

ŞEKİL LİSTESİ

Şekil 2.5. Veri madenciliği çalışmasında kullanılan metodoloji [1]	12
Şekil 3.1. $A=\{a,b,c,d,e\}$ için farklı veri tabanı gösterimleri (Adamo, 2000)	14
Şekil 3.2. $I=\{1,2,3,4\}$ için kafes (Hipp,Güntzer,Nakhaeizadeh,2000)	19
Şekil 3.3. Algoritmaların sistematikleştirilmesi (Hipp,Güntzer,Nakhaeizadeh,2000)	21
Şekil 3.4. $I=\{1,2,3,4\}$ 'ın ağacı (Hipp,Güntzer,Nakhaeizadeh,2000)	23
Şekil 3.5 Örnek 3.3'e ait veri tabanındaki kalem ve destek değerleri şekli [5]	24
Şekil 3.6 Örnek bir sınıflandırma (Adamo,2000)	26
Şekil 4.1 Örnek 4.1'e ait FP-ağaç yapısı	35
Şekil 4.2 Şartlı Örüntü-Temelleri Yaratarak Sık Örüntüleri Madenleme (J.Han, J.Pei,Y. Yin, 1999)	37
Şekil 4.3 Sık kalem kümelerini oluşturma (Zaki,Hsiao, 1999)	39
Şekil 4.4 Tamamlanmış alt küme kafesi (Zaki,Hsiao, 1999)	40
Şekil 4.5 Kalem kümeleri ve Tid kümelerinin temel özellikleri	41
Şekil 4.6 Closet kullanılarak kapalı sık kalem kümelerini madenleme (F.C.I olarak kısaltılır) (J.Pei,J.Han,R.Mao,2000)	48
Şekil 4.7 IBM-Artificial verisinde sık kalem kümesi üreten algoritmaların çalışma süreleri..	56
Şekil 4.8 BMS-POS verisinde sık kalem kümesi üreten algoritmaların çalışma süreleri	57
Şekil 4.9 BMS-WebView-1'de sık kalem kümesi üreten algoritmaların çalışma süreleri	57
Şekil 4.10 BMS-WebView-2 verisinde sık kalem kümesi üreten algoritmaların çalışma süreleri (Zheng,Kohavi,Mason,2000)	58
Şekil 4.11 (a) T5.I2.D100K verisine ait karşılaştırma grafiği	59
Şekil 4.11 (b) T10.I2.D100K verisine ait karşılaştırma grafiği	59
Şekil 4.11 (c) T10.I2.D100K verisine ait karşılaştırma grafiği	60
Şekil 4.11 (d) T10.I4.D100K verisine ait karşılaştırma grafiği	60
Şekil 4.11 (e) T20.I4.D100K verisine ait karşılaştırma grafiği	60
Şekil 4.11 (f) T20.I6.D100K verisine ait karşılaştırma grafiği	61
Şekil 4.11 Farklı veri gruplarına göre farklı ilişkisel kural algoritmalarının karşılaştırma örnekleri (J.Hipp, U.Güntzer, G.Nakhaeizadeh, 2000)	61
Şekil 5.1 HDA ağacı: (a) şimdiki durum (b) yeni durum (Adamo, 2000)	67
Şekil 5.2. AprioriTid algoritmasının bir çalışma anındaki görüntüsü (Adamo, 2000)	74
Şekil 5.3 Akköprü verisi	82
Şekil 5.4 Düzenlenen Akköprü verisi	83
Şekil 5.5 BMS-WebView-1 verisi	84
Şekil 5.6 BMS-WebView-2 verisi	84
Şekil 5.7 Düzenlenen BMS-WebView-1 verisi	84
Şekil 5.8 Düzenlenen BMS-WebView-2 verisi	85
Şekil 5.9 Akköprü verisi üzerinde Apriori sonuçları	88
Şekil 5.10 Akköprü verisi üzerinde AprioriTid sonuçları	89
Şekil 5.11 Akköprü verisi üzerinde AprioriHybrid sonuçları	90
Şekil 5.12 Akköprü verisi üzerinde DHP sonuçları	92
Şekil 5.13 Algoritmaların Akköprü verisi üzerinde karşılaştırılması	93
Şekil 5.14 Algoritmaların Akköprü verisi üzerinde karşılaştırılması	94
Şekil 5.15 BMS-WebView-1 verisi üzerinde apriori sonuçları	95
Şekil 5.16 BMS-WebView-1 verisi üzerinde aprioritid sonuçları	96
Şekil 5.17 BMS-WebView-1 verisi üzerinde apriorihybrid sonuçları	97
Şekil 5.18 BMS-WebView-1 verisi üzerinde DHP sonuçları	98
Şekil 5.19 BMS-WebView-1 verisi üzerinde algoritmaların karşılaştırılması	99
Şekil 5.20 BMS-WebView-1 verisi üzerinde algoritmaların karşılaştırılması	99

ÇİZELGE LİSTESİ

Çizelge 4.1 Apriori Algoritması [4]	32
Çizelge 4.2 Birleştirme adımı [4]	32
Çizelge 4.3 Örnek 4.1'e ait veri tabanı örneği	35
Çizelge 4.4 Algoritmanın birinci adımında oluşturulan şartlı örüntü temeli	37
Çizelge 4.5 Şartlı Örüntü temelinden şartlı FP-ağacı oluşturma.....	38
Çizelge 4.6 CHARM Algoritması	42
Çizelge 4.7 Eclat Algoritması	43
Çizelge 4.8 Örnek 4.2'ye ait örnek veri tabanı	45
Çizelge 4.9. Örnek 4.3 için işlem Veri tabanı TDB (J.Pei,J.Han,R.Mao,2000)	46
Çizelge 4.10 Partition – Yerel Kalem kümesi Madenleme	53
Çizelge 4.11 Partition	53
Çizelge 4.12 Örnek 4.5'de verilen algoritmalarının karşılaştırılmasında kullanılan verilerin özellikleri	56
Çizelge 4.13 Örnek 4.6'ya ait karşılaştırma işleminde kullanılan verilerin özellikleri.....	58
Çizelge 5.1 AIS algoritması (Adamo,2000)	63
Çizelge 5.2 Apriori algoritması (Adamo, 2000)	65
Çizelge 5.3 Apriori-aday kalem kümesi oluşturma prosedürü (Adamo, 2000)	65
Çizelge 5.4 Apriori-aday kalem kümelerini ekleme prosedürü (Adamo, 2000).....	67
Çizelge 5.5 Apriori-aday kalemlerini sayma prosedürü (Adamo, 2000)	68
Çizelge 5.6 Örnek 5.3'de kullanılan veri tabanı örneği	69
Çizelge 5.7 Örnek 5.3'de elde edilen sonuçlar.....	70
Çizelge 5.8 AprioriTid algoritması (Adamo, 2000).....	72
Çizelge 5.9 Aday Oluşturma Prosedürü (Adamo, 2000)	73
Çizelge 5.10 DHP algoritması (Adamo, 2000)	78
Çizelge 5.11 Aday oluşturma prosedürü(Adamo, 2000).....	79
Çizelge 5.12 Adayları sayma ve kısaltma prosedürü (Adamo, 2000).....	80
Çizelge 5.13 PossFreq dizisini sayma ve kısaltma (Adamo, 2000)	81
Çizelge 5.14 Veri kümelerinin özellikleri	85
Çizelge 5.15 Akköprü verisi üzerinde Apriori sonuçları	87
Çizelge 5.16 Akköprü verisi üzerinde AprioriTid sonuçları.....	88
Çizelge 5.17 Akköprü verisi üzerinde AprioriHybrid sonuçları	90
Çizelge 5.18 Akköprü verisi üzerinde DHP sonuçları	91
Çizelge 5.19 BMS-WebView-1 verisi üzerinde Apriori sonuçları	95
Çizelge 5.20 BMS-WebView-1 verisi üzerinde AprioriTid sonuçları.....	96
Çizelge 5.21 BMS-WebView-1 verisi üzerinde AprioriHybrid sonuçları	97
Çizelge 5.22 BMS-WebView-1 verisi üzerinde DHP sonuçları	98
Çizelge 5.23 BMS-WebView-2 verisi üzerinde Apriori sonuçları	100
Çizelge 5.24 BMS-WebView-2 verisi üzerinde AprioriTid sonuçları.....	101
Çizelge 5.25 BMS-WebView-2 Verisi Üzerinde AprioriHybrid Sonuçları	102
Çizelge 5.26 İlişkisel kural üretme algoritması.....	105

ÖNSÖZ

Veri madenciliği; büyük miktarda veri içinden gelecekle ilgili tahmin yapmamızı sağlayacak bağıntı ve kuralların bilgisayar programları kullanılarak aranmasıdır. Örneğin süper marketlerde veri analizi yapılarak her ürün için bir sonraki ayın satış tahminleri çıkarılabilir; müşteriler, satın aldıkları ürünlere bağlı olarak gruplandırılabilir; yeni bir ürün için potansiyel müşteriler belirlenebilir; müşterilerin zaman içindeki hareketleri incelenerek onların davranışları ile ilgili tahminler yapılabilir. Binlerce ürünün ve müşterinin olabileceği düşünülürse bu analizin gözle ve elle yapılamayacağı, otomatik olarak yapılmasının gerektiği ortaya çıkar. Veri madenciliği burada devreye girer.

Geleceğin, en azından yakın geleceğin, geçmişten çok fazla farklı olmayacağını varsayarsak geçmiş veriden çıkarılmış olan ilişkisel kurallar gelecekte de geçerli olacak ve ilerisi için doğru tahmin yapmamızı sağlayacaktır. Bu amaçla veri madenciliği dalı altında ilişkisel kural madenciliği konusu son zamanlarda önem kazanmış ve birçok araştırmanın konusu olmuştur. İlişkisel kuralları bulabilmek için birçok algoritma geliştirilmiş ve bu algoritmaların performanslarını arttırmak veya yeni algoritmalar geliştirmek için araştırmalar yapılmıştır.

Bu tez çalışmasında ilişkisel kural madenciliği algoritmaları araştırılmıştır. Bu konudaki en temel algoritmalarından biri olan Apriori ve ondan türetilen diğer algoritmalar incelenmiş ve C dilinde programlanarak çeşitli veriler üzerinde karşılaştırmaları yapılmıştır.

Çalışmam boyunca beni destekleyen; tez yöneticim Prof. Dr. Oya Kalıpsız'a, tüm destek ve yardımlarından dolayı eşim Hüseyin Erol'a, çalışma ve oda arkadaşlarıma ve özellikle aileme destekleri ve yardımları için teşekkür ederim.

ÖZET

Bilişim uygulamalarının yaygınlaşması ile, bilgisayarlarda büyük miktarlarda veri depolanmasına başlanmıştır. Günümüz veri tabanı sistemleri, kullanıcıya depolanan bütün bilgilere kolayca ulaşabileceği araçları ve fonksiyonları sunmamaktadır. Büyük veri tabanlarında saklı olan bu bilgilere ulaşmak ve bu bilgileri kullanmak üzere, otomatik bilgi keşfetmeye yarayan teknikler geliştirilmektedir. Bu tekniklerden biri olan ilişkisel kurallar bulma, depolanan verilerden, ilginç ve sıklıkla rastlanan şemaları tanıma işlevinin, yani veri araştırmasının çok önemli bir dalıdır. İlişkisel kurallar, nesnelerin bir arada olma durumlarını belirlemeyi amaçlar ve bir çok alanda geniş kullanılabilirliğe sahiptir. İlişkisel kurallar bulma, yoğun nesne kümelerinin (verilerde sıkça bir arada görülen nesnelerin) hesaplanması esasına dayanır ve büyük veri tabanlarında hesaplanması oldukça pahalı bir işlemdir. Bu nedenle, daha önce belirlenmiş ilişkisel kuralların korunması oldukça önemli bir konudur.

Bu çalışmada, ilişkisel kural üreten temel algoritmalarından biri olan Apriori ve onun türleri olan AprioriTid, AprioriHybrid ve DHP algoritmaları incelenmiş ve C dilinde bu algoritmaların programları yazılmıştır. Algoritmalar performanslarına, yöntem farklılıklarına, kural üretmeye başladıkları minimum eşik değerlerine göre farklı veri kümeleri üzerinde karşılaştırılmıştır. Son olarak da tüm algoritmaların ortak olarak kullandıkları bir ilişkisel kural üretme prosedürü yazılmış ve üretilen kurallar bir dosya ortamında saklanmıştır.

Anahtar kelimeler: Veri Madenciliği, İlişkisel kural madenciliği, Apriori, AprioriTid, AprioriHybrid, DHP, algoritma.

ABSTRACT

By improvement of data processing methods, a large amount of data is stored in computers. Today's database systems offer a lot of devices and tools to people to get data simply. Automatic information discovery methods are improved to get and use information that is stored in large databases. One of these methods is association rule discovery that is important to find interesting and frequent patterns in data. Association rules purpose counting of objects that get together frequently in data. So it has a large usability at a large of area. Association rule discovery is based on counting of dense object sets and that counting is very expensive in large databases. So it is important to protect rules that are found at first.

In this study, Apriori that is one of the main algorithms of associaiton rule mining and its variations that are AprioriTid and AprioriHybrid, and DHP that is based on Apriori are researched and these algorithms are programmed in C language. Algorithms are compared with their run time performance, methods and minimum support tresholds on different data sets. At last, an association rule procedure is used as common by all algorithms and rules that are produced are stored in a file format.

Keywords: Data Mining, Associaiton Rule Mining, Apriori, AprioriTid, AprioriHybrid, DHP, algorithm.

1. GİRİŞ

Veri tabanlarından ilişkisel kuralların bulunması veri madenciliğinin en önemli konularından biridir. İlişkisel kural madenciliği, veri tabanlarında bir arada sık görülen ilişkileri ortaya çıkaran ve özetleyen kuralları (ilişkisel kurallar) bulmaktır. Bir alışveriş sırasında veya birbirini izleyen alışverişlerde müşterinin hangi ürün veya hizmetleri satın almaya eğilimli olduğunun belirlenmesi, müşteriye daha fazla ürünün satılmasını sağlama yollarından biridir. Satın alma eğilimlerinin tanımlanmasını sağlayan ilişkisel kurallar ve ardışık zamanlı örüntüler, pazarlama amaçlı olarak pazar sepeti analizi (market basket analysis) adı altında veri madenciliğinde yaygın olarak kullanılmaktadır. Bununla birlikte bu teknikler, tıp, finans ve farklı olayların birbirleri ile ilişkili olduğunun belirlenmesi sonucunda değerli bilgi kazanımının söz konusu olduğu ortamlarda da önem taşımaktadır. Bu amaçla ilişkisel kural madenciliğinde kullanılan birçok algoritma vardır. Bu çalışmada amaç, bu algoritmaların araştırılıp, performans incelemelerinin yapılmasıdır.

İlişkisel kurallar aşağıda sunulan örneklerde görüldüğü gibi eş zamanlı olarak gerçekleşen ilişkilerin tanımlanmasında kullanılır.

- Müşteriler bira satın aldığı anda, % 75 ihtimalle patates cipsi de alırlar,
- Düşük yağlı peynir ve yağsız yoğurt alan müşteriler, %85 ihtimalle diet süt de satın alırlar.

Ardışık zamanlı örüntüler ise aşağıda sunulan örneklerde görüldüğü gibi birbirleri ile ilişkisi olan ancak birbirini izleyen dönemlerde gerçekleşen ilişkilerin tanımlanmasında kullanılır.

- X ameliyatı yapıldığında, on beş gün içinde % 45 ihtimalle Y enfeksiyonu oluşacaktır,
- İMKB endeksi düşerken A hisse senedinin değeri % 15'den daha fazla artacak olursa, üç iş günü içerisinde B hisse senedinin değeri % 60 ihtimalle artacaktır,
- Çekiç satın alan bir müşteri, ilk üç ay içerisinde % 15, bu dönemi izleyen üç ay içerisinde % 10 ihtimalle çivi satın alacaktır.

İlişkisel Kuralların En Önemli Uygulama Alanları:

- Sepet analizi,
- Çapraz-pazarlama (cross-marketing),
- Katalog tasarımı,
- Yerleşim düzeni tasarımı,
- Promosyon analizleri.

2. VERİ MADENCİLİĞİ

2.1 Veri Madenciliği Nedir?

Bilgisayar sistemleri her geçen gün hem ucuzlamakta, hem de güçleri artmaktadır. İşlemciler gittikçe hızlanmakta ve disklerin kapasiteleri artmaktadır. Artık bilgisayarlar daha büyük miktardaki veriyi saklayabilmekte ve daha kısa sürede işleyebilmektedir. Bunun yanında bilgisayar ağlarındaki ilerleme ile bu veriye başka bilgisayarlardan da hızla ulaşabilmek mümkündür. Bilgisayarların ucuzlaması ile sayısal teknoloji daha yaygın olarak kullanılmaktadır. Veri doğrudan sayısal olarak toplanmakta ve saklanmaktadır. Bunun sonucu olarak da detaylı ve doğru bilgiye ulaşılabilir.

Veri kendi başına değersizdir. Bilgi bir amaca yönelik işlenmiş veridir. Veriyi bilgiye çevirmeye *veri analizi* denir. Bilgi de bir soruya yanıt vermek için veriden çıkardığımız sonuç olarak tanımlanabilir. Veri; sayılar, harfler ve onların anlamıdır. Yani sadece sayılar veya harfler değildir. Veri hakkındaki bu veriye *metaveri* denilmektedir.

Süpermarketlerde veri analizi yaparak her mal için bir sonraki ayın satış tahminleri çıkarılabilir; müşteriler satın aldıkları mallara bağlı olarak gruplandırılabilir; yeni bir ürün için potansiyel müşteriler belirlenebilir; müşterilerin zaman içindeki hareketleri incelenerek onların davranışları ile ilgili tahminler yapılabilir. Binlerce malın ve müşterinin olabileceği düşünülürse bu analizin gözle ve elle yapılamayacağı, otomatik olarak yapılmasının gerektiği ortaya çıkar.

Veri madenciliği büyük miktarda veri içinden gelecekle ilgili tahmin yapmamızı sağlayacak bağıntı ve kuralların bilgisayar programları kullanılarak aranmasıdır. Geleceğin, en azından yakın geleceğin, geçmişten çok fazla farklı olmayacağı varsayılırsa geçmiş veriden çıkarılmış olan kurallar gelecekte de geçerli olacak ve ilerisi için doğru tahmin yapmamızı sağlayacaktır [1].

Veri Madenciliğinin İşletmelerde Kullanım Amacı

İstatistiğin amacı nasıl ana kütle hakkında anlamlı bilgiler elde etmek ve yorum yapmaksa veri madenciliğinin amacı da anlamlı bilgiler elde etmek ve bunu eyleme dönüştürecek kararlar için kullanmaktır. Burada ana kütle -örneğin- mevcut veya potansiyel müşteriler olabilir. Müşterilerin profillerini, satın alma eğilimlerini, bir ürünü veya hizmeti kabul etme veya etmeme ihtimallerini tahmin etme, veri madenciliğinde hedeflenen amaçlar arasındadır. Bu tahminler, ana hedef olan stratejiyi belirlemek için çeşitli kararlar vermek amacıyla

kullanılırlar [2].

Ürün ve hizmet sektöründe, müşterilerle ilgili veri madenciliği uygulama amaçlarına ilişkin çok çeşitli örnekler verilebilir. En karlı pazar bölümlerini, en karlı müşterileri, yeni bir promosyon kampanyasında müşteriye sunulan ürün veya hizmetin kabul edilme oranlarını saptamak pazarlamada veri madenciliğinin uygulamasında önemli amaçlardır. Veri madenciliği uygulamalarından elde edilecek faydalara ilişkin bazı örnekler aşağıda sıralanmıştır:

- Bir işletme, önceden kendi müşterisi olup sonradan rakip firmaya giden müşterilerle ilgili analizler yaparak rakiplerini tercih eden müşterilerinin özelliklerini elde edebilir ve bundan yola çıkarak gelecek dönemlerde kaybetme olasılığı olan müşterilerinin kimler olabileceği yolunda tahminlerde bulunarak onları kaybetmemek, kaybettiklerini geri kazanmak için strateji geliştirebilir.
- Ürün veya hizmette hangi özelliklerin ne derecede müşteri memnuniyetini etkilediği, hangi özelliklerinden dolayı müşterinin bunları tercih ettiği ortaya çıkarılabilir.
- Müşterilerin kredi riskleri hesaplanarak hangi müşterilerin kredi riskinin yüksek olduğu, hangi müşterilerin geri ödemesini zamanında yapamayabileceği ile ilgili tahminler yapılabilir. Kredi kartı ödemelerini aksatan, gecikmeli olarak yapan veya hiç yapmayanların özelliklerinden yola çıkılarak bundan sonra aynı duruma düşebilecek muhtemel kişiler saptanabilir.
- En karlı mevcut müşteriler saptanarak, potansiyel müşteriler arasından en karlı olabilecekler belirlenebilir. Karlı müşteriler tesbit edilerek onlara özel kampanyalar uygulanabilir. En masraflı müşteriler daha masrafsız müşteri haline dönüştürülebilir. Örneğin en çok bankacılık işlemi yapanlar ortaya çıkarılıp bunlar şube bankacılığı yerine daha masrafsız internet bankacılığına yönlendirilebilirler.
- Bir ürün veya hizmetle ilgili bir kampanya programı oluşturmak için hedef kitlenin seçiminden başlayarak, bunun hedef kitleye hangi kanallardan sunulacağı kararına kadar olan süreçte veri madenciliği kullanılabilir.

2.2 Veri Madenciliği Teknikleri

Bağıntı:

Sepet analizinde müşterilerin beraber satın aldığı malların analizi yapılır. Buradaki amaç mallar arasındaki pozitif veya negatif korelasyonları bulmaktır. Çocuk bezi alan müşterilerin mama da satın alacağını veya bira satın alanların cips de alacağını tahmin edebiliriz ama ancak otomatik bir analiz bütün olasılıkları gözönüne alır ve kolay düşünilemeyecek bağıntıları da bulur. Örneğin;

- “Çocuk bezi alan müşterilerin %30’u bira da satın alır.”

Sınıflandırma:

Amaç bir malın özellikleri ile müşteri özelliklerini eşlemektir. Böylece bir müşteri için ideal ürün veya bir ürün için ideal müşteri profili çıkarılabilir. Örneğin bir otomobil satıcısı şirketin geçmiş müşteri hareketlerinin analizi ile aşağıdaki gibi iki kural bulursa genç kadınların okuduğu bir dergiye reklam verirken küçük modelinin reklamını verir.

- “Genç kadınlar küçük araba satın alır. Yaşlı ve zengin erkekler büyük ve lüks araba satın alır.”

Regresyon:

Başvuru skorlamada (application scoring), bir finans kurumuna kredi için başvuran kişi ile ilgili finansal güvenilirliği hesaplar; örneğin 0 ile 1000 arasında bir skor verilerek bu hesaplama işlemi yapılır. Bu skor kişinin özellikleri ve geçmiş kredi hareketlerine dayanılarak hesaplanır. Örneğin;

- “Ev sahibi olan, evli, aynı iş yerinde beş yıldan fazladır çalışan, geçmiş kredilerinde geç ödemesi bir ayı geçmemiş bir erkeğin kredi skoru 825’dir.”

Zaman İçinde Sıralı Örüntüler:

Davranış skoru (behavioral score), başvuru skorundan farklı olarak kredi almış ve taksitleri ödeyen bir kişinin sonraki taksitlerini ödeme/geciktirme davranışını notlamayı amaçlar. Örneğin;

- “İlk üç taksidinden iki veya daha fazlasını geç ödemiş olan müşteriler %60 olasılıkla kanuni takibe gidiyor.”

Benzer Zaman Sıraları:

Amaç, zaman içindeki iki hareket serisi arasında bağıntı kurmaktır. Bunlar iki malın zaman içindeki satış miktarları olabilir. Örneğin dondurma satışları ile kola satışları arasında pozitif, dondurma satışları ile sahlep satışları arasında negatif bir bağıntı beklenebilir. Örneğin;

- “*X şirketinin hisse fiyatları ile Y şirketinin hisse fiyatları benzer hareket ediyor.*”

İstisnalar (Fark Saptanması):

Amaç; önceki uygulamaların aksine kural bulmak değil, kurala uymayan istisnai hareketleri bulmaktır. Bu da olası sahtekarlıkların saptanmasını (fraud detection) sağlar. Örneğin; Visa kredi kartı için yapılan CRIS sisteminde bir yapay sinir ağı, kredi kartı hareketlerini takip ederek müşterinin normal davranışına uymayan hareketler için müşterinin bankası ile temasa geçerek müşteri onayı istenmesini sağlar. Fark saptanması yöntemi örneğin aşağıdaki soruya cevap bulmamıza olanak sağlar.

- “*Normalden farklı davranış gösteren müşterilerim var mı?*”

Döküman Madenciliği (text mining):

Amaç; dökümanlar arasında ayrıca elle bir tasnif gerekmeden benzerlik hesaplayabilmektir. Bu işlem genelde otomatik olarak çıkarılan anahtar sözcüklerin tekrar sayısı sayesinde yapılır. Örneğin;

- “*Arşivimde (veya internet üzerinde) bu dökümana benzer hangi dökümanlar var?*”

2.3 Kullanılan Teknikler

Veri madenciliğinde en çok kullanılan yöntemler, istatistiksel yöntemler, bellek tabanlı yöntemler, yapay sinir ağları ve karar ağaçlarıdır. Bunlar;

İstatistiksel Yöntemler:

Veri madenciliği çalışması esas olarak bir istatistik uygulamasıdır. İstatistik literatüründe son elli yılda bu amaç için değişik teknikler önerilmiştir. Bu teknikler istatistik literatüründe çok boyutlu analiz (multivariate analysis) başlığı altında toplanır ve genelde verinin parametrik bir modelden (çoğunlukla çok boyutlu bir Gauss dağılımından) geldiğini varsayar. Bu varsayım altında sınıflandırma (classification; discriminant analysis), regresyon, kümeleme (clustering), boyut azaltma (dimensionality reduction), hipotez testi, varyans analizi, ilişki (association;

dependency) kurma gibi teknikler istatistikte uzun yıllardır kullanılmaktadır (Rencher, 1995).

Bellek Tabanlı Yöntemler:

Bellek tabanlı veya örnek tabanlı bu yöntemler (memory-based, instance-based methods; case-based reasoning) istatistikte 1950'li yıllarda önerilmiş olmasına rağmen o yıllarda gerektirdiği hesaplama ve bellek kapasitesi yüzünden kullanılamamış; fakat günümüzde bilgisayarların ucuzlaması ve kapasitelerinin artmasıyla, özellikle de çok işlemcili sistemlerin yaygınlaşmasıyla kullanılabilir olmuştur. Bu yönteme en iyi örnek ;

- “en yakın k komşu algoritması (k -nearest neighbor)”

dır (Mitchell, 1997).

Yapay Sinir Ağları (YSA):

1980'lerden sonra yaygınlaşan yapay sinir ağlarında (artificial neural networks) amaç fonksiyon, birbirine bağlı basit işlemci ünitelerinden oluşan bir ağ üzerine dağıtılmıştır (Bishop, 1996). Yapay sinir ağlarında kullanılan öğrenme algoritmaları veriden, üniteler arasındaki bağlantı ağırlıklarını hesaplar. YSA, istatistiksel yöntemler gibi veri hakkında parametrik bir model varsaymaz. Yani uygulama alanı daha genişler ve bellek tabanlı yöntemler kadar yüksek işlem ve bellek gerektirmez.

Karar ağaçları:

İstatistiksel yöntemlerde veya yapay sinir ağlarında veriden bir fonksiyon öğrenildikten sonra bu fonksiyonun insanlar tarafından anlaşılabilir bir kural olarak yorumlanması zordur. Karar ağaçlarında ise ağaç kökten yaprağa doğru inilerek kurallar (IF-THEN rules) yazılabilir (Mitchell, 1997). Bu şekilde kural çıkarma (rule extraction), veri madenciliği çalışmasının sonucunun geçerlenmesini sağlar. Bu kurallar ile uygulama konusunda uzman bir kişiye gösterilerek sonucun anlamlı olup olmadığı denetlenebilir. Sonradan başka bir teknik kullanılacak bile olsa öncelikle karar ağacı ile kısa bir çalışma yapmak, önemli değişkenler ve yaklaşık kurallar konusunda bize bilgi verir ve tavsiye edilir [1].

2.4 Veri Ambarı

Günümüzde pekçok kurumda, artan miktarda, bütünleşik olmayan, tekrarlanan, dağınık ve yönetimi, erişimi, karar sürecinde kullanımı güç olan veri problemi bulunmaktadır. Veri ambarı, farklı kaynaklardan elde edilen verinin kullanıcıların bir iş ya da karar sürecinde

kullanabileceği şekilde sunulduğu bütünleşik bir veri saklama ortamıdır. (Kalıpsız,1998)

Günümüzde ticari işletmelerin bilgi sistemlerinin iki ayrı başlık altında toplandığı söylenebilir:

- **Canlı sistemler:** Bu sistemlerde güncel veriler bulunur. Günlük yapılan işleri gerçeklemek ve sonuçları saklamak için tasarlanan bu sistemlere, marketlerde ya da mağazalarda stok takibi, üyelerin borçları, satış işlemleri ve ödeme kayıtları gibi bilgilerin işlendiği ve tutulduğu bilgi sistemleri örnek verilebilir. Bu tür veride canlı sistemlerde erişebilirlik esastır; yani veriye en kısa sürede ulaşmak, işlemleri en kısa sürede sona erdirmek hedeflenir. Örneğin süpermarkette bir satış işleminin mümkün olduğu kadar çabuk bitirilmesi istenir. Bu nedenle canlı sistemler çevrim içi çalışma prensibi ile tasarlanır.
- **Karar Destek Sistemleri:** İşletmelerde yer alan ikinci tür bilgi sistemleri ise karar destek sistemleridir. Bu sistemlerde yer alan bilgiler çeşitli incelemelerden ve araştırmalardan geçirilerek, yöneticilerin ileride işletmenin karını ya da verimliliğini arttırması, gelecekte izlenecek şirket politikalarının belirlenmesi ve benzeri yönetsel kararların alınmasını kolaylaştırır ve bu kararların daha doğru verilmesine yardımcı olurlar. Bu sistemlerde verilerin erişimi birinci kriter değildir ve herhangi bir veriye herhangi bir anda çabuk erişmek gerekmez. Karar destek sistemlerinde öncelik performanstadır. Bu sistemlerde veriler, canlı sistemlere oranla çok daha büyük boyutlardadır. Dolayısıyla bu verilerin incelenmesi ve bu incelemelerden sonuçlar çıkartılması, sistem kaynaklarını aşırı kullanmakta ve uzun süre almaktadır. Bu nedenle karar destek sistemlerinde, yapılacak incelemelerin ve araştırmaların performansını arttırmak için bir takım önlemler alınmış ve iyileştirmeler yapılmıştır.

Veri ambarı, bir karar destek sistemi olarak nitelendirilebilir. Veri ambarı aslında günlük işlemlerin gerçekleştiği canlı sistemlerin arka planında bulunmaktadır. Canlı sistemlerde oluşan veriler periyodik olarak veri ambarına aktarılırlar. Bu periyodun seçimi tamamen veri ambarını kullanan işletmenin ihtiyaçları doğrultusunda belirlenir ve bir gün, bir hafta veya bir ay gibi çok değişken olabilir. Dolayısıyla bir veri ambarı çevrimdışı olarak çalışmaktadır. Yani bir veri ambarı içerisindeki kayıtlar güncel olmayabilir, çoğunlukla da değildir.

Bu şekilde veri ambarına aktarılan veriler üzerinde daha sonra gerçekleştirilen incelemeler ve araştırmalar neticesinde yöneticiler ve iş uzmanlarının daha rahat karar alabilecekleri ve yapılacak işlere ışık tutan bir takım sonuçlar üretilir. Bu sonuçlar istatistiksel, olasılıksal

raporlar şeklinde olabileceği gibi, bu büyük veri yığını içerisinde yer alan en tecrübeli yöneticilerin bile göremeyeceği veri ilişkilerinin belirlenmesi şeklinde de olabilir. İlişkisel veri madenciliği konusu da bu ihtiyaçlardan dolayı önem kazanmıştır.

2.4.1 Bir Veri Ambarı Yapısı

Bir veri ambarının yapısı organizasyon içindeki bütün son kullanıcılara verileri ve işlem sonuçlarını sunan, en gelişmiş iletişimi sağlayan bir dizi birbiriyle bütünleşik alt bileşenlerden oluşur. Bunlar;

- İşlemsel (operational) Veri Tabanı,
- Enformasyon Ulaşım katmanı,
- Veri Ulaşım Katmanı,
- Veri Dizini (Data Directory, Metadata) Katmanı,
- İşlem (process) Yönetim Katmanı,
- Uygulama Haberleşmesi Katmanı,
- Veri Ambarı Katmanı,
- Veri Sunum Katmanı.

İşlemsel Veri Tabanı:

İşlemsel sistemler, kritik işlemsel ihtiyaçlar için verileri işlerler. Bunu gerçekleştirebilmek amacıyla uzun süreçlerdeki tecrübelerden yararlanılarak en verimli iş modelleri tanımlanmış ve işlemsel veri tabanları oluşturulmuştur. Bu nedenle, işlemsel sistemler üzerindeki limitli odaklanmalar, yönetim ve de enformasyonel amaçlı verilere ulaşmada zorlukların çıkmasına neden olmuşlardır. İşlemsel verilere ulaşmadaki bu zorlukların çoğu 10-15 yıllık işlemsel sistemler tarafından da imkansız gibi görülmeye başlanmıştır. Ancak bahsettiğimiz veri tabanları bu zorlukları bertaraf ederek kolay bir şekilde verilere ulaşılmasını sağlamaktadırlar.

Enformasyon Ulaşım Katmanı:

Enformasyon ulaşım katmanının bir veri ambarındaki yeri, son kullanıcının direkt olarak veriye ulaşması ile ilgili kısmıdır. Bu amaca yönelik olarak enformasyon ulaşım katmanı kullanıcıların bu verilere ulaşması için araçlar sunar; bunlar hergün kullandığımız Excel, Lotus 1-2-3, Focus, Access, vs.'dir. Bu katman donanım ve yazılımları, grafikleri, sunuları ve sonuçları kağıt üzerine aktaran yazıcıları da kapsar. Özellikle geçtiğimiz son on yıl içinde

bilgisayar ağlarının oldukça kolay kurulur ve kullanılabilir hale gelmesi ortak bir platformda çalışmayı da kolay hale getirmiştir.

Günümüzde verileri analiz etmek ve sunmak için masalarımızda konunun çok daha fazla felsefi boyutları düşünülerek tasarlanmış araçlar mevcuttur. Bununla birlikte, ham verileri operasyonel işlemlerde kullanılabilir hale getirirken çok önemli sorunlar da ortaya çıkmıştır. Bu sorunların çözüm örneklerinden birisi olarak da verilerin sorgulanmasında ortak bir sorgulama dilinin kullanılmasını gösterebilmek mümkündür.

Veri Ulaşım Katmanı:

Bir veri ambarı yapısının veri ulaşım katmanı enformasyon ulaşım katmanı ile işlemsel katmanın birbiriyle haberleşmesini sağlayan bir katmandır. Günümüzdeki ağ (network) dünyasında kabul edilen ortak sorgulama dili olan SQL bu sebeple çıkmıştır. İlk olarak IBM tarafından geliştirilen bu veri tabanı sorgulama dili geçtiğimiz 20 yılda bir veri sorgulama dilinden öte bir veri değişimi dili olarak endüstriyel bir standart halini almıştır. Veri Ambarcılığının en önemli stratejilerinden birisi "veriye evrensel ulaşım sağlama" dır. Veriye evrensel ulaşım sağlamak demek, son kullanıcının teorik olarak da olsa veriye herhangi bir ekstra araç, yazılım kullanmadan ihtiyaç duyduğu her an her yerden ulaşabilmesidir.

Veri Dizini (Data Directory ,Metadata) Katmanı:

Evrensel veri ulaşımını sağlayabilmek için, verilerin yapısı da elden geçirilmelidir. İdeal bir veri ambarına sahip olabilmek için kullanılabilir haldeki çeşitli meta-data tiplerine ihtiyaç vardır. Mükemmel olan, kullanıcıların veri ambarına verilerin nerede tutulduğunu bile bilmeksizin ulaşabilmesidir.

İşlem Yönetim Katmanı:

İşlem yönetim katmanı, zamanı belli olan, yapılması ve devamlılığı sürekli olan bir dizi işlemleri ihtiva eder. İşlem yönetim katmanı yüksek seviyedeki işlerin kontrol altında tutulabilmesini sağlayan, işleri planlayan ve yüksek miktardaki verileri her zaman doğru ve yeni tutan bir ara birim olarak da düşünülebilir.

Uygulama Haberleşme Katmanı:

Uygulama haberleşme katmanı enformasyonun, ağ haberleşmesi teknikleri ve teknolojileriyle bütün organizasyon içinde nakledilmesinden sorumlu olan katmandır. Uygulama

haberleşmesi orta kademe katman diye de adlandırılabilir; fakat sadece ağ haberleşmesinden öte, uygulamaların birbirinden yalıtılmasından, verilerin işlemsel ya da enformasyonel olarak ayırımına ve verilerin doğru formata çevriminden de sorumludur. Uygulama haberleşmesi katmanı, bilgisayarlar arasındaki uygulama işlemlerinin sonuçlarının tutulması ve doğru zamanda doğru kimseye en doğru verileri sağlamak amacıyla da kullanılır.

Veri Ambarı Katmanı (Fiziksel Katman):

Veri Ambarları temel olarak enformasyonel veriyi kullanırlar. Çoğu zaman insanlar veri ambarlarının fiziksel olarak verileri depolamadığını, bunun sanal olarak verilerin farklı bir formatta görünümünden ibaret olduğunu düşünürler. Uygulamaların çoğunda veri, veri ambarlarında depolanmayıp, fiziksel bir veri ambarında tutulurlar. Günümüzdeki veri ambarları web tabanlı kişiselleştirilmiş platformları kullanmaya başlamıştır ancak temelde aynı yapıyı kullanırlar.

Veri Sunum Katmanı:

Bir veri ambarı yapısının sonuncu bileşeni de veri sunum katmanıdır. Veri sunum katmanı kopya yönetim katmanı ya da replikasyon katmanı olarak da adlandırılır. Fakat bu katman, işlemsel veri tabanlarına ulaşılarak yapılan tespit, düzeltme, sonuçlandırma, uyumlu hale getirme, yükleme işlemleri için gereken bütün aşamaları da kapsamaktadır.

Veri sunumu karmaşık programlama tekniklerine de gereksinim duyar. Ancak günümüzde hızla gelişen veri ambarcılığı araçları ile bu aşama her geçen gün daha da kolaylaşmaktadır. Veri sunum katmanı, mevcut işlemsel veri içindeki kalıpları ve veri modellerini de tespit etmek amacıyla veri kalitesini analiz eden muhtelif filtreleri ve/veya programları da içermektedir [8].

2.5 OLAP

Veri hacminin büyümesi ve veri ambarı kullanıcılarının karar destek sistemi işlemlerini daha iyi anlamaları için, veri ambarının kullanımını kolaylaştıracak daha karmaşık tekniklere gereksinim duyulmaktadır. Çevrim içi çözümsel işlem (Online Analytical Processing, OLAP), ham veriden elde edilen bilgiye hızlı, uyumlu, etkileşimli erişim sağlayarak analist ve yöneticilerin kurumun gerçek boyutlarını görmelerine imkan sağlayan yazılım teknolojisidir. Ayrıca; OLAP, çok boyutlu analizi kolaylaştırmak üzere gerçekleştirilen işlevler dizisidir. Çok boyutlu analiz, boyutlar veya çeşitli sınıflar olarak gruplandırılan veriyi kullanma yeteneğidir (John Wiley & Sons,1997).

2.5.1 OLAP Sisteminin İşlevsel Gereksinimleri

OLAP araçları, birçok boyuta göre özetlemeler, kıyaslamalar ve hesaplamalar için SQL'den daha üstün olan çok boyutlu bir veri analizi gerçekleştirirler. OLAP sisteminin işlevsel gereksinimleri şunlardır:

- Hızlı erişim ve hesaplamalar,
- Güçlü analitik olanaklar (toplam, ortalama, kıyaslama, oranlama, trend hesapları),
- Esneklik (DSS yazılımlarında bulunmayan kullanıcının programda değişiklik yapabilmesi),
- Çoklu kullanıcı desteği (Client-Server uygulamalarında çalışabilme özelliği) (John Wiley & Sons,1997).

Karar yönelimli uygulamalarda tablolama ve veri tabanı uygulamaları yer almaktadır. Tablolama kullanımı uç kullanıcılar tarafından gerçekleştirildiği halde, veri tabanı uygulamaları bilgi teknolojisi uzmanları tarafından kullanılmaktadır.

Ürün toplamaları ve alt toplamaları, bu yıl/geçen yıl kıyaslaması, haftalık ortalamalar, en çok satan ürün veya bu yılın pazar değişimini belirlemek amacıyla tablolama analizleri kullanılmaktadır. Tablolama programları iki boyutlu olarak düzenlenirse, basit sorgular hazırlanarak küçük veri grupları analiz edilebilir. Büyük veri gruplarının analitik sorgulaması ise, ilişkisel veri tabanı sorgulaması ve SQL sorgulaması ile gerçekleştirilir.

2.5.2 Temel Çok Boyutlu OLAP Özellikleri

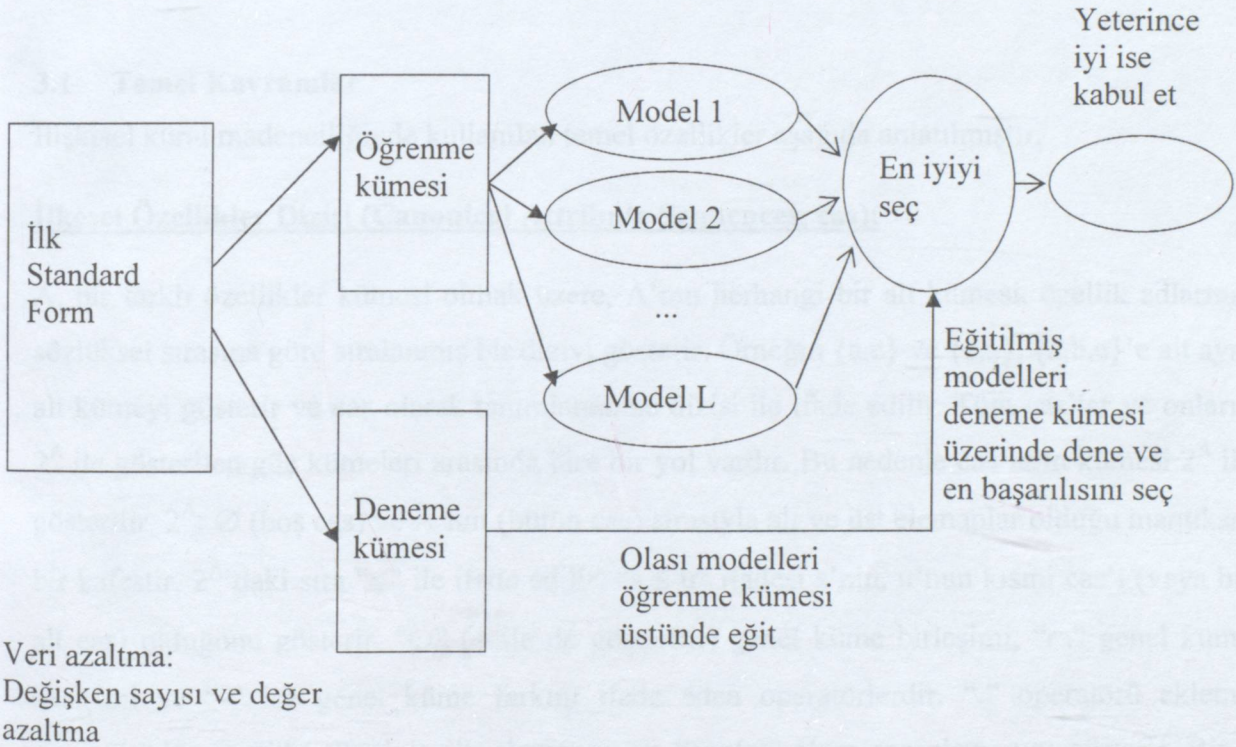
OLAP problemlerine tablolama yazılımları ve veri tabanı yönetim sistemleri ile çözüm getirmek zordur. OLAP uygulamalarındaki karmaşık, çok düzeyli bilgi biriktirme ve analizleri için çok boyutlu yazılımlar tasarlanmıştır.

OLAP boyutlu yazılım ürünlerinin temel anahtar özellikleri; çok boyutluluk, esnek ekran gösterimi, boyutların hiyerarşik yapısı ve formül – bağ (link) uygulamasıdır. Çok boyutluluk ile veri seti tanımlama kolaylığı; varlıklar, ölçütler ve elemanlar için kullanılmaktadır. Örneğin, satış izleme sisteminde, yer – satıcı – müşteri – zaman boyutları vardır. Çok boyutlu sistemlerde, bilgisayar ekranında, satırlar, sütunlar ve sayfalardan oluşan üç boyutlu düzenlemeler ile esnek ekran görünümü sağlanmaktadır. Çok boyutlu sistemlerde boyutların hiyerarşik olarak yapılandırılması sağlanabilir. Örneğin zaman boyutunda; gün, hafta, ay veya üç aylık seviyelerde birimleme gerçekleştirilebilmektedir. Bir boyutun elemanı ve diğer boyutun herbir elemanı ile formül veya bağ aracılığı ile ilişki kurulur (Kalıpsız, 1998).

2.6 Metodoloji

Bir veri madenciliği çalışmasında kullanılan metodoloji Şekil 2.5’da gösterilmiştir. Standart form içinde verilen veri, öğrenme ve deneme verisi olmak üzere ikiye ayrılır. Her uygulamada kullanılabilecek birden çok teknik vardır ve önceden hangisinin en başarılı olacağını kestirmek çok zordur. Bu nedenle öğrenme kümesi üzerinde L değişik teknik kullanılarak L tane model oluşturulur. Sonra bu L model deneme kümesi üzerinde denenerek en başarılı olanı, yani deneme kümesi üzerindeki tahmin başarısı en yüksek olanı seçilir.

Eğer bu en iyi model yeterince başarılıysa kullanılır, aksi takdirde başa dönerek çalışma tekrarlanır. Tekrar sırasında başarısız olan örnekler incelenerek bunlar üzerindeki başarının nasıl arttırılabileceği araştırılır. Örneğin standart forma yeni alanlar eklenerek programa verilen bilgi arttırılabilir; veya mevcut bilgi değişik bir şekilde kodlanabilir; veya amaç daha değişik bir şekilde tanımlanabilir.



Şekil 2.5. Veri madenciliği çalışmasında kullanılan metodoloji [1]

3. İLİŞKİSEL KURAL MADENCİLİĞİ

Bilişim uygulamalarının yaygınlaşması ile, bilgisayarlarda büyük miktarlarda veri depolanmasına başlanılmıştır. Ancak günümüz veri tabanı sistemleri, kullanıcıya depolanan bütün bilgilere kolayca ulaşabileceği araçları ve fonksiyonları sunmamaktadır. Büyük veri tabanlarında saklı olan bu bilgilere ulaşmak ve bu bilgileri kullanmak üzere, otomatik bilgi keşfetmeye yarayan teknikler geliştirilmektedir. Bu tekniklerden biri olan ilişkisel kurallar bulma tekniği; depolanan verilerden, ilginç ve sıklıkla rastlanan şemaları tanıma işlevinin, yani veri araştırmasının çok önemli bir dalıdır. İlişkisel kurallar, nesnelerin bir arada olma durumlarını belirlemeyi amaçlarlar ve bir çok alanda geniş kullanılabilirliğe sahiptirler. İlişkisel kurallar bulma, yoğun nesne kümelerinin hesaplanması esasına dayanır ve büyük veri tabanlarında hesaplanması oldukça pahalı bir işlemdir. Bu nedenle, daha önce belirlenmiş ilişkisel kuralların korunması oldukça önemli bir konudur.

3.1 Temel Kavramlar

İlişkisel kural madenciliğinde kullanılan temel özellikler aşağıda anlatılmıştır.

İlkesel Özellikler Dizisi (Canonical Attribute Sequences, cas):

A, bir farklı özellikler kümesi olmak üzere, A'nın herhangi bir alt kümesi, özellik adlarının sözlüksel sırasına göre sıralanmış bir diziyi gösterir. Örneğin {a,c} ve {c,a}, {a,b,c}'e ait aynı alt kümeyi gösterir ve cas olarak tanımlanan ac dizisi ile ifade edilir. Tüm cas'lar ve onların 2^A ile gösterilen güç kümeleri arasında bire bir yol vardır. Bu nedenle cas'ların kümesi 2^A ile gösterilir. 2^A ; $\emptyset$ (boş cas) ve A'nın (bütün cas) sırasıyla alt ve üst elemanlar olduğu mantıksal bir kafestir. 2^A 'daki sıra " $\leq$ " ile ifade edilir. " $s \leq u$ " ifadesi s'nin, u'nun kısmi cas'ı (veya bir alt cas) olduğunu gösterir. " $\cup$ " (+ ile de gösterilir) genel küme birleşimi, " $\cap$ " genel küme kesişimi ve " $-$ " ise genel küme farkını ifade eden operatörlerdir. " $\cdot$ " operatörü ekleme operatörüdür ve "ilk(s)", s'nin ilk elemanını ve "son(s)" s'nin son elemanını gösterir. Bir s cas'ı ve herhangi bir a özelliği için "s.a" ifadesi sadece s'deki tüm b'ler için (eğer b'ler sözlüksel sırasına göre a'dan küçükse) geçerlidir. "A[i]", A'daki i'ninci özelliği gösterir. Bir s cas'ının özellik sayısı " $|s|$ " ile ifade edilir.

Örneğin A kümesi {a,b,c} ve $ac \leq abc$, $ab \leq abc$ olsun.

ab, abc'nin bir önek sırasındır ama ac değildir.

$Ac \cup b = ac + b = abc$,

$$ac \cap c = c,$$

$$abc - ac = b,$$

$$ab.c = abc,$$

$ac.b$ geçerli değil

$$A[1] = a \quad (A[0] \text{ tanımsız}),$$

$$|ac| = 2$$

$$|A| = 3 \quad (\text{Adamo, 2000}).$$

Veri Tabanı:

Bir D veri tabanı için, mantıksal ilişkisel bir tablo Şekil 3.1'deki (a) alanı olarak verilirse, bu yatay olarak Şekil 3.1'deki (b) alanı gibi veya düşey olarak Şekil 3.1'deki (c) alanı gibi gösterilir. Yatay organizasyon bir (r,cas) çiftinden oluşur. Burada "r" sıra numarasını ve "cas" da bir ilkesel özellik dizisini ifade eder. Bu çift aynı zamanda veri tabanında bir işleme (transaction) karşılık gelir. Düşey organizasyon bir "a,liste" çifti kümesinden oluşur. Burada "a" değeri bir özelliği ve "liste" değeri de dizi numaralarının sıralı bir listesini ifade etmektedir.

D ilişkisel tablosu (a)	D yatay organizasyonu (b)	D düşey organizasyonu (c)
a b c d e	r cas	a b c d e
1 1 1 0 1 1	1-> abde	1 1 2 1 1
2 0 1 1 0 1	2-> bce	3 2 4 3 2
3 1 1 0 1 1	3-> abde	4 3 5 5 3
4 1 1 1 0 1	4-> abce	5 4 6 6 4
5 1 1 1 1 1	5-> abcde	5 5
6 0 1 1 1 0	6-> bcd	6

Şekil 3.1. $A=\{a,b,c,d,e\}$ için farklı veri tabanı gösterimleri (Adamo, 2000)

İlişkisel Kural Madenciliği:

İlişkisel kural madenciliği, veri içinde bir arada görülen özelliklerin kural olarak ortaya

çıkarılması işlemidir. Genel olarak $A_1 \wedge A_2 \dots \wedge A_K \Rightarrow B_1 \wedge B_2 \dots \wedge B_L$ olarak gösterilir ve her kural bir destek (support) ve güven (confidence) değeri ile ifade edilir.

Destek :

“ $A \Rightarrow B$ ” kuralının sağ ve sol tarafının aynı anda birlikte sağlanması ihtimalidir. Herhangi bir s cas’ı için;

$$\lambda(s) = \{r \mid (r, w), s \leq u\} \quad (3.1)$$

ifadesi D veri tabanında s’yi kısmi cas olarak içeren işlemlerin alt kümesidir.

$\lambda(s)$, s’nin destek değeridir. $\lambda(s)$ ’nin destek sayısı $|\lambda(s)|$ olarak gösterilir.

$$\text{destek}(s) = |\lambda(s)| / D \quad (3.2)$$

ifadesi s’nin destek oranıdır. Bir s cas’ı eğer $\text{destek}(s) \geq \sigma$ şartını sağlıyorsa (σ , kullanıcı tanımlı bir eşik değeridir) σ -sık’dır denir (Adamo, 2000).

Örneğin D, Şekil 3.1’deki gibi tanımlı ve destek oranı 0.5 alınan bir veri tabanı olsun. Bu destek oranının üstünde olan semboller ve destek değerleri aşağıda listelenmiştir:

- $\text{Destek}(b) = |\{1,2,3,4,5,6\}| / 6 = \% 100$
- $\text{Destek}(e) = |\{1,2,3,4,5\}| / 6 = \% 83$
- $\text{Destek}(a) = \text{destek}(ab) = \text{destek}(ae) = \text{destek}(abe) = |\{1,3,4,5\}| / 6 = \% 67$
- $\text{Destek}(c) = \text{destek}(bc) = |\{2,4,5,6\}| / 6 = \% 67$
- $\text{Destek}(d) = \text{destek}(bd) = |\{1,3,5,6\}| / 6 = \% 67$
- $\text{Destek}(ad) = \text{destek}(de) = \text{destek}(abd) = \text{destek}(ade) = \text{destek}(bde) = \text{destek}(abde) = |\{1,3,5\}| / 6 = \% 50$
- $\text{Destek}(ce) = \text{destek}(bce) = |\{2,4,5\}| / 6 = \% 50$

Güven :

“ $A \Rightarrow B$ ” kuralının sol tarafının sağlanması durumunda sağ tarafının da sağlanması olasılığıdır.

Kuvvetli İlişkisel Kural :

Minimum destek ve minimum güven şartlarını sağlayan kurallardır.

K-Kalem Kümesi (K-Item Set) :

Kişisel bilgisayar, yazıcı ve fare (mouse) 3-kalem kümesine örnektir. Burada destek sayısı kaç işlemde kişisel bilgisayar, yazıcı ve fare olduğu bilgisidir.

Sık Kalem Kümesi (Frequent Item Sets) :

Belli bir minimum destek sayısının üstünde rastlanan kalem kümeleridir. Örneğin elli işlemde kişisel bilgisayar, yazıcı ve fare alınmış ise ve minimum destek sayısı 20 ise bu işlem sık 3-kalem kümesidir.

Örnek bir veri tabanından aşağıdaki sonuçlar elde edilmiştir:

- $Yaş (X, "20...29") \wedge Gelir (X, "1 milyar...2 milyar") \Rightarrow Al (X, "CD çalar")$
- $Destek = \%5$
- $Güven = \%60$

Burada X, müşteriye temsil eden bir değişken olmak üzere;

- *Bütün müşterilerin %5'i 20 ile 29 yaş grubunda, geliri 1 ile 2 milyar arasında ve CD çalar almıştır.*
- *Ayrıca bu yaş ve gelir gurubundaki müşterilerin ise %60'ı CD çalar almıştır.*

İlişkisel Kural:

s ve u'nun $s \cap u = \emptyset$ şartını sağlayan iki keyfi cas olduğu varsayılın. σ ($0 \leq \sigma \leq 1$) ve γ ($0 \leq \gamma \leq 1$) bir sayı çifti olsun. Burada “ σ ” destek eşik değeri ve “ γ ” güven eşik değeri olarak adlandırılan kullanıcı tanımlı eşik değerleridir.

$s \rightarrow u$ şeklindeki bir ilişkisel kural aşağıdaki şartları sağlarsa $\sigma\gamma$ - geçerli olur:

- $|\lambda(s \rightarrow u)| / |D| \geq \sigma$
- $|\lambda(s \rightarrow u)| / |\lambda(s)| \geq \gamma$
- $|\lambda(s \rightarrow u)| = \lambda(s \cup u)$

$$\text{Destek } (s \rightarrow u) = |\lambda(s \rightarrow u)| / |D| \quad (3.3)$$

ifadesi $s \rightarrow u$ 'nun destek oranıdır.

$$\text{Güven } (|\lambda(s \rightarrow u)| / |\lambda(s)|) \quad (3.4)$$

ifadesi $s \rightarrow u$ 'nun güven oranıdır.

s ve u sırasıyla $s \rightarrow u$ kuralının girdisi ve sonucudur.

Örnek 3.1: D , Şekil 3.1'deki gibi tanımlı bir veri tabanı olsun. $\sigma = \%50$ ve $\gamma = \%100$ alındığında $\sigma\gamma$ - geçerli ilişkisel kurallar aşağıda listelenmiştir:

- $a \rightarrow b$ $c \rightarrow b$ $ab \rightarrow e$ $ae \rightarrow b$ $de \rightarrow b$ $abd \rightarrow e$
- $a \rightarrow e$ $d \rightarrow b$ $ad \rightarrow b$ $ce \rightarrow b$ $ad \rightarrow b$ $ade \rightarrow b$
- $a \rightarrow be$ $e \rightarrow b$ $ad \rightarrow e$ $de \rightarrow a$ $de \rightarrow ab$ $bde \rightarrow a$

$|\lambda(s \rightarrow u)| / |D|$, bir ilişkinin ilginç olması için s ve u 'nun yeterli sayıda işlemde aynı anda bulunmasını ifade eder. Bu “yeterlilik” ölçüsü kullanıcı tanımlı σ eşik değeri ile tanımlanır. $|\lambda(s \cup u)| \leq |\lambda(s)|$ ifadesi geçerlidir, çünkü “ $s \cup u$ ” yu içeren her işlem aynı zamanda s 'yi de içerir ama bu ifadenin tersi genellikle doğru değildir. Mantıksal olarak ilişkisel kural sadece $|\lambda(s \cup u)| = |\lambda(s)|$ sağlandığında geçerli olur. Bu olayda u , bir işlemde sadece s 'nin de olması durumunda kısmi cas olur. $|\lambda(s \rightarrow u)| / |\lambda(s)| \geq \gamma$ ifadesi bir ilişkisel kuralın yaklaşık olarak doğru olduğunu gösterir. Bu “yaklaşık” ifadesi bir kullanıcı tanımlı γ oranıyla tanımlanır. Sonraki 1 olduğunda ilişkisel kuralın resmen doğru olması gerekir. Aşağıda gösterildiği gibi destek($s \rightarrow u$) ve conf($s \rightarrow u$) ölçümleri olasılıkla ilişkilidir. Destek değeri aşağıdaki gibi ifade edilmiştir:

$$\text{destek}(s) = |\lambda(s)| / |D| = \Pr(s) \quad (3.5)$$

Destek ifadesine bağlı olarak cas ifadelerini genişletmek gerekmektedir. Bu amaçla $\wedge$ (kesişim), $\vee$ (birleşim) ve $\neg$ (ters) operatörleri kullanılır.

Destek($s \wedge u$) ifadesi, s ve u 'nun her ikini de kısmi cas'lar olarak içeren D 'deki işlemlerin alt kümelerini gösterir. Bu ifadenin gösterimi aşağıdadır:

$$\text{Destek}(s \wedge u) = \text{Destek}(s \cup u) \quad (3.6)$$

$$\text{Destek}(s \wedge u) = |\lambda(s) \cap \lambda(u)| / |D| = \Pr(s \wedge u) \quad (3.7)$$

$$\text{Destek}(s \cup u) = |\lambda(s \cup u)| / |D| = \Pr(s \cup u) \quad (3.8)$$

Buradan aşağıdaki sonuçlar elde edilir:

$$\text{destek}(s \rightarrow u) = |\lambda(s \cup u)| / |D| = \Pr(s \cup u) = \Pr(s \wedge u) \quad (3.9)$$

$$\begin{aligned}
\text{güven}(s \rightarrow u) &= |\lambda(s \cup u)| / |\lambda(s)| \\
&= (|\lambda(s \cup u)| / |D|) / (|D| / |\lambda(s)|) \\
&= \Pr(s \wedge u) / \Pr(s) = \Pr(u | s)
\end{aligned} \tag{3.10}$$

Destek($s \vee u$) ifadesi her bir s veya u 'yu veya her ikisini de kısmi cas'lar olarak içeren D 'deki işlemlerin alt kümelerini gösterir. Bu tanımdan şu sonuca ulaşılır:

$$\begin{aligned}
\text{Destek}(s \vee u) &= |\lambda(s) \cup \lambda(u)| / |D| \\
&= |\lambda(s)| / |D| + |\lambda(u)| / |D| - |\lambda(s) \cap \lambda(u)| / |D| \\
&= \text{destek}(s) + \text{destek}(u) - \text{destek}(s \cup u) \\
&= \Pr(s) + \Pr(u) - \Pr(s \wedge u) \\
&= \Pr(s \vee u)
\end{aligned} \tag{3.11}$$

Destek($s \neg u$) ifadesi s 'yi kısmi cas olarak içermeyen D 'deki işlemlerin alt kümesini gösterir. Buradan ise aşağıdaki sonuç elde edilir:

$$\text{Destek}(s \neg u) = |D - \lambda(s)| / |D| = 1 - \Pr(s) = \Pr(\neg s) \text{ (Adamo, 2000).} \tag{3.12}$$

3.2 Problem Tanımı

Verilen σ ($0 \leq \sigma \leq 1$) ve γ ($0 \leq \gamma \leq 1$) eşik oranları ile ilişkisel kuralları bulmak için bir D veri tabanının madenlenmesi işlemi, ($s \rightarrow u$) ilişkisini $\sigma\gamma$ -geçerli ilişkisel kural ($\text{destek}(s \rightarrow u) \geq \sigma$ ve $\text{conf}(s \rightarrow u) \geq \gamma$) yapmak için tüm s ve u çifti cas'larının elde edilmesine dayanır. Problem aşağıdaki iki adımla çözülebilir.

1. " v " bir σ -sık cas'ı ifade etmek üzere tüm σ -sık cas v 'ler bulunur. Tüm v 'ler şu şartı sağlamalıdır:

$$\text{Destek}(v) \geq \sigma$$

2. Birinci adımda bulunan tüm v cas'ları için tüm ($s \rightarrow u$) ilişkisel kuralları oluşturulur:

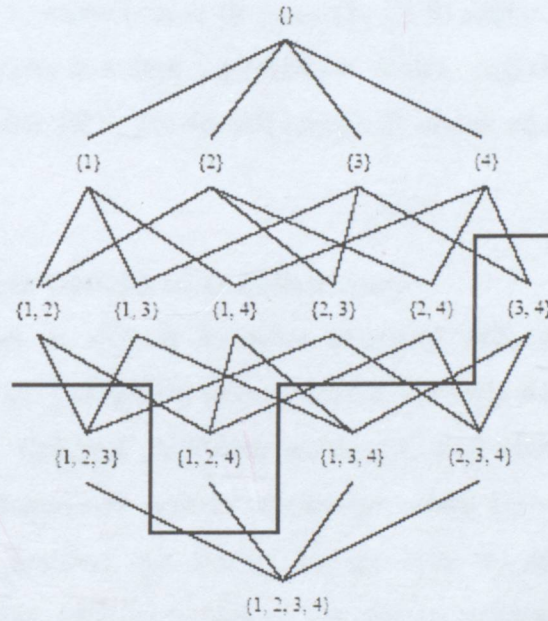
$$s < v \text{ ve } u = v - s, \text{ güven } (s \rightarrow u) \geq \gamma \text{ (Adamo, 2000).}$$

3.3 Arama Alanı

Problem tanımındaki prosedürün ilk adımı 2^A mantıksal kafesindeki tüm cas'ları birer birer saymayı gerektirir. Veri tabanı ve/veya özellik kümesi büyük bir arama alanını idare edecek kadar büyük olabilir. Ana hafıza gereksinimlerini azaltmak ve/veya sayma işlemini paralel olarak yapmak için arama alanı birçok bölüme ayrılabilir. Bu bölümler bağımsızca ve sıralı veya paralel olarak işleme tabi tutulabilirler (Adamo, 2000).

3.3.1 Arama alanının Üstünden Geçme (Traverse)

Minimum destek değerini sağlayan tüm kalem kümelerinin bulunması gerekir. Arama alanının çok büyük olması $I=\{x_1, \dots, x_n\}$ 'in tüm alt kümelerinin bulunmasını zorlaştırır. Gerçekte kalemlerin (item) sayısı üssel olarak büyür. Örneğin $I=\{1,2,3,4\}$ için arama alanının bir kafes biçiminde gösterimi şekil 3.2'deki gibidir.



Şekil 3.2. $I=\{1,2,3,4\}$ için kafes (Hipp,Güntzer,Nakhaeizadeh,2000)

Sık kalem kümeleri şeklin üst bölümünde, sık olmayan kalem kümeleri de alt bölümünde yer alır. Her ne kadar her kalem kümesi için destek değerleri açıkça belirlenmediyse de koyu çizginin sık kalem kümeleriyle sık olmayan kalem kümelerini birbirinden ayırdığı varsayılır. Bu sınır kalem kümesi desteğinin aşağı yönde kapanma (downward closure) özelliği ile oluşturulup, herhangi bir D veri tabanından ve minimum destek değerinden bağımsızdır.

Genel algortimaların temel prensibi, bu sınırı arama alanını etkili bir şekilde kısaltmak

(pruning) için kullanmaktır. Sınır bulunur bulunmaz üstünde olan kalem kümelerinin destek değerleri belirlenir ve sınırın altındakiler elenir.

Kalem kümelerinin aşağı yönde kapanma özelliği; eğer bir E sınıfının E' ana (parent) sınıfı, en az iki sık kalem kümesi içermezse E nin de hiçbir sık kalem kümesi içermediğini ifade eder. Eğer ağaçta aşağı yönde ilerlerken böyle bir E' sınıfı karşılaşırsa sık kalem kümesiyle sık olmayanları birbirinden ayıran sınıra ulaşılmış olur ve bu sınırın gerisinden gidilmez. Böylece E ve E'nin tüm torunları arama alanından çıkarılır.

Sonraki adım, aranacak kalem kümelerinin sayısının sınırlandırılmasını sağlar. Sadece sık ve sık olmayan kalem kümeleri arasında, sınır arama sırasında ziyaret edilen bu kalem kümelerinin destek değerleri belirlenir. Sınır arama stratejisi tamamen tercihe göre yapılır.

Günümüzün genel yaklaşımları hem genişlik öncelikli aramayı (breadth first search, BFS) hem de derinlik öncelikli aramayı (depth first search, DFS) sağlar. BFS ile tüm (k-1) kalem kümelerinin destek değerleri, k-kalem kümelerinin destek değerlerini hesaplamadan önce belirlenir. Bunun tam tersine DFS, yinelemeli (rekürsif) olarak ağaçtan aşağı doğru yukarıda anlatıldığı gibi alçalır.

3.3.2 Kalem Kümelerinin Desteklerinin Belirlenmesi

Potansiyel olarak sık olan ve kafesin üstünden geçerken desteğinin belirlenmesine karar verilen bir kalem kümesi, aday kalem kümesi (candidate itemset) olarak adlandırılır. Genel bir kalem kümesinin destek değerini belirleme yaklaşımı doğrudan onun veri tabanındaki oluşumlarını saymaktır. Bunun için o anda aranan her kalem kümesi için bir sayaç tutulur. Baştan sıfırlanan sayaç, bulunan her kalem kümesi için bir arttırılır ve hesaba dayalı adresleme ağacı (HDA, hash tree) veya benzer bir yapıya uygulanır. Her işlemin tüm alt kümeleri değil sadece adaylarda olanlar veya adayların en az biriyle ortak bir öneki (prefix) olanlar seçilir.

Diğer bir yaklaşım küme kesişimleriyle adayların destek değerlerini belirlemektir. Bir tid, tek bir işlem belirleyicisidir. Tek bir işlem için tid listesi bu kalemi içeren işlemlere karşılık gelen belirleyicilerin kümesidir. Bu nedenle tid listeleri her x kalem kümesi için vardır ve "x.tidlist" olarak gösterilir. Bir $C = x \cup y$ adayının tid listesi şöyle hesaplanır:

$$C.tidlist = x.tidlist \cap y.tidlist \quad (3.13)$$

Tid listeleri etkili kesişimler sağlamak için artan sırada sıralanır. Bir adayın gerçek desteği

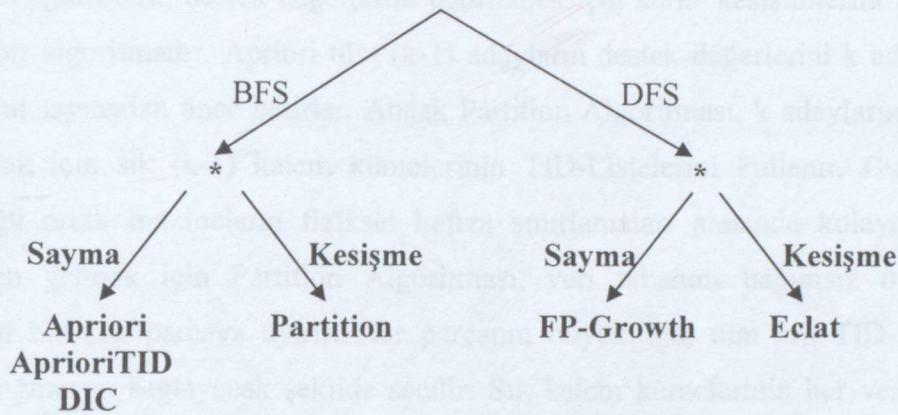
| C.tidlist | değeri belirlenerek elde edilir.

3.3.3 Sistematikleştirme

Burada ele alınacak algoritmalar şekil 3.3'deki gibi sistematikleştirilmiştir. Algoritmaların herbiri:

- Arama alanının üstünden geçme
- Kalem kümelerinin destek değerlerini belirleme

stratejilerine göre karakterize edilmiştir.



Şekil 3.3. Algoritmaların sistematikleştirilmesi (Hipp,Güntzer,Nakhaeizadeh,2000)

3.3.4 BFS ve Oluşum Sayma

Bu tipteki en tanınmış algoritma, kalem kümelerinin destek değerlerinin aşağı yönde kapanma özelliğini destekleyen Apriori'dir. Apriori algoritmasında bu özellik, sık olmayan alt kümesi olan adayların destek değerlerinin hesaplanmadan elenmeleriyle sağlanır. BFS bir adayın tüm alt kümelerinin destek değerlerinin ilerideki aşamalarda bilinmesini sağladığı için bu iyileştirme yapılabilir. Apriori, veri tabanını bir kez tarayarak bir k kaleminin tüm adaylarını her işlemdeki adaylara bakarak sayar ve bunun için bir HDA ağaç yapısı oluşturur. Bu ağacın yapraklarından birine ulaşırsa, işlemde bulunan ortak bir ön eke sahip adayların kümesi bulunmuş olur. Daha sonra bu adaylar tüm işlemlerde aranır ve bulunan adayların ağaçtaki sayacı artırılır.

AprioriTid algoritması, temel Apriori yaklaşımının her adımda veri tabanına sığınması özelliğini kullanmayan ve her işlemi o anda içerdiği adaylarla gösteren farklı bir Apriori çeşididir. AprioriHybrid algoritması da bu iki yaklaşımı birleştirilir.

SETM ise bir AprioriTid benzeri algoritma olup uygulamayı doğrudan SQL ile gerçekleştirir.

DIC algoritması, Apriori'nin adayları oluşturma ve sayma arasındaki katı ayırımını sadeleştiren ileri bir versiyonudur. Bu algoritma, bir aday henüz tüm işlemlerde görülmemiş olsa bile minimum destek değerine ulaştığı zaman, bir önek ağacı kullanarak ondan türeyen ek adayları oluşturmaya başlar. HDA ağac yapısının tam tersine ağacın her düğümü (yaprak veya iç düğüm) bir aday için ayrılır. Böylece bir düğüme ulaşıldığında bu düğümlerle ilişkili kalem kümelerinin işlemde var olduğu kesinleşir. Ayrıca, destek değeri belirleme ve aday oluşum işlemlerinin birbirine bağlanmasıyla veri tabanı taramalarının sayısı azaltılmış olur.

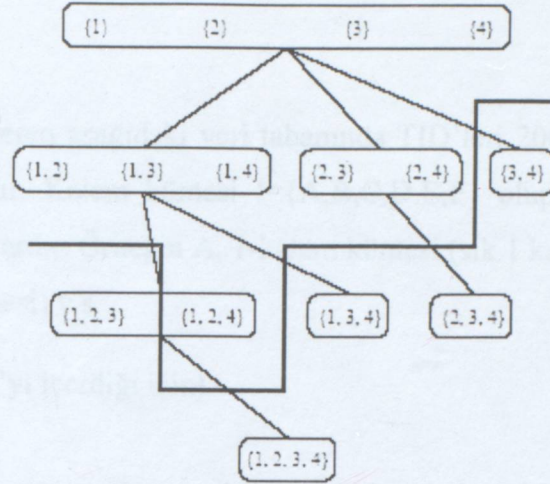
3.3.5 BFS ve TID-Listesi Kesişimleri

Partition Algoritması, destek değerlerini belirlemek için küme kesişimlerini kullanan Apriori benzeri bir algoritmadır. Apriori tüm $(k-1)$ adayların destek değerlerini k adaylarının destek değerlerini saymadan önce belirler. Ancak Partition Algoritması, k adayların TID-Listelerini oluşturmak için sık $(k-1)$ kalem kümelerinin TID-Listelerini kullanır. Bu ara sonuçların büyüklüğü ortak makinelerin fiziksel hafıza sınırlamaları arasında kolayca artar. Bunun üstesinden gelmek için Partition Algoritması, veri tabanını bağımsız olarak müdahale edilebilen bir çok parçaya ayırır. Her parçanın büyüklüğü, tüm ara TID-Listelerinin ana belleğe sığmasını sağlayacak şekilde seçilir. Sık kalem kümelerinin her veri tabanı parçası için belirlenmesinden sonra, yerel sık kalem kümelerinin evrensel (global) olarak da sık olmasını sağlamak için veri tabanının bir kere daha taranması gerekir.

3.3.6 DFS ve Oluşumları Sayma

Oluşum sayma işlemi aday kümelerinin makul büyüklükte olduğunu varsayarak bunların herbiri için bir veri tabanı taraması gerektirir. Örneğin BFS'yi temel alan Apriori, k büyüklüğündeki her aday için veri tabanını tarar. Ancak her düğüm için veri tabanını taramak çok büyük taşmalara neden olabileceği için DFS'nin oluşum sayma ile basit kombinasyonu pratik bir çözüm değildir.

Son yıllarda FP-Growth adında yeni bir yaklaşım ileri sürülmüştür. FP-Growth bir ön işlem adımında FP-tree olarak adlandırılan yüksek yoğunluklu bir işlem verisi elde eder. FP-tree'nin oluşturulması, oluşumları sayma ve DFS ile yapılır. Daha önceki DFS yaklaşımlarına zıt olarak FP-Growth şekil 3.4'deki ağacın düğümlerini takip etmez, fakat arama alanındaki kalem kümelerinin bazı parçalarına doğrudan iner. İkinci bir adımda FP-Growth tüm sık kalem kümelerinin destek değerlerini elde etmek için FP-tree yapısını kullanır.



Şekil 3.4. $I=\{1,2,3,4\}$ 'ın ağacı (Hipp,Güntzer,Nakhaeizadeh,2000)

3.3.7 DFS ve TID-Liste Kesişimleri

Eclat algoritması DFS ile TID-Liste kesişmelerini birleştirir. DFS kullanılırken TID-Listeyi, kökten (root) o anda bellekte aranan sınıfa (class) doğru olan yolda tutmak yeterli olur. Bu işlem Partition Algoritması'nın yaptığı gibi veri tabanını parçalara ayırmaktır.

Eclat "hızlı kesişmeler" olarak adlandırılan bir iyileştirme yöntemi kullanır. İki TID-Liste kesiştirildiğinde, sayılabilirliği minimum destek değerine ulaşan sonuç TID-Listeler kullanılır. Diğer bir deyişle her kesişim, bu sınıra ulaşıp ulaşamayacağından emin olunduğu anda elenebilir. Eclat genel olarak sadece büyüklüğü ≥ 3 olan sık kalem kümelerini oluşturur (Hipp,Güntzer,Nakhaeizadeh,2000).

3.4 Sepet Analizi

Sadece satın alma işlemlerine dayanarak müşteri davranışları hakkında bilgi edinmeyi sağlar. Bu bilgiler müşteri özelliklerini içermez. Yöneticiler, muhtemelen hangi ürün veya ürün gruplarının birlikte satıldığı ile ilgilenirler. Buna göre fiyat indirimi stratejileri uygulanır. Örneğin, eğer bilgisayar ve yazıcı birlikte alınıyorsa, yazıcıdaki fiyat indirimi bilgisayar satışlarını hızlandırabilir.

Örnek 3.2:

- $Al(bilgisayar) \Rightarrow Al(yazıcı) (destek = \%10, güven = \%60)$

Burada bütün işlemlerin $\%10$ u bilgisayar ve yazıcı içerir. Bilgisayar alanların $\%60$ 'ı yazıcı da

alır.

Örnek 3.3:

Satın alma işlemlerini içeren aşağıdaki veri tabanında TID'leri 2000, 1000, 4000, 5000 olan dört işlem bulunmaktadır. Kalem kümesi $I=\{A,B,C,D,E,F\}$ olup toplam $(2^6-1) = 63$ boş olmayan kalem kümesi vardır. Örneğin A, 1-kalem kümesi (sık 1 kalem kümesi), AB 2-kalem kümesi (sık 2 kalem kümesi) v.s

$\text{destek}(A) = 3$ (3 işlem A'yı içerdiği için)

İşlem ID	S.Alınan Parçalar
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Minimum destek değeri %50,
Minimum güven değeri %50,

- $A \Rightarrow C$ (%50, %66.6)
- $C \Rightarrow A$ (%50, %100)

Kural $A \Rightarrow C$ için:

$\text{Destek}(\{A \& C\}) = \%50$

$\text{Güven}(\{A \& C\}) = \text{destek}(\{A \& C\}) / \text{destek}(\{A\}) = \%66.6$

İşlem ID	S.Alınan Parçalar
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Sıkkalem Kümesi	Destek
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

Şekil 3.5 Örnek 3.3'e ait veri tabanındaki kalemler ve destek değerleri şekli [5]

3.5 İlişkisel Kuralların Türleri

İlişkisel kuralların çeşitleri aşağıda anlatılmıştır:

3.5.1 Hiyerarşik İlişkisel Kurallar

Uygulamaların çoğunda özellikler kümesi, bir sınıf hiyerarşisi içinde verilir. Sınıflar, hiyerarşik özellikler kümesine göre tek veya toplu görünüm yansıtılır. Örneğin Şekil 3.6, marketten alınan sepet verisinin hiyerarşisini tanımlayan örnek bir sınıfı gösterir.

Burada, çoklu sınıf seviyelerine yayılabilen özellikler içeren kuralların üretilmesi sağlanır.

Örneğin verilen bir σ destek ve γ güven eşik değerleri için aşağıdaki gibi $\sigma\gamma$ -geçerli kurallar bulunmaya çalışılır:

- *Alkolsüz içecek -> meze destek değeri = %50, güven değeri = %90 veya*
- *Alkollü içecek -> yerfıstığı destek değeri = %30, güven değeri = %90*

Büyük destek değerleri, yüksek sınıf seviyesinde daha çok mecuttur. Bu nedenle;

- *alkollü içecek -> meze*

kuralı bir $\sigma\gamma$ - geçerli kural olabilirken

- *markal -> meze veya meşrubat -> meze*

kuralları $\sigma\gamma$ -geçerli olmayabilir. Çünkü ilk kural minimum destek değerine, ikinci kural ise minimum güven değerine sahip olmayabilir.

Çoklu sınıflar, farklı görünümüler yansıtacak şekilde eş zamanlı olarak gösterilirler. Örneğin şekil 3.6'da maden suları basit içecekler olarak düşünülen fakat eş zamanlı olarak diyet yiyecekler sınıfında da özel olan parçalardır. Burada giden yöndeki dallanma bir "veya (or) tipi" dallanmadır. Örneğin içecekler; maden suları veya alkolsüz içecekler veya alkollü içecekler olabilir. Gelen yöndeki dallanma ise bir "ve (and) tipi" dallanmadır. Örneğin; maden suları bir taraftan içecekler sınıfına ait iken diğer taraftan da eş zamanlı olarak diyet yiyecekler sınıfına aittir.

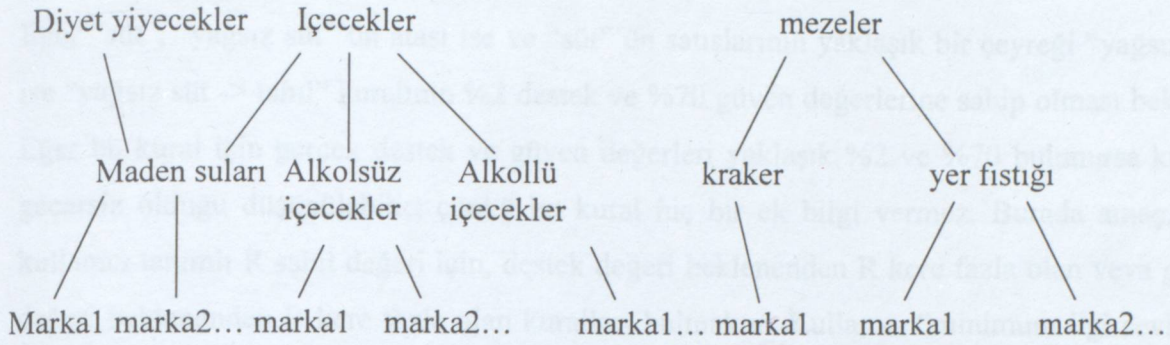
Olasılık mantığına göre sınıflandırma işlemi, ilişkisel kuralların gücünü kesişimler ile birleştiren bir yoldur. Örneğin;

- *İçecekler -> mezeler*

kuralı aynı zamanda

- *(maden suları veya alkolsüz içecekler veya alkollü içecekler veya...) -> (yer fıstığı veya kraker veya...)*

olarak da okunabilir. Aynı zamanda bu, kesişimi ortaya koymayı kısıtlayan bir yoldur. Çünkü kesişen ifadeler birleşen ifadeler gibi kolayca bulunamazlar. Kesişimin kullanıcı-tanımlı sınıflar yoluyla ilişkisel kurula tanıtılması gerekir.



Şekil 3.6 Örnek bir sınıflandırma (Adamo,2000)

İşletme açısından, sınıflandırmanın kısaltma avantajı olabilir. Sınıflar gereksiz olan ilişkisel kuralların iptal edilmesini sağlayabilir. Örneğin; yeteri kadar büyük bir destek değeri ile daha genel kurallardan (daha yüksek sınıf seviyesi özellikleri içeren kurallar) tahmin edilebilecek kurallar gibi (Adamo,2000).

3.5.2 Çok Seviyeli İlişkisel Kural

Bazı işlem veri tabanları hiyerarşik yapıları içerirler ve bu hiyerarşinin farklı seviyelerine yayılan genelleştirilmiş ilginç ilişkisel kurallar ortaya çıkar.

Yüksek seviyeli kurallar:

- “Süt satın alan müşterilerin %80’i ekmek de satın alabilir”

gibi kurallardır.

Düşük seviyeli kurallar:

- “Eğer müşteriler %2’si süt alırsa onların %70’i beyaz ekmek de alır”

şeklindeki kurallardır.

Yüksek seviyeli kurallar yüksek destek değerine sahip olabilir. Düşük seviyeli kurallar ise minimum desteğe sahip olmayabilir fakat daha bilgi verici olabilirler.

x' , x in atası olsun. Eğer $\{x,y\}$ minimum destek değerine sahip olursa $\{x,y'\}$, $\{x',y\}$ ve $\{x',y'\}$ de minimum destek değerine sahip olur. Ama eğer $x \rightarrow y$ kuralı minimum destek ve güven değerlerine sahip olursa yalnızca $x \rightarrow y'$ kuralının her iki minimum destek ve güven değerlerine sahip olması kesinleşir. $x' \rightarrow y$ ve $x' \rightarrow y'$ kuralları minimum destek değerine sahip olurken minimum güven değerine sahip olmayabilirler. Örneğin;

- *Süt -> tahıl (%8 destek, %70 güven)*

Eğer “süt”, “yağsız süt”ün atası ise ve “süt”ün satışlarının yaklaşık bir çeyreği “yağsız süt” ise “yağsız süt -> tahıl” kuralının %2 destek ve %70 güven değerlerine sahip olması beklenir. Eğer bu kural için gerçek destek ve güven değerleri yaklaşık %2 ve %70 bulunursa kuralın geçersiz olduğu düşünülebilir; çünkü bu kural hiç bir ek bilgi vermez. Burada amaç, bazı kullanıcı tanımlı R sabit değeri için, destek değeri beklenenden R kere fazla olan veya güven değeri beklenenden R kere fazla olan kuralları bulmaktır. Kullanıcı “minimum ilgi seviyesi” R’yi belirleyebilir [4].

3.5.3 Negatif İlişkisel Kural

Geleneksel ilişkisel kural madenciliği, aralarında sık ve yüksek korelasyon olan sıkıca bağlı kalem kümeleri arasındaki ilişkileri bulmayı amaçlar. İlişkisel kurallar, bir uygulamada birlikte ortaya çıkan parçaların belirlenmesine yardımcı olurlar. Destek değeri-Güven değeri çatısından oluşan kurallar pozitif kurallardır. Bazı kalem kümelerinin varlığını, veri tabanının içinde başka kalem kümelerinin de olabileceğini gösterir.

Uygulamalarda $A \rightarrow B$ şeklindeki bir kural “eğer A oluşursa B de genellikle oluşur” tahmininde kullanılır. Stok planlama, ürün yerleştirme, süpermarket yönetimi gibi uygulamalarda bu ilişkisel kural, B’yi A’nın yanına yerleştirmek için kullanılabilir. Aşağıda bir market için ilişki analizi önerilerine örnekler verilmiştir:

- *Yüksek sınıf ürünler veya ürün kümeleri satıcıya yakın yerleştirilebilir.*
- *A ve B ürünleri için $A \rightarrow B$ şeklinde bir kural varsa B, A’nın yanına yerleştirilebilir.*

İlk strateji, her gün sıkça satılan ürünlere satıcı tarafından kolay ulaşılabilmesini amaçlar ve çok etkili bir pazarlama imkanı sağlar. İkinci strateji ise, birlikte sıkça satın alınan bir grup parçanın seçilerek birlikte sergilenmesi nedeniyle satıcının zaman kaybını önlemektedir. Örnek 3.4 ve Örnek 3.5 farklı tipteki ilişkisel kuralları göstermektedir.

Örnek 3.4:

A,B,C,D,E,F ‘den oluşan altı ürün, $A \rightarrow B$ ve $E \rightarrow F$ şeklinde iki ilişkisel kural ve süper markette bu altı ürünü yerleştirmek için iki raf mevcuttur. A ile B ve E ile F aynı rafta yerleştirilebilirler.

İki pozitif ilişkisel kuralda C ve D ürünleri diğer hiçbir ürünle pozitif ilişkili değildir.

$A \rightarrow \neg C$ şeklindeki kurallara negatif ilişkisel kural denir ve bu kural A'nın bir işlemde olması durumunda C'nin aynı işlemde yüksek olasılıkla olmadığını ifade eder. Örneğimizde A ve C arasında negatif bir ilişki olsaydı (ve C'nin başka negatif ilişkisi olmaması kaydıyla) C'yi E ve F ile aynı rafa koyabilirdik.

Örnek 3.5:

Bir emlak işindeki yatırımcılar; çevre kalitesi, doğal kaynak kullanımı, ekonomik ve politik doğa ile ilgili problemlerle karşılaşabilirler. Çevresel problemler, kaynak kullanımı uyumsuzluğu ve bu uyumsuzluk ve sorunların politik çözümlerini arttıran durumların analizi, geçmiş verilerden elde edilen sadece pozitif değil, aynı zamanda negatif ilişkisel kurallara da bağlı olmalıdır.

Bir karar vericinin, avantaj faktörlerinin yanında hangi dezavantaj faktörlerinin de nadiren ortaya çıkabileceğini bilmek istemesi durumunda yukardaki örnekteki negatif ilişkisel kurallar büyük önem kazanmaktadır. Çünkü negatif ilişkisel kurallar sık olmayan kalem kümelerinde saklıdır ve varolan veri madenciliği teknikleri bu kuralları belirlemede işe yaramazlar.

Birçok uygulamada karar verme süreci (ürün yerleştirme ve yatırım analizi gibi) bir miktar avantaj ve dezavantaj faktörü içerir. Negatif etkileri minimuma indirmek ve pozitif yararları maksimuma çıkarmak için, hangi yön etkisi faktörlerinin beklenen avantaj faktörlerinin oluşması durumunda nadiren meydana geleceği dikkate alınmalıdır. Bu nedenle $A \rightarrow \neg C$ şeklindeki negatif ilişkisel kurallar karar verme sürecinde çok önemlidir. Çünkü $A \rightarrow \neg C$ (C dezavantaj faktörü olabilir) kuralı A'nın oluşması durumunda C'nin nadiren oluştuğunu gösterir. Bu nedenle örnek 3.4 ve örnek 3.5, farklı pozitif ve negatif faktörler arasındaki ilişkilerin faydalı olduğunu gösterir.

İkinci olarak, bilim ve mühendislik alanındaki deneyimler negatif ilişkilerin (matematikteki negatif sayılar ve mantıktaki negatif görüşler) pozitif ilişkilerle aynı önem rolüne sahip olduğunu gösterir.

İlişkisel kuralların veri tabanında nasıl saklı olduğu Örnek 3.6'da açıklanmıştır.

Örnek 3.6:

Bir markete ait n sepet içeren bir sepet verisinde çay 't' ile kahve de 'c' ile gösterilmektedir.

- $destek(t) = 0.25$

ve

- $destek(t \cup c) = 0.2$

olmak üzere $t \rightarrow c$ için

- $destek = 0.2$
- $güven = destek(t \cup c) / destek(t) = 0.2 / 0.25 = 0.8$

Her iki destek ve güven değerleri de çok yüksek olduğundan $t \rightarrow c$ 'nin geçerli bir kural olduğu sonucuna varılır.

- $destek(c) = 0.6$
- $destek(t) = 0.4$
- $destek(t \cup c) = 0.05$
- $minimum\ güven = 0.52$

olarak kabul edilirse bu durumda şartlı olasılık;

- $destek(t \cup c) / destek(t) = 0.05 / 0.4 = 0.125 < mingüven = 0.52$

ve

- $destek(t \cup c) = 0.05$

değeri çok düşük olduğundan $t \cup c$ sık olmayan bir kalem kümesidir ve $t \rightarrow c$ kuralı geçerli değildir. Fakat;

- $destek(t \cup \neg c) = destek(t) - destek(t \cup c) = 0.4 - 0.05 = 0.35$

değeri çok yüksektir ve

- $destek(t \cup \neg c) / destek(t) = 0.35 / 0.4 = 0.875 > mingüven$

değeri olduğundan $t \rightarrow \neg c$ kuralı elde edilebilir.

$t \rightarrow \neg c$ kuralı negatif bir kuraldır. Bu nedenle negatif ilişkisel kural elde etmek için sık olmayan kalem kümeleri ($t \cup c$ gibi) incelenmelidir.

Bu örnekte de gösterildiği gibi negatif ilişkisel kuralları belirlemek için veri tabanında sık olmayan kalem kümelerinin belirlenmesi gerekir (Zhang, 2002).

3.5.4 Nicel (Quantitative) İlişkisel Kural

İlişkisel veri madenciliği konusunda yapılan araştırmaların bir çoğu Boole ilişkisel kurallar üzerinde yapılmıştır (Agrawal ve Srikant, 1994; Agrawal et al., 1996; Brin et al., 1997; Savasere et al., 1995). Boole ilişkisel kurallara bir örnek aşağıdadır:

- “Eğer bir müşteri bir diş fırçası (bir veya fazla) alırsa, o zaman diş macunu da alır (aynı işlemde)”

Bu kuralın genel gösterimi:

- “Diş fırçası $\rightarrow$ Diş macunu”

biçimindedir.

Boole ilişkisel kurallar bir işlemde satın alınan parçaların miktarı ile ilgilenmez. Pekçok yazar (Agrawal et. Al. gibi), nicel ilişkisel kuralların bazı uygulamalar için önemli olduğunu ve bunları bulmanın ileri çalışmalarda gerekeceğini savunurlar. Örnek 3.7’de nicel kurala bir örnek verilmiştir.

Örnek 3.7:

- “Eğer bir müşteri iki, üç veya dört diş fırçası alırsa, o zaman üç ile altı arasında diş macunu tüpü alır.”

Bu kuralın gösterim şekli;

- Diş fırçası: $[2,4]$ $\rightarrow$ Diş macunu: $[3,6]$

olup nicel ilişkisel kural olarak adlandırılır. Nicel ilişkisel kuralların bulunması problemi, yüksek bir uygulama potansiyeliyle ilginç ve önemli bir araştırma konusu olmuştur. Nicel ilişkisel kuralların kullanımı market sepet verisiyle sınırlanmaz. Hemen hemen bütün veri tabanı uygulamaları; maaş, ağırlık ve benzeri sayısal özellikler içerirler. Ayrıca, bazı ileri veri modelleri de özel sayısal özelliklere dayanırlar (J.Wijssen, R. Meersman, 1998).

4. İLİŞKİSEL KURAL MADENCİLİĞİ ALGORİTMALARI

4.1 Apriori Temelli Algoritmalar

Apriori temelli algoritmalar beşinci bölümde ayrıntılı olarak anlatılacağı için bu bölümde onlara kısaca değinilmiş ve algoritmaların temel işlevleri anlatılmıştır.

AIS:

AIS algoritmasında aday kalem kümeleri veri tabanı taranırken anında bulunup sayılır. Bir işlem okunduktan sonra, bir önceki taramada sık olduğu görülen kalem kümelerinden hangilerinin bu işlemde olduğuna bakılır. Yeni aday kalem kümeleri bu sık kalem kümelerinin işlemdeki diğer kelimelerle birleştirilmesiyle elde edilir.

SETM:

Bu algoritma sık kalem kümelerinin hesaplanmasında SQL kullanılmasını amaçlar. AIS gibi bu algoritma da veri tabanından okunan işlemlere göre adayları anında oluşturur. SETM algoritması aday üretimi için standart SQL birleştirme işlemini kullanmak için aday oluşturma işlemini sayma işleminden ayırır.

Apriori:

AIS ve SETM algoritmalarının dezavantajı zamanla küçülen çok sayıda aday kalem kümesini oluşturup saymalarıdır. Bu nedenle performansı iyileştirmek için Apriori algoritması önerilmiştir. Apriori algoritması yalnızca veri tabanının üstünden bir önceki geçişte elde edilen sık kalem kümelerini kullanarak veri tabanındaki işlemleri göz önüne almadan, sayılacak yeni adayları oluşturur. Apriori'nin AIS ve SETM'e göre özellikle büyük veri kümeleri için daha iyi olduğu açıktır.

Apriori'nin anafikri; desteğin "aşağı doğru kapalı" özelliğine dayanır. Minimum desteğe sahip kalem kümeleri sık kalem kümesi olup, bu sık kalem kümelerinin alt kümeleri de sık olmalıdır. k kalem içeren aday kalem kümeleri, $k-1$ kalem içeren sık kalem kümelerinin birleştirilip, sık olmayan herhangi bir altküme içerenlerinin silinmesiyle elde edilebilir.

Önce tüm sık 1-kalem kümeleri bulunduktan sonra ikili, dörtlü şekilde devam eden diğer kalem kümeleri de bulunur. Böylece her iterasyon esnasında yalnızca önceki iterasyonda sık görülen adaylar bir sonraki iterasyonda yeni bir aday oluşturmak için kullanılırlar. Algoritma hiç sık kalem kümesi kalmayınca sonlanır.

Çizelge 4.1 Apriori Algoritması [4]

```

L1 = {frequent 1-itemset};
For (k =2; Lk-1 !=∅; k++) do begin
    Ck = apriori_gen (Lk-1); //Yeni adaylar
    Forall transactions t ∈ D do begin
        Ct = subset (Ck,t); //t’de bulunan adaylar
        Forall candidates c ∈ Ct do
            c.count++;
    end
    Lk = {c ∈ Ck | c.count ≥ mindestek }
End
Answer = ∪k Lk;

```

Apriori-gen fonksiyonu L_{k-1} 'i parametre olarak alır ve tüm sık k-kalem kümelerinin bir süper kümesini geri döndürür. Bu, birleştirme adımı ve kısaltma adımından oluşur. Çizelge 4.2 de gösterilen birleştirme adımında L_{k-1} ile L_{k-1} 'i birleştirir:

Çizelge 4.2 Birleştirme adımı [4]

```

Insert into Ck
Select p.item1, p.item2,...,p.itemk-1, q.itemk-1
From Lk-1 p, Lk-1 q
Where p.item1 = q.item1,...p.itemk-2 = q.itemk-2, p.itemk-1 < q.itemk-1

```

Kısaltma adımında bazı $(k-1)$ -alt kümeleri L_{k-1} 'in içinde olmayan $c \in C$ şartını sağlayan tüm c 'leri siler. Alt küme fonksiyonu HDA ağaç yapısını kullanarak işlem veri tabanındaki tüm k-kalem kümesi adaylarını bulur.

Apriori Algoritmasının Türleri:

AprioriTID ve AprioriHybrid algoritmaları Apriori algoritmasının iki türüdür. Sonraki bölümde detaylı olarak anlatılacak olan bu algoritmalara bu bölümde kısaca değinilmiştir.

1 - AprioriTID:

Bu algoritmanın en ilginç özelliği, ilk geçişten sonra aday kalem kümelerinin destek değerlerinin hesabı için D veri tabanının kullanılmamasıdır. Onun yerine bir önceki geçişte kullanılan aday kalem kümelerinin kodlandığı bir küme kullanılır. Sonraki geçişlerde bu kodlanan kümenin boyutu veri tabanından çok daha küçük hale gelebilir, böylece okuma gücünden kazanç sağlanır.

2 - AprioriHybrid:

Algoritmaların çalışma zamanlarının performans analizine göre, önceki geçişlerde Apriori daha iyi performans gösterir; ama sonraki geçişlerde AprioriTid Apriori'yi geçer. Ayrıca her ikisi de aynı aday oluşturma prosedürünü kullanır. Bunları göz önünde bulundurarak AprioriHybrid algoritması, Apriori'nin ve AprioriTid'in en iyi özelliklerini birleştirmeyi amaçlar. AprioriHybrid ilk geçişlerde Apriori'yi kullanır, sonraki geçişlerde de C_k kümesinin (orjinal kalem kümesinin kodlanmış kümesi) geçişin sonunda belleğe sığabileceğinin anlaşılması üzerine AprioriTid'e geçiş yapar. Ancak Apriori'den AprioriTid'e geçiş yapmak da bir maliyet getirir.

DHP :

Sık kalem kümesi üretiminde, sık kalem kümelerinden aday kümelerini üretme yöntemi performans açısından çok önemlidir. Aday kalem kümeleri ne kadar büyük olursa sık kalem kümelerini üretmek için daha çok işlem maliyeti gerekir. Önceki iterasyonlardaki işlem, gerçekte toplam çalışma maliyetinin başında gelir. Bu durum başlangıçtaki aday kalem kümelerinin üretilmesi işleminin (özellikle 2-kalem kümeleri), performansı arttırmak için anahtar bir çözüm olduğunu gösterir. Bunları göz önüne alarak DHP önerilmiştir. DHP, HDA tabanlı bir algoritma olup özellikle de sık 2-kalem kümelerinin üretilmesinde etkilidir. DHP'nin biri etkili sık kalem kümesi üretimi, diğeri etkili işlem veri tabanı boyutunu azaltma olan iki önemli özelliği vardır.

$L_{k-1} * L_{k-1}$ den elde edilen tüm k-kalem kümelerinin Apriori'deki gibi C_k 'ya eklenmesinin yerine, DHP bu k-kalem kümesini yalnızca HDA süzgecinden geçmesi durumunda C_k 'ya ekler. Bu HDA süzme işlemi C_k boyutunu önemli ölçüde azaltır.

* Bu algoritmalara ait pseudo kodları 5. bölümde verilmiştir.

Örnek 4.1: Aşağıda minsup = 3 olan örnek bir veri tabanı verilmiştir.

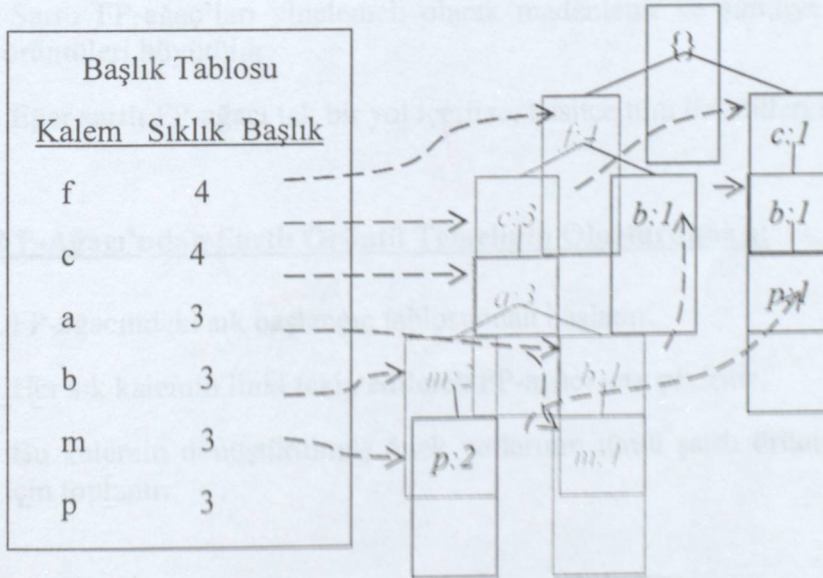
Çizelge 4.3 Örnek 4.1'e ait veri tabanı örneği

TID	Satın alınan Kalemler	Sık Kalemler
100	{f,a,c,d,g,i,m,p}	{f,c,a,m,p}
200	{a,b,c,f,l,m,o}	{f,c,a,b,m}
300	{b,f,h,j,o}	{f,b}
400	{b,c,k,s,p}	{c,b,p}

Aşağıdaki adımlar ışığında sık bir veri yapısı tasarlanabilir:

Adımlar:

- Veri tabanını bir kere taranır ve sık 1-kalem kümeleri bulunur.
- Sık kalem kümeleri azalan sırada sıralanır.
- Veri tabanı tekrar taranır ve FP-ağaç yapısı oluşturulur.



Şekil 4.1 Örnek 4.1'e ait FP-ağaç yapısı

Bir FP-ağaç yapısı şu şekilde tanımlanabilir:

- “null” olarak adlandırılan bir kök düğüm, bu kök düğümün çocuğu olan bir kalemin önek alt ağaçları kümesi ve bir sık-kalem başlık tablosu.
- Kalem önek alt ağacındaki her düğüm üç alan içerir: kalem ismi, kalem sayısı, ve düğüm linki. Burada kalem ismi bu düğümün hangi kalemi gösterdiğini ifade eder. Kalem sayısı bu düğüme erişen yolun gösterdiği işlemlerin sayısını tutar. Düğüm linki ise FP-ağacında aynı isimdeki bir sonraki düğüme link kurar veya aynı isimde başka düğüm yoksa bunun değeri null olur.
- Sık-kalem başlık tablosundaki her giriş iki alan içerir: (1) kalem-isim ve (2) düğüm-linkinin başlangıç yeri. Bu alan bu kalem ismini taşıyan FP-ağacındaki ilk düğümü gösterir.

FP-ağaç yapısı oluşturulduktan sonra artık bu ağaç madenlenebilir (J.Han,J. Pei,Y. Yin,1999)

4.2.1.2 FP-Ağaç Yapısının Madenlenmesi Adımları

- FP-ağacı’ndaki her düğüm için şartlı örüntü temeli oluşturulur.
- Her şartlı örüntü-temelinden şartlı FP-ağacı oluşturulur.
- Şartlı FP-ağaç’ları yinelemeli olarak madenlenir ve şimdiye kadar sağlanan sık örüntüleri büyütülür.
- Eğer şartlı FP-ağacı tek bir yol içerirse, basitçe tüm örüntüleri sayılır.

1. Adım: FP-Ağacı’ndan Şartlı Örüntü Temelinin Oluşturulması:

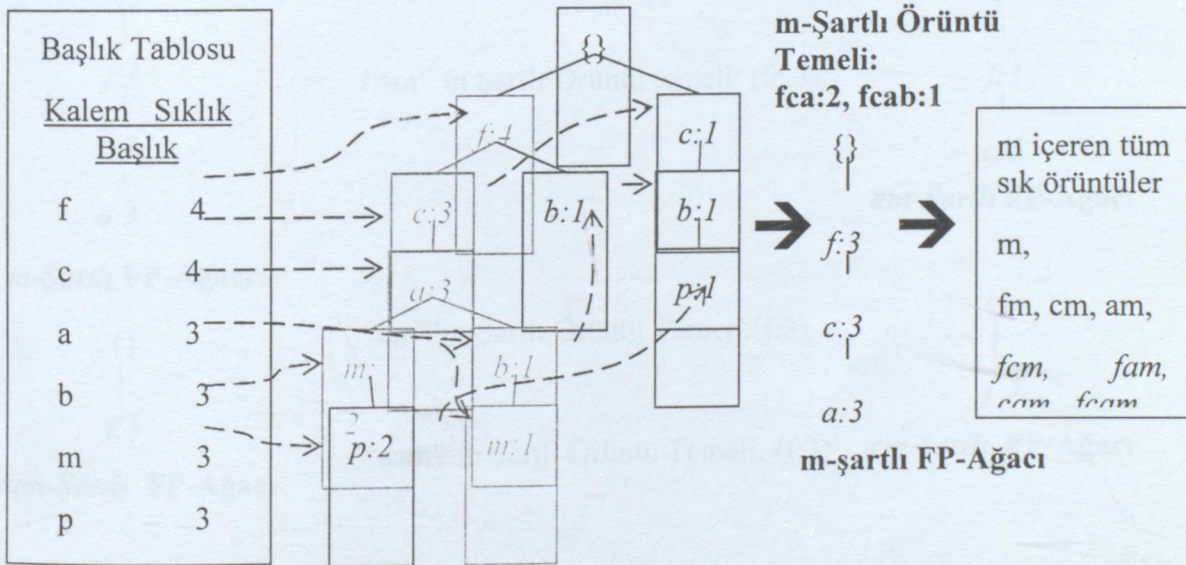
- FP-ağacındaki sık başlangıç tablosundan başlanır.
- Her sık kalemin linki takip edilerek FP-ağacı ters çevirilir.
- Bu kalemin dönüştürülmüş önek yollarının tümü şartlı örüntü temeli oluşturmak için toplanır.

Çizelge 4.4 Algoritmanın birinci adımında oluşturulan şartlı örüntü temeli

Kalemler	Şartlı Örüntü Temeli
c	f:3
a	fc:3
b	fca:1, f:1, c:1
m	fca:2, fcab:1
p	fcam:2, cb:1

2. Adım: Şartlı FP-Ağacının Oluşturulması:

- Her örüntü-temeli için;
- Temeldeki her kalem için sayı toplanır.
- Örüntü temelinin sık kalemleri için FP-Ağacı oluşturulur.



Şekil 4.2 Şartlı Örüntü-Temelleri Yaratarak Sık Örüntüleri Madenleme (J.Han, J.Pei, Y.Yin, 1999)

Çizelge 4.5 Şartlı Örüntü temelinden şartlı FP-ağacı oluşturma

Kalem	Şartlı Örüntü-Temeli	Şartlı FP-Ağacı
p	{{(fcam:2), (cb:1)}}	{{(c:3)} p
m	{{(fca:2), (fcab:1)}}	{{(f:3, c:3, a:3)} m
b	{{(fca:1), (f:1), (c:1)}}	Empty
a	{{(fc:3)}}	{{(f:3, c:3)} a
c	{{(f:3)}}	{{(f:3)} c
f	Empty	Empty

3. Adım: Şartlı FP-Ağacını Yinelemeli Olarak Madenleme:

{}
|
f:3
|
c:3
|
a:3

m-Şartlı FP-Ağacı

{}
|
f:3

cam-Şartlı FP-Ağacı

“am” in Şartlı Örüntü temeli: (fc:3)

{}
|
f:3
|
c:3

am-Şartlı FP-Ağacı

“cm” in Şartlı Örüntü Temeli:(f:3)

{}
|
f:3

“cam” in Şartlı Örüntü Temeli: (f:3)

cm-Şartlı FP-Ağacı

4.2.2 CHARM

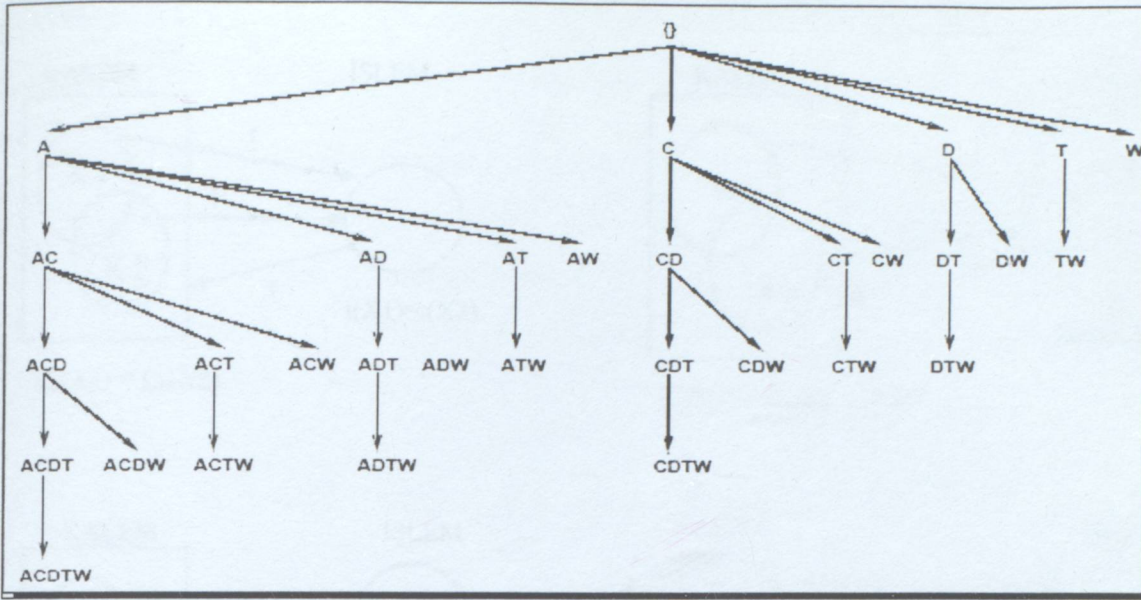
Bu algoritmanın hedefi kapalı sık kalem kümelerini madenlemek ve bunlardan ilişkisel kurallar elde etmektir. Bu süreçteki temel hesaplama adımı kapalı sık kalem kümelerinin bulunmasıdır. Bu kümenin oluşturulması Apriori benzeri aşağıdan-yukarı arama metodlarıyla pek mümkün değildir.

CHARM tüm kapalı sık kalem kümelerinin sayılması için etkili bir algoritmadır. CHARM'ın diğer algoritmalarda olmayan önemli bir özelliği, aynı zamanda hem kalem kümesi alanını hem de işlem alanını aramasıdır. Diğer algoritmalar ise sadece kalem kümesi arama alanını ararlar. Ayrıca CHARM kapalı sık kümelerini sayarken bir kapalı kalem kümesinin tüm alt kümelerini saymaktan sakınır.

Her iki kalem kümesini ve işlem alanını arama işlemi CHARM'ın kapalı olmayan pek çok alt kümeyi saymak yerine, kapalı sık kalem kümelerini hızlı bir şekilde belirlemek için bir çok seviyeyi atlayan yeni bir arama metodu kullanmasını sağlar. Ayrıca CHARM iki kısaltma stratejisi kullanır. Sadece tüm ilişkisel madencilik metodlarının yaptığı gibi sık olmayan alt kümeler üzerinde temellenen adayları değil, aynı zamanda kapanma özelliği olmayanların üzerindeki de sadeleştirir. Sonuç olarak CHARM, HDA ağaç yapısı gibi iç veri yapılarını kullanmaz. Burada önemli bir işlem, iki kalem kümesinin birleşimi ve kalem kümelerinin içerildiği iki işlem listesinin kesişimidir (Zaki,Hsiao, 1999).

FARKLI VERİTABANI KALEMLERİ					
	A	C	D	T	W
VERİTABANI	TÜM SIK KALEM KÜMELERİ				
İşlem Kalemler	MİNİMUM-DESTEK = %50				
1	A C T W	Destek	Kalem Kümeleri		
2	C D W	%100 (6)	C		
3	A C T W	%83 (5)	W,CW		
4	A C D W	%67 (4)	A, D, T, AC, AW,		
5	A C D T W		CD, CT, ACW		
6	C D T	%50 (3)	AT, DW, TW, ACT, ATW		
			CDW, CTW, ACTW		

Şekil 4.3 Sık kalem kümelerini oluşturma (Zaki,Hsiao, 1999)



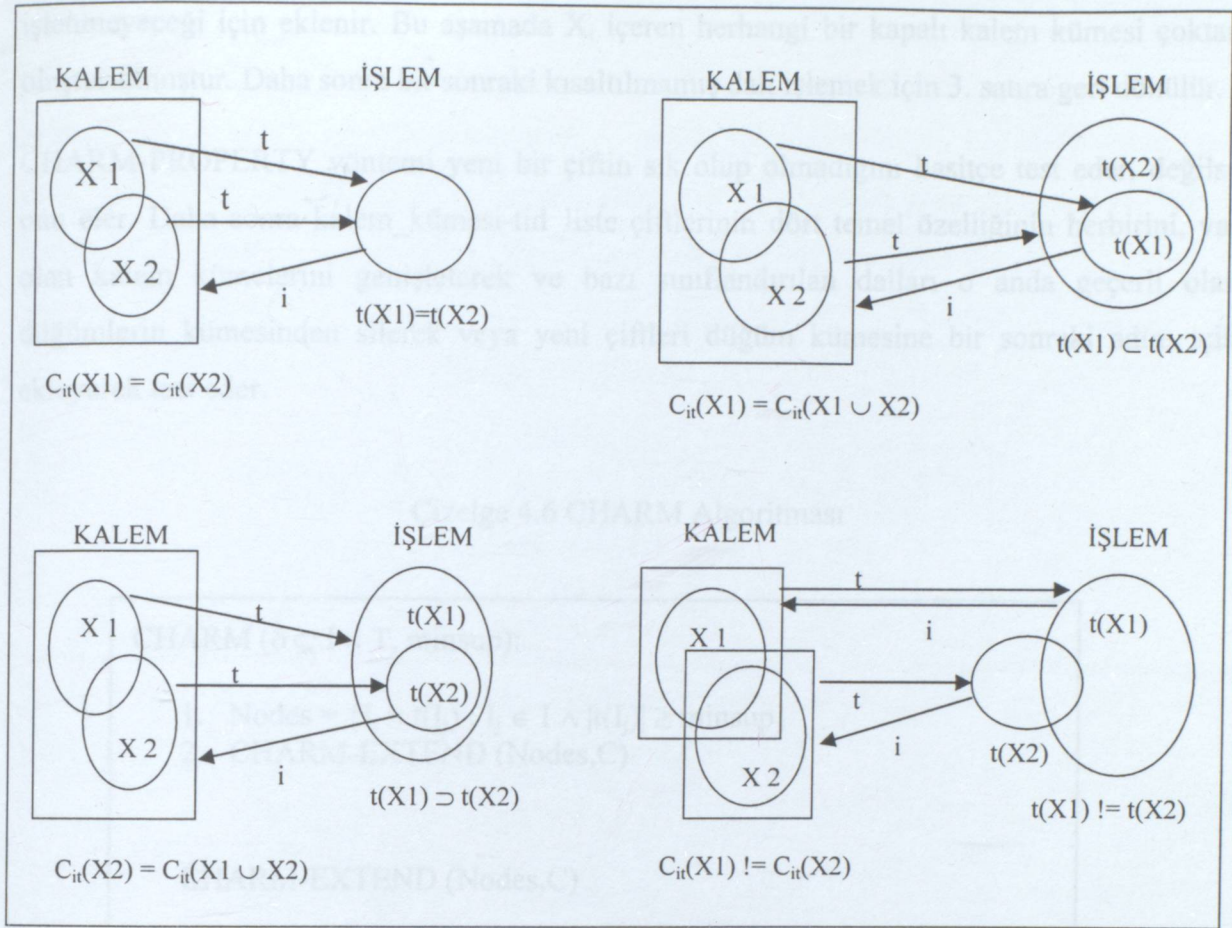
Şekil 4.4 Tamamlanmış alt küme kafesi (Zaki, Hsiao, 1999)

Şekil 4.4, şekil 4.3’de verilen örnek veri tabanımızdaki 5 kalem üzerinde tamamlanmış altküme kafesini gösterir (yalnızca ana ebeveyn linki karmaşıklığı azaltmak için gösterilmiştir). CHARM’daki ana düşünce her kafes düğümünün, çocuklarının sık olup olmadığını test etmek için işlemektir. Tüm sık olmayan, aynı zamanda kapalı olmayan dallar azaltılır. Her düğümün çocuğu aynı dal üzerinde kendinden sonra gelen kardeşlerinin düğümleriyle birleştirilerek elde edilir. Örneğin A, kendi kardeşleri olan C, D, T ve W ile birleştirilerek AC, AD, AT ve AW çocukları üretilir. Bir çok arama metodu mevcutken (örneğin; genişlik-öncelikli, derinlik-öncelikli, en iyi-öncelikli veya diğer hybrid aramaları gibi) CHARM alt küme kafesini derinlik-öncelikli olarak arar.

4.2.2.1 CHARM Algoritma Tasarımı

Herhangi bir X kalem kümesi için onun tid kümesi olan $t(X)$ ’ye ve herhangi bir Y tid kümesi için onun kalem kümesi olan $i(Y)$ ’ye erişimimiz olduğunu varsayalım.

CHARM giriş veri tabanındaki tüm sık kavramları sayar. Bir kavram bir $X \times Y$ çifti olarak verilir. Burada $X = i(Y)$ bir kapalı kalem kümesidir ve $Y = t(X)$ kapalı bir tid kümesidir. Kavramları aramaya tid kümesi alanından veya kelem kümesi alanından başlanabilir. Ama, tipik olarak kelimelerin sayısı işlemlerin sayısından çok küçüktür ve sonuçta kapalı kalem kümeleriyle ilgilendiğimizden, aramaya tek kelimeler ve onlarla ilişkili tid kümeleriyle başlanılır.



Şekil 4.5 Kalem kümeleri ve Tid kümelerinin temel özellikleri

4.2.2.2 Pseudo-Kod Tanımı

Algoritma, sık tek kalemlerin ve onların tid kümelerinin (hedef tanıtıcı kümeler) incelenmesi için düğüm kümelerini sıfırlayarak başlar (1. Satır). Temel hesaplama işlemi, C ile gösterilen kapalı sık kalem kümelerini geri döndüren CHARM-EXTEND yönteminde yapılır.

CHARM-EXTEND, her dalın uygulanabilirliği için test edilmesinden sorumludur. Bu yöntem, o anda geçerli olan düğüm kümesindeki her kalem_kümesi-tid_kümesi çiftini çıkarır ($X_i \times t(X_i)$, satır 3) ve toplam f sırasına göre ondan sonra gelen diğer çiftlerle ($X_i \times t(X_i)$, satır 5) onu birleştirir. İki kalem_kümesi-tid_liste çiftinin birleşimi 6. satırda hesaplanır. CHARM-PROPERTY yöntemi, sonuç kümesini gerekli destek değeri için test eder ve ayrıca yukarıda bahsedilen dört özelliği de uygular. Bu yöntem o anda geçerli olan düğüm kümesini, diğer çiftlerde çoktan varolan kalem_kümesi-tid_liste çiftlerini silerek değiştirebilir. Ayrıca yeni düğümler kümesindeki yeni üretilen sık çocuk çiftleri de ekler. Eğer bu küme boş değilse o yinelemeli olarak derinlik-öncelikli biçimde işleme alınır (Satır 8). O zaman X_i 'nin muhtemelen genişletilmiş X kalem kümesini kapalı kalem kümesinde daha fazla

işlenmeyeceği için eklenir. Bu aşamada X_i içeren herhangi bir kapalı kalem kümesi çoktan oluşturulmuştur. Daha sonra bir sonraki kısaltılmamış dalı işlemek için 3. satıra geri dönülür.

CHARM-PROPERTY yöntemi yeni bir çiftin sık olup olmadığını basitçe test eder, değilse onu eler. Daha sonra kalem_kümesi-tid_liste çiftlerinin dört temel özelliğinin herbirini, var olan kalem kümelerini genişleterek ve bazı sınıflandırılan dalları o anda geçerli olan düğümlerin kümesinden silerek veya yeni çiftleri düğüm kümesine bir sonraki adım için ekleyerek test eder.

Çizelge 4.6 CHARM Algoritması

CHARM ($\delta \subseteq I \times T$, minsup):

1. Nodes = $\{I_j \times t(I_j) : I_j \in I \wedge |t(I_j)| \geq \text{minsup}\}$
2. CHARM-EXTEND (Nodes,C)

CHARM-EXTEND (Nodes,C)

3. for each $X_i \times t(X_i)$ in Nodes
4. New N = $\emptyset$ and $X = X_i$
5. for each $X_j \times t(X_j)$ in Nodes, with $f(j) > f(i)$
6. $X = X \cup X_j$ and $Y = t(X_i) \cap t(X_j)$
7. CHARM-PROPERTY (Nodes,NewN)
8. if NewN $\neq \emptyset$ then CHARM-EXTEND (NewN)
9. $C = C \cup X$ //if X is not subsumed
10. if ($|Y| \geq \text{minsup}$) then
11. if $t(X_i) = t(X_j)$ then //Property 1
12. Remove X_j from Nodes
13. Replace all X_i with X
14. else if $t(X_i) \subset t(X_j)$ then //Property 2
15. Replace all X_i with X
16. else if $t(X_i) \supset t(X_j)$ then //Property 3
17. Remove X_j from Nodes
18. Add $X \times Y$ to NewN
19. else if $t(X_i) \neq t(X_j)$ then //Property 4
20. Add $X \times Y$ to NewN

4.2.3 Eclat

Zaki et al. tarafından 1997'de geliştirilen, sık kalem kümelerini (ayrıca kapalı sık kalem kümeleri ve en büyük sık kalem kümeleri de dahil) bulmayı sağlayan bir algoritmadır. Alt küme kafesinde derinlik-öncelikli arama yapar ve işlem dizinlerini kesiştirerek kalem kümelerinin destek değerini belirler. Algoritma çizelge 4.7'de verilmiştir [3].

Çizelge 4.7 Eclat Algoritması

```

Input:  $D, \sigma, I$  Output:  $F[I](D, \sigma)$ 
1:  $F[I] := \{\}$ 
2: for all  $i \in I$  occurring in  $D$  do
3:    $F[I] := F[I] \cup \{I \cup \{i\}\}$ 
4:   // Create  $D_i$ 
5:    $D_i := \{\}$ 
6:   for all  $j \in I$  occurring in  $D$  such that  $j > i$  do
7:      $C := \text{cover}(\{i\}) \cap \text{cover}(\{j\})$ 
8:     if  $|C| \geq \sigma$  then
9:        $D_i := D_i \cup \{j, C\}$ 
10:    end if
11:  end for
12: // Depth-first recursion
13: Compute  $F[I \cup \{i\}](D_i, \sigma)$ 
14:  $F[I] := F[I] \cup F[I \cup \{i\}]$ 
15: end for

```

Bir aday kalem kümesi, destek değeri algoritmanın 6. satırında hesaplanan her $I \cup \{i, j\}$ kümesi ile gösterilir. Algoritmanın ürettiği aday kalem kümelerinin sayısı diğer derinlik-öncelikli algoritmalara göre çok daha fazladır. Karşılaştırma yapacak olursak Eclat, aslında aday kalem kümelerini sadece Apriori'nin birleştirme adımını kullanarak oluşturur. Veri tabanındaki tüm kalemler, oluşturulan aday kalem kümelerinin sayısını azaltmak için destek değerinin azalması yönünde yeniden sıralanabilir ve bu nedenle sayılması gereken kesişimlerin sayısı ve oluşturulan tüm kalem kümelerinin toplam boyutu azalır. Algoritmanın 10. ve 11. satırları arasında bu yeniden sıralama işlemi her yinelemeli adım için yapılabilir. Apriori ile karşılaştırılacak olunursa, tüm kalem kümelerinin destek değerlerinin sayılması çok daha etkili olarak yapılır.

Hipp et al tarafından optimize edilmiş diğer bir yaklaşım da Apriori ve Eclat'ı birleştirip tek bir Hybrid'e dönüştüren algoritmadır. Algoritma önce Apriori kullanarak sık kalem

kümelerini oluşturmaya genişlik-öncelikli bir yaklaşımla başlar, sonra Eclat kullanarak derinlik-öncelikli bir iterasyona geçiş yapar. Tam geçiş noktası kullanıcı tarafından belirlenmelidir. Bu stratejinin gerçek performans iyileştirmesi tüm aday 2-kalem kümelerinin üretilmesi esnasında ortaya çıkar. Gerçekten, veri tabanındaki bir çok kalem sık olduğu zaman, Eclat her muhtemel 2-kalem kümesini veri tabanında bulursa da bulunmasa da oluşturur. Diğer taraftan eğer işlem veri tabanı çok sayıda sık kalem kümesi işlemi içerirse, Apriori'nin tüm 2'li alt kümelerini oluşturmaya gerek duyması gibi, Eclat gene de Apriori'yi geçer. Elbette, hala aday kalem kümeleri içeren işlemlerin sayısı ana belleğe yüklemek için çok büyük olduğu sürece, Apriori iyi bir şekilde ilerlemeye devam ederken Eclat'a geçiş yapmak imkansız olabilir (Bart Goethals, 2003).

4.2.4 Closet

İlişkisel kural madenciliği sıkça, çok büyük miktardaki sık kalem kümelerini ve ilişkisel kuralları elde etmeyi sağlar. Son çalışmalar ilginç bir alternatif ortaya koymaktadır; “kapalı sık kalem kümelerini ve onların uygun kurallarını bulma”. Bu kurallar da ilişkisel madencilikle aynı güce sahiptir fakat sunulacak kuralların sayısını önemli ölçüde azaltır.

Sık örüntü madenciliğinin, veri madenciliğinin pek çok önemli alanında önemli bir rolü olduğu anlaşılmıştır. Örneğin ilişkiler, sıralı örüntüler, bölümler (episodes) gibi. Ama sık örüntü madenciliğinin sıkça çok büyük miktarda sık kalem kümesi ve kurallar oluşturduğu da bilinmektedir. Bu durum performansı ve madenciliğin etkililiğini azaltır.

Son yıllarda Pasquier Et Al tarafından önerilen ilginç bir alternatif vardır: “Sık kalem kümelerinin tümünü ve onların kurallarını madenlemek yerine, ilişkisel madencilik sadece kapalı sık kalem kümelerini ve onların uygun kurallarını bulmaya gerek duyar”. Bundan önemli bir çıkarım şudur: Kapalı sık kalem kümelerinin madenciliği tüm sık kalem kümelerinin madenciliğiyle aynı güçtedir; fakat üretilecek olan gereksiz kuralları önemli ölçüde azaltır ve hem verimliliği hem de madenciliğin etkililiğini artırır.

Örneğin; bir veri tabanı yalnızca iki işlem içersin “ $\{(a_1, a_2, \dots, a_{100}), (a_1, a_2, \dots, a_{50})\}$ ” ve minimum destek 1 ve minimum güven de %50 olsun. Geleneksel ilişkisel madencilik $2^{100}-1$ yaklaşık 10^{30} sık kalem kümesi üretir. Bunlar $(a_1), \dots, (a_{100}), (a_1, a_2), \dots, (a_{99}, a_{100}), \dots, (a_1, a_2, \dots, a_{100})$, ve çok büyük sayıda ilişkisel kurallardır. Oysa bir sık kalem kümesi madenciliği yalnızca iki kapalı sık kalem kümesi $\{(a_1, a_2, \dots, a_{50}), (a_1, a_2, \dots, a_{100})\}$ ve bir ilişkisel kural “ $((a_1, a_2, \dots, a_{50}) \rightarrow (a_1, a_2, \dots, a_{100}))$ ” üretir. Çünkü diğerlerinin tümü bu bir taneden kolayca elde edilebilir.

Pasquier Et Al, A-close adlı Apriori temelli bir madencilik algoritması önerirken, Zaki ve Hsiao diğer bir madencilik algoritmasını önermişlerdir. Charm adı verilen bu algoritma parça temelli bir veri yapısını kullanarak madenciliğin etkililiğini arttırmaya çalışır. Araştırmalar sonucunda A-close ve Charm'ın her ikisinin de uzun örüntülerin madenciliğinde veya büyük veritabanlarında düşük minimum destek eşiği kullanılması durumlarında maliyetli olduğu anlaşılmıştır. Ancak yapılan araştırmalarda Closet algoritmasının büyük veritabanlarında etkili ve ölçeklenebilir olduğu ortaya çıkmıştır.

Tüm sık kalem kümelerinin madenciliğinde iki problem vardır: Bunlardan ilki; bir işlem veri tabanında çok büyük sayıda sık kalem kümesi olabilir (özellikle destek değeri küçük olduğunda). İkincisi ise çok büyük miktarda ilişkisel kural olabilir. Bu kadar büyük miktarda kuralın kullanıcılar tarafından anlaşılıp işlenmesi çok zor olabilir.

Bu problem için ilginç bir alternatif, kapalı sık kalem kümelerinin ve onların uygun ilişkisel kurallarının madenciliğidir (J.Pei,J.Han,R.Mao,2000).

4.2.4.1 Kapalı Sık Kalem Kümesi

Bir x kalem kümesi aşağıdaki gibi tanımlı bir x' kalem kümesi yoksa kapalıdır:

- $x' \setminus x$ in uygun bir süper kümesidir.
- x i içeren her işlem aynı zamanda x' nü de kapsar.

Eğer kapalı bir x kalem kümesinin desteği verilen destek eşik değerini geçerse sıktır.

Örnek 4.2: Çizelge 4.8'de minimum destek = 2 olan bir veri tabanı örneği verilmiştir.

Çizelge 4.8 Örnek 4.2'ye ait örnek veri tabanı

İşlem ID	İşlemdeki Kalemler
10	A,c
20	a, b, d
30	a, b, c, d
40	A, d

Sık kalem kümeleri: $\{(a), (b), (c), (d), (a,b), (a,c), (a,d), (b,d), (a,b,d)\}$

Kapalı sık kalem kümeleri: $\{(a), (a,c), (a,d), (b,d), (a,b,d)\}$ (J.Pei,J.Han,R.Mao,2000)

4.2.4.2 Kapalı Sık Kalem Kümeleri Üzerinde İlişkisel Kural

$x \rightarrow y$ kuralının kapalı sık kalem kümesi üzerinde ilişkisel kural olabilmesi için:

- Her iki x ve $x \cup y$ kapalı sık kalem kümesi olmalıdır.
- $x \subset z \subset (x \cup y)$ şartını sağlayan bir z kapalı sık kalem kümesi yoktur.
- Kuralın güveni, verilen güven eşik değerini geçer.

İlişkisel kural medenciliğine benzer olarak kapalı sık kalem kümeleri üzerindeki tüm ilişkisel kuralların kümesi iki adımda elde edilebilir:

- Kapalı sık kalem kümelerininin kümesini minimum destek değeriyle madenlemek.
- Minimum güven değeriyle kapalı sık kalem kümeleri üzerinde tüm ilişkisel kuralların kümesini oluşturmak.

Örnek 4.3: Bir işlem veri tabanı-TDB Çizelge 4.9'da verilmiştir: Burada minimum destek = 2 ve minimum güven = %50 olarak kabul edilmiştir.

Çizelge 4.9. Örnek 4.3 için işlem Veri tabanı TDB (J.Pei,J.Han,R.Mao,2000)

İşlem ID	İşlemdeki Kalemler
10	a ,c , d, e, f
20	A, b, e
30	C, e, f
40	a, c, d, f
50	C, e, f

TDB'de toplam yirmi sık kalem kümesi bulunmaktadır. Bunların yalnızca altısı kapalıdır: "acdf, cef, ae, cf, a, e". Ne ac ne de d kapalıdır, çünkü onları içeren her işlem aynı zamanda f'yi de içerir.

İlişkisel Kuralların Bulunması: Örneğin acdf sık kalem kümesi için cf, desteği 4 olan bir alt kümedir. Bu nedenle $R: cf \rightarrow ad$ kuralının güveni $\text{destek}(acdf) / \text{destek}(cf) = 2 / 4 \geq \%50$. Böylece R ilişkisel bir kuraldır. acdf sık kalem kümesi için toplam 14 ilişkisel kural üretilebilir: Her $x \subset \{a,c,d,f\}$ ($x \neq \emptyset$) çıkarımından $\{a,c,d,f\} \rightarrow x$ ilişkisel bir kuraldır.

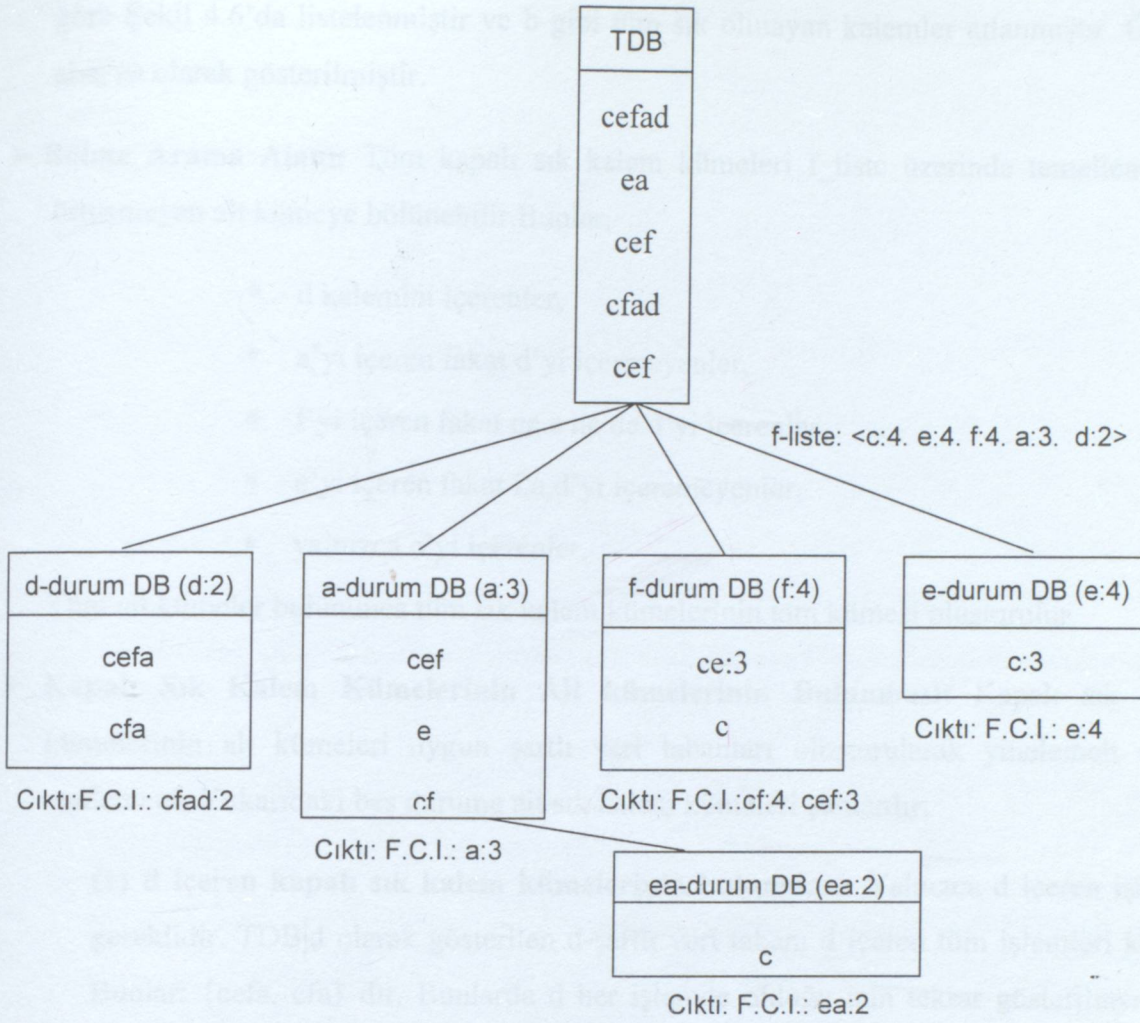
$cf \rightarrow ad$, $a \rightarrow cdf$ kuralları hariç acdf'den üretilen tüm ilişkisel kurallar %100 güven değerine sahiptir. Örneğin $c \rightarrow adf$ %100 güvene sahiptir, çünkü c ve f her zaman birlikte oluşur. Bu

kapalı sık kalem kümeleri üzerinde ilginç görülen, sadece ilişkisel kurallardır. Ayrıca böyle ilginç ilişkisel kuralların güven değerlerini hesaplamak için yalnızca kapalı sık kalem kümelerinin destek değerlerine gerek vardır. Örneğin $cf \rightarrow ad$ ve $a \rightarrow cdf$ elde etmek için sadece $acdf$, cf ve a 'nın destek değerlerini bilmemize gerek vardır.

Şimdi kapalı sık kalem kümeleri üzerindeki tüm ilişkisel kuralları elde edelim. Her kural $x \rightarrow y$ (destek, güven) biçiminde görünür. Burada x ve y kalemlerdir. Kapalı sık kalem kümesi, $acdf$ kapalı sık kalem kümesi olarak yalnızca 2 alt kümeye sahiptir. Bunlar a ve cf 'dir. Buradan da iki ilişkisel kural elde edilir: $cf \rightarrow ad$ (2, %50) ve $a \rightarrow cdf$ (2,%67). Diğer dördü şunlardır : $e \rightarrow cf$ (3, %75), $cf \rightarrow e$ (3, %75), $e \rightarrow a$ (2, %50), $a \rightarrow e$ (2, %67).

Kapalı sık kalem kümelerinin tümünün büyük veritabanlarından bulunmasına kapalı sık kalem kümelerinin bulunması problemi denir. Kapalı sık kalem kümelerinin madencilikteki tekniklerden önce kapalı sık kalem kümelerinin bir özelliği anlatılacaktır:

Olgu: X ve Y iki kalem kümesi ve $destek(X) = destek(Y)$ olsun. Eğer $Y \subset X$ ise Y kapalı bir kalem kümesi olmaz.



Şekil 4.6 Closet kullanılarak kapalı sık kalem kümelerini madencilme (F.C.I olarak kısaltılır) (J.Pei,J.Han,R.Mao,2000)

4.2.4.3 Kapalı Sık Kalem Kümelerinin Madenciliği

Kapalı sık kalem kümelerinin madenlenmesi işlemi örnek 4.4 üzerinde anlatılacaktır:

Örnek 4.4: Bu örnekte çizelge 4.9’da verilen ve minimum_destek = 2 olan TDB işlem veri tabanı için kapalı sık kalem kümelerinin medenciliğinde kullanılan parçala ve al metodu anlatılacaktır.Bu metodun adımları sırasıyla şöyledir:

- **Sık Kalem Kümelerinin Bulunması:** Sık kalem kümelerini bulmak ve f_liste olarak adlandırılan ($f_liste = \langle c: 4, e: 4, f: 4, a:3, d: 2 \rangle$; burada kelimler destek miktarlarının azalması yönünde sıralanmıştır ve “:” sonrasında gelen rakamlar kelimelerin destek değerlerini göstermektedir) genel sık kalem kümelerinin listesini elde etmek için TDB taranır. Daha kolay anlaşılması için her işlemdeki sık kalem kümeleri f_liste’deki sıraya

göre Şekil 4.6'da listelenmiştir ve b gibi tüm sık olmayan kalem atlanmıştır. Örneğin abe, ea olarak gösterilmiştir.

➤ **Bölme Arama Alanı:** Tüm kapalı sık kalem kümeleri f_liste üzerinde temellenen beş örtüşmeyen alt kümeye bölünebilir. Bunlar;

- d kalemni içerenler,
- a'yı içeren fakat d'yi içermeyenler,
- f'yi içeren fakat ne a ne de d'yi içerenler,
- e'yi içeren fakat f,a,d'yi içermeyenler,
- yalnızca c'yi içerenler.

Tüm alt kümeler bulununca tüm sık kalem kümelerinin tam kümesi oluşturulur.

➤ **Kapalı Sık Kalem Kümelerinin Alt kümelerinin Bulunması:** Kapalı sık kalem kümelerinin alt kümeleri uygun şartlı veri tabanları oluşturularak yinelemeli olarak madenlenir. Yukarıdaki beş duruma ait sık kalem kümeleri şunlardır;

(1) d içeren kapalı sık kalem kümelerinin bulunması: Yalnızca d içeren işlemler gereklidir. TDB|d olarak gösterilen d-şartlı veri tabanı d içeren tüm işlemleri kapsar. Bunlar: {cefa, cfa} dır. Bunlarda d her işlemde olduğu için tekrar gösterilmemiştir. d'nin desteği 2'dir. c,f,a kalemleri TDB|d'de iki kez tekrarlanmıştır. Bu d'yi içeren her işlemin aynı zamanda c, f, a'yı da içerdiği anlamına gelir. Ayrıca e, TDB|d'de yalnızca bir kere tekrarlandığı için sık değildir. Bu nedenle cfad : 2 kapalı sık kalem kümesidir. Bu kalem kümesi TDB|d'deki tüm sık kalemlerini içerdiği için TDB|d'nin madenlenmesi işlemi sona erer.

(2) a'yı içeren fakat d'yi içermeyen kapalı sık kalem kümelerinin bulunması: Benzer olarak a-şartlı veri tabanı TDB|a = {cef, e, cf} dir. d'yi içeren tüm kapalı sık kalem kümeleri TDB|d' de bulunduğu için her işlemdeki d ihmal edilir. Destek(a) = 3 olduğu için ve a-şartlı veri tabanındaki her işlemde olan hiçbir kalem olmadığı için sadece a:3 bir sık kalem kümesidir.

a'yı içeren fakat d'yi içermeyen, geri kalan kapalı sık kalem kümelerini bulmak için öteki a-şartlı veritabanlarını tasarlamamız gerekir. Önce a-şartlı veri tabanındaki sık kalemlerin kümesi yerel bir sık kalem listesi olan $f_liste_a = \langle c :2, e:2, f:2 \rangle^3$ yı oluşturur. Yerel sık olmayan kalem kümeleri global f_liste'de olsalar bile önemsenmezler.

f_liste_a 'ya göre a 'yı içerip d 'yi içermeyen kapalı sık kalem kümeleri üç alt kümeye bölünebilir:

- a ve f 'yi içerip d 'yi içermeyenler,
- ae 'yi içerip d 'yi içermeyenler,
- ac 'yi içerip d , e veya f 'yi içermeyenler.

Bunlar, şartlı veritabanlarını oluşturarak yinelemeli olarak madenlenebilir.

fa 'nın destek değeri, fa 'nın ve daha önceden bulunan bir kapalı sık kalem kümesinin de süper kümesi olan $cfad$ 'inkine eşittir. Bu şunu ifade eder: “ fa 'yı içeren her işlem $cfad$ 'ı da içermelidir”. Bu nedenle fa 'yı içerip d 'yi içermeyen hiç bir kapalı sık kalem kümesi yoktur. Benzer olarak ca , $cfad$ 'ın bir alt kümesi olduğu için ve destek $(ca) = \text{destek}(cfad)$ olduğundan ca 'yı içerip d , e veya f 'yi içermeyen kapalı sık kalem kümesi yoktur.

ea -şartlı veri tabanı, $TDB|_e = \{c\}$ hiçbir sık kalem kümesi oluşturamaz. Bu nedenle $ea : 2$ kapalı bir sık kalem kümesi olabilir.

(3) f 'yi içerip a ve d 'yi içermeyen kapalı sık kalem kümelerinin bulunması: f -şartlı veri tabanı, $TDB|_f = \{ce : 3, c\}$ olup burada $ce : 3$ ifadesi ce 'nin üç kere tekrarlandığını gösterir. c , f -şartlı veri tabanındaki her işlemde olduğu için ve cf aynı destek değeriyle herhangi bir kapalı sık kalem kümesinin alt kümesi olmadığı için $cf : 4$ kapalı bir sık kalem kümesidir. fc 'nin destek değeri de f ve c 'ninkine eşit olduğundan (f ve c her zaman birlikte oluşur) c 'yi içerip d f 'yi içermeyen hiçbir kapalı sık kalem kümesi yoktur. Ayrıca $cef : 3$, bulunan herhangi bir kalem kümesinin bir alt kümesi değildir, bu nedenle kapalı bir sık kalem kümesidir.

(4) e 'yi içerip f , a ve d 'yi içermeyen kapalı sık kalem kümelerinin bulunması: Benzer olarak e -şartlı veri tabanı $TDB|_e = \{c : 3\}$ dır. Fakat ce kapalı bir sık kalem kümesi değildir. Çünkü ce , cef 'in bir alt kümesidir ve $\text{destek}(ce) = \text{destek}(cef)$ 'dir. Ama $e : 4$ kapalı bir sık kalem kümesidir.

(5) Yalnızca c 'yi içeren kapalı sık kalem kümelerinin bulunması: c adımımda c 'yi içerip f içermeyen kapalı sık kalem kümelerinin olmadığını biliyoruz. Bu nedenle sadece c 'yi içeren kapalı sık kalem kümesi yoktur.

- **Özetle Bulunan Kapalı Sık Kalem Kümelerinin Kümesi:** $\{acdf : 2, a : 3, ae : 2, cf : 4, cef : 3, e : 4\}$ 'dir (J.Peı,J.Han,R.Mao,2000).

4.2.4.4 CLOSET Algoritması

Girdi: TDB işlem veri tabanı ve destek eşik değeri mindestek.

Çıktı: Kapalı sık kalem kümelerinin tam kümesi.

Başlangıç: Kapalı sık kalem kümelerinin kümesi FCI olsun. $FCI = \emptyset$ olarak başlatılır.

- **Sık kalem kümelerinin bulunması:** TDB işlem veri tabanı taranır, sık kalemelerin listesi olan f_liste hesaplanır.
- Kapalı sık kalem kümelerinin yinelemeli olarak madenlenmesi: CLOSET ($\emptyset$, TDB, f_liste , FCI) çağırılır.

Altprogram CLOSET (X, DB, f_liste , FCI)

Parametreler:

1. X: DB'nin bir X-şartlı veri tabanı olması durumunda sık kalem kümesine veya DB'nin TDB olması durumunda $\emptyset$ 'ye karşılık gelir.
2. DB: Şartlı veri tabanının işlem veri tabanı
3. f_list : DB'nin sık kalem listesi
4. FCI: Önceden bulunan kapalı sık kalem kümelerinin kümesi.

Metod:

1. Y, DB'nin her işleminde görülen $f_listesi$ 'ndeki kalemelerin kümesi olsun. $X \cup Y$ aynı destek değeriyle, FCI'daki bazı kalem kümelerinin uygun bir alt kümesi değilse FCI'ya eklenir.
2. DB için FP-tree yapısı oluşturulur. Önceden seçilen kelimeler hariç tutulabilir.
3. Mümkün olan kapalı sık kalem kümeleri seçilir.
4. $f_listesi$ 'nde kalan her kalem için şartlı veri tabanı düzenlenir, aynı zamanda bu şartlı veritabanları için yerel sık kalem listeleri hesaplanır.
5. $f_listesi$ 'nde kalan her i kalemi için, sonuncusundan başlayarak CLOSET (iX , $DB|i$, $f_listesi$, FCI) çağırılır (eğer iX , aynı destek sayısı ile önceden bulunan kapalı sık kalem kümelerinin bir alt kümesi değilse). Burada $DB|i$; DB ve uygun sık kalem listesi olan $f_listesi$ ile ilgisi olan i -şartlı veri tabanıdır (J.Pei,J.Han,R.Mao,2000).

4.2.5 Partitioning

Partition algoritması Savasere et al. tarafından önerilmiş olup tüm bilinen yaklaşımlardan farklı bir algoritmadır. Burada veri tabanı, düşey veri tabanı yapısı kullanılarak ana belleğe yüklenir ve bir kalem kümesinin destek değeri onun iki alt kümesinin kapaklarını (covers) kesiştirerek hesaplanır. Daha özel olarak, her sık kalem için, algoritma onun kapağını kaydeder. Bir k-kalem kümesi adayı I 'nin desteğini hesaplamak için (I , Apriori'deki gibi onun iki alt kümesi olan X ve Y 'nin birleştirilmesiyle oluşturulur) algoritma X ve Y 'nin kapaklarını kesiştirir.

Elbette tüm kalemlerin kapaklarını kaydetmek, tüm veri tabanının ana belleğe yüklenmesi anlamına gelir. Büyük veritabanları için bu imkansız olabilir. Bu nedenle Partition algoritması şu yolu dener: "Veri tabanı bir çok ayrık parçaya bölünür ve algoritma her parça için onda sık olan tüm kalem kümelerini oluşturur. Veri tabanının parçaları, her biri ana belleğe kendisi sığacak şekilde seçilir."

Algoritma her parçanın tüm sık kalem kümelerini birleştirir. Bu, bütün veri tabanının üzerinden elde edilen tüm sık kalem kümelerinin bir süper kümesinde sonlanır, ama bütün veri tabanında sık olan bir kalem kümesi bu parçalardan birinde sık olmak zorundadır. Ozaman tüm kalem kümelerinin gerçek destek değerleri, veri tabanının ikinci bir kez taranması esnasında hesaplanır. Tekrar, her parça düşey veri tabanı yapısını kullanarak ana belleğe okunur ve her kalem kümesinin destek değeri, bu kalem kümesinde bulunan tüm kalemlerin kapaklarını kesiştirerek hesaplanır. Partition algoritması aşağıda verilmiştir.

Çizelge 4.10 Partition – Yerel Kalem kümesi Madenleme

Input: D, σ Output: $F(D, \sigma)$

```

1: Compute  $F_1$  and store with every frequent item its cover
2:  $k := 2$ 
3: while  $F_{k-1} \neq \{\}$  do
4:    $F_k := \{\}$ 
5:   for all  $X, Y \in F_{k-1}, X[i] = Y[i]$  for  $1 \leq i \leq k-2$ , and  $X[k-1] < Y[k-1]$  do
6:      $I = \{X[1], \dots, X[k-1], Y[k-1]\}$ 
7:     if  $\forall J \subset I : J \in F_{k-1}$  then
8:        $I.\text{cover} := X.\text{cover} \cap Y.\text{cover}$ 
9:       if  $|I.\text{cover}| \geq \sigma$  then
10:         $F_k := F_k \cup I$ 
11:       end if
12:     end if
13:   end for
14:    $k++$ 
15: end while

```

Çizelge 4.11 Partition

Input: D, σ Output: $F(D, \sigma)$

```

1: Partition  $D$  in  $D_1, \dots, D_n$ 
2: // Find all local frequent itemsets
3: for  $1 \leq p \leq n$  do
4:   Compute  $C_p := F(D_p, \lceil \sigma_{rel} \cdot |D_p| \rceil)$ 
5: end for
6: // Merge all local frequent itemsets
7:  $C_{global} := \cup_{1 \leq p \leq n} C_p$ 
8: // Compute actual support of all itemsets
9: for  $1 \leq p \leq n$  do
10:  Generate cover of each item in  $D_p$ 
11:  for all  $I \in C_{global}$  do
12:     $I.\text{support} := I.\text{support} + |I[1]|.\text{cover} \cap \dots \cap I[|I|].\text{cover}|$ 
13:  end for
14: end for
15: // Extract all global frequent itemsets
16:  $F := \{I \in C_{global} \mid I.\text{support} \geq \sigma\}$ 

```


Her ne kadar tüm kalemelerin kapakları ana belleğe yüklenebilse de, her parça için tüm yerel sık kalem kümelerinin oluşturulması esnasında, tüm yerel aday k-kalem kümelerinin kapakları ana belleğe yüklenemeyebilir. Ayrıca, algoritma büyük ölçüde veri tabanının heterojenliğine de bağlıdır ve çok sayıda yerel sık kalem kümeleri oluşturarak performansta önemli ölçüde azalmayla sonuçlanabilir. Ama veri tabanının tamamı ana belleğe sığarsa ve herhangi bir iterasyondaki tüm kapakların toplamı ana belleğin sınırlarını aşmazsa o zaman algoritma Apriori'yi önemli ölçüde geçer. Elbette bu, hızlı kesişim tabanlı mekanizmadan dolayıdır.

4.2.6 Paralel İlişkisel Kural Madenciliği Algoritmaları

İlişkisel kuralları paralel olarak bulmak için hesaplama, haberleşme, bellek kullanımı, senkronizasyon ve probleme özgü bilginin kullanımı gibi bazı koordinasyon önlemlerini göz önünde bulundurmamak gerekir. R.Agrawal ve J.C. Shafer tarafından 1996 yılında yayınlanan "Parallel Mining of Association Rules" adlı makalede paylaşımı olmayan çoklu işlemciler üzerinde ilişkisel kuralların madenlenmesi için 3 algoritma önerilmiştir:

- Sayı Dağıtım Algoritması
- Veri Dağıtım Algoritması
- Aday Dağıtım algoritması

Sayı dağıtım algoritması, haberleşmeyi minimuma indirmeyi amaçlar. Bu algoritma "gereksiz hesaplamaların paralelinde buna zıt olarak haberleşmekten sakınan boş işlemcilere" izin veren basit bir prensip kullanır.

Veri Dağıtım algoritması, sistemin toplam ana belleğinin daha etkili bir şekilde kullanılmasını sağlar. Bu algortmada her işlemci birbirini dışlayan adayları sayar. Böylece işlemcilerin sayısı arttırılırsa çok büyük miktarda aday bir geçişte sayılabilir. Bu algortmanın dezavantajı her işlemcinin kendi yerel verisini her geçişte diğer tüm işlemcilere yayınlaması gerektiğidir.

Aday dağıtım algoritması, her iki veri ve adayları her işlemcinin bağımsızca işleyebileceği bir şekilde bölümlere ayırır.

J.S. Park, M.S. Chen ve P.S. Yu tarafından 1995 yılında yayınlanan "Efficient Hash-Based Algorithm for Mining Association Rules" adlı makelede önerilen PDM algoritması DHP algoritmasını paralelleştirmeye çalışır. Her düğüm global olarak sık olan kalem kümelerini, onların destek değerlerini aday kümelerinkiyile değiştirerek sayar. HDA tekniğini uygulamak için tüm düğümler HDA sonuçlarını yayınlamalıdır. Tüm HDA demetleri arasında sadece toplam sayısı bir eşik değerinden büyük olanlar demet sayı değişimi için seçilirler,

böylece tüm demetler yayınlanmak zorunda kalınmaz. Paralel ilişkisel kural madenciliği için iki algoritma daha vardır; biri akıllı veri dağıtım algoritması ve diğeri Hybrid dağıtım algoritmasıdır. Akıllı veri dağıtım algoritması paralel bilgisayarların toplam bellek alanını akıllı aday bölümlenme planı kullanarak etkili bir şekilde kullanır. Ayrıca veriyi işlemciler arasında taşımak için etkili bir haberleşme mekanizması kullanır. Hybrid dağıtım algoritması, iyi yük dengesini sürdürmek için aday kümesini dinamik olarak bölümler. Diğer bir algoritma DMA (İlişkisel kuralların dağıtılmış madenciliği), ilişkisel kuralları dağıtılmış veritabanlarında madenlemek için önerilir. DMA az sayıda bir aday kümesi oluşturur ve her aday kümesi için destek sayımı değişimi için yalnızca $O(n)$ mesaj gerektirir. Burada n dağıtık bir veri tabanındaki alanların sayısıdır. T.Shintani ve M. Kitsuregawa tarafından 1998 yılında yayınlanan “Parallel Mining Algorithms for Generalized Association Rules with Classification Hierarchy” adlı makalede sınıf hiyerarşili ilişkisel kuralları madenlemek için paralel bir algoritma önerilmiştir. Bu algoritmada mümkün bellek alanı, sıkça oluşan aday kalem kümelerini belirleyerek ve onları tüm işlemciler üzerine kopyalayarak tümünden kullanılır. Gerçi sık kalem kümeleri hiçbir haberleşme olmaksızın yerel olarak işlenebilir. Böylece işlemciler arasındaki yük fazlalığını etkili olarak azaltır [4].

4.3 Algoritmaların Deneysel Olarak Karşılaştırılması Örnekleri

Literatürde, ilişkisel kural madenciliği alanında birçok algoritmanın karşılaştırma çalışmaları yapılmıştır. Bu karşılaştırmalar farklı türlerdeki algoritmalar arasında olabildiği gibi Apriori temelli algoritmalar arasında da yapılmıştır. Öncelikle farklı türde algoritmalar için yapılan çalışmalardan örnekler verelim:

Farklı Türde Algoritmalar Arasında Yapılan Karşılaştırma Örnekleri:

Örnek4.5 İlk örnek olarak aşağıda yazarları verilen algoritmalar arasında yapılan karşılaştırma örnekleri verilecektir.

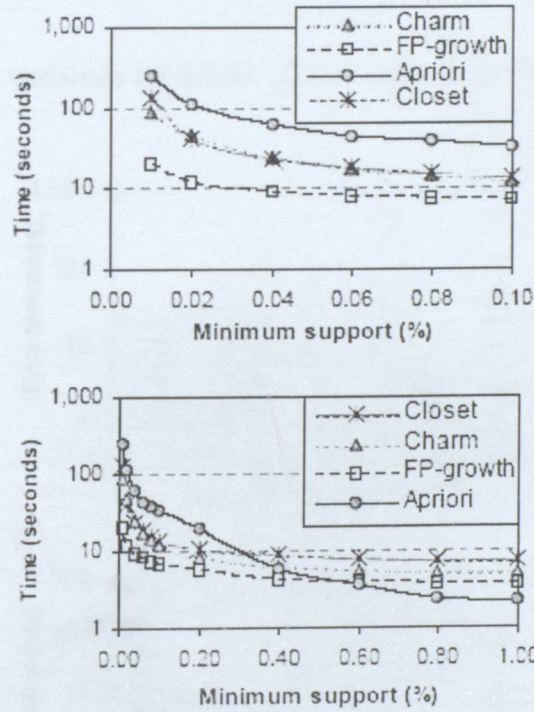
- Apriori: C. Borgelt’un uygulaması
- Charm: M. Zaki
- FP-growth: J. Han’nın araştırma gurubu
- Closet: J. Han’nın araştırma gurubu

Bu karşılaştırma çalışması için çizelge 4.12’de özellikleri anlatılan dört veri kümesi kullanılmıştır:

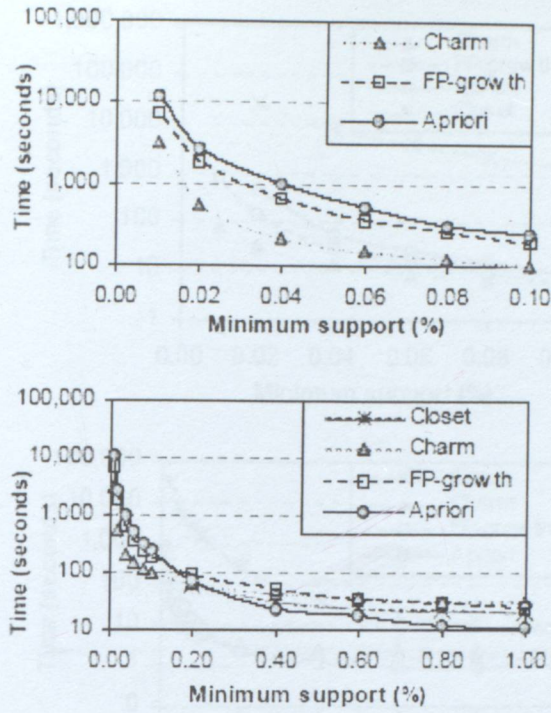
Çizelge 4.12 Örnek 4.5’de verilen algoritmalarının karşılaştırılmasında kullanılan verilerin özellikleri

	İşlem Sayısı	Farklı Ürünler	Maksimum İşlem Boyu	Ortalama İşlem Boyu
BMS-WebView-1	59,602	497	267	2.5
BMS-WebView-2	77,512	3,340	161	5.0
IBM-Artificial	100,000	870	29	10.1
BMS-POS	515,597	1,657	164	6.5

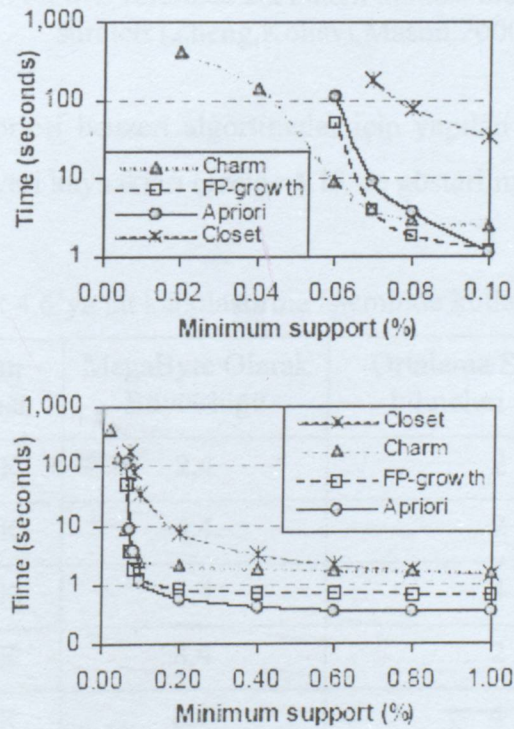
Aşağıdaki grafikler daha önce yapılan karşılaştırma çalışmalarının sonuçlarını göstermektedir:



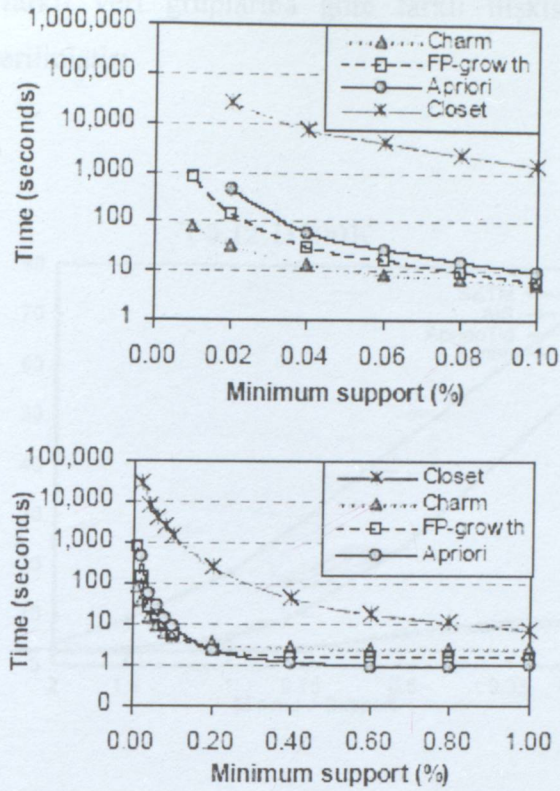
Şekil 4.7 IBM-Artificial verisinde sık kalem kümesi üreten algoritmaların çalışma süreleri



Şekil 4.8 BMS-POS verisinde sık kalem kümesi üreten algoritmaların çalışma süreleri



Şekil 4.9 BMS-WebView-1'de sık kalem kümesi üreten algoritmaların çalışma süreleri



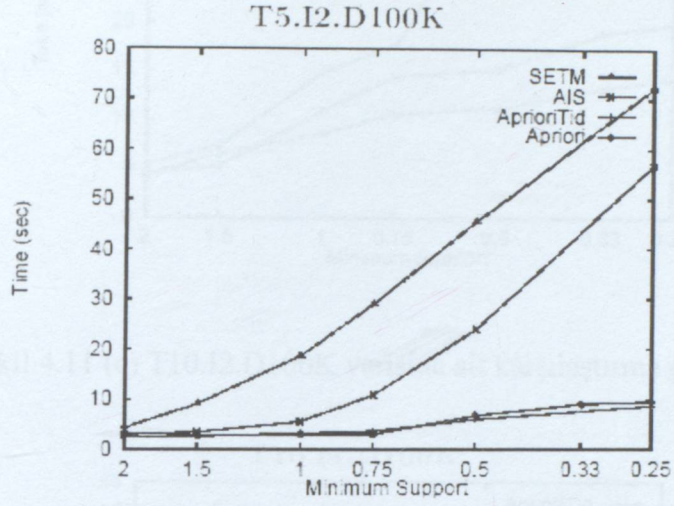
Şekil 4.10 BMS-WebView-2 verisinde sık kalem kümesi üreten algoritmaların çalışma süreleri (Zheng,Kohavi,Mason,2000)

Örnek 4.6: Burada Apriori benzeri algoritmalar için yapılan karşılaştırma örneklerine yer verilecektir. Kullanılan veri kaynakları çizelge 4.13’de gösterilmiştir:

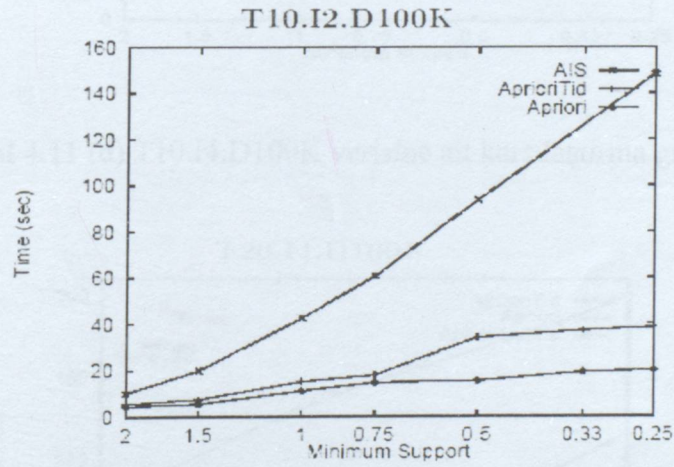
Çizelge 4.13 Örnek 4.6’ya ait karşılaştırma işleminde kullanılan verilerin özellikleri

	İşlem Sayısı	MegaByte Olarak Büyüklüğü	Ortalama Sık kalem kümeleri Boyutu	Ortalama İşlem Boyutu
T5.I2.D100K	100K	2,4	2	5
T10.I2.D100K	100K	4,4	2	10
T10.I4.D100K	100K	4,4	4	10
T20.I2.D100K	100K	8,4	2	20
T20.I4.D100K	100K	8,4	4	20
T20.I6.D100K	100K	8,4	6	20

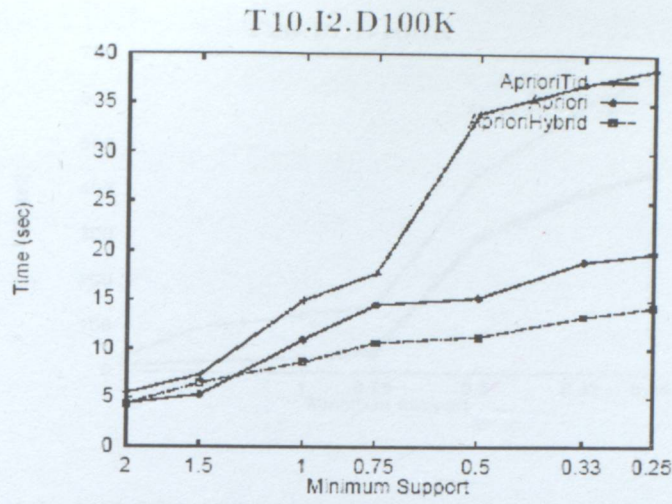
Şekil 4.11'de on bir farklı veri gruplarına göre farklı ilişkisel kural algoritmalarının karşılaştırma örnekleri verilmiştir:



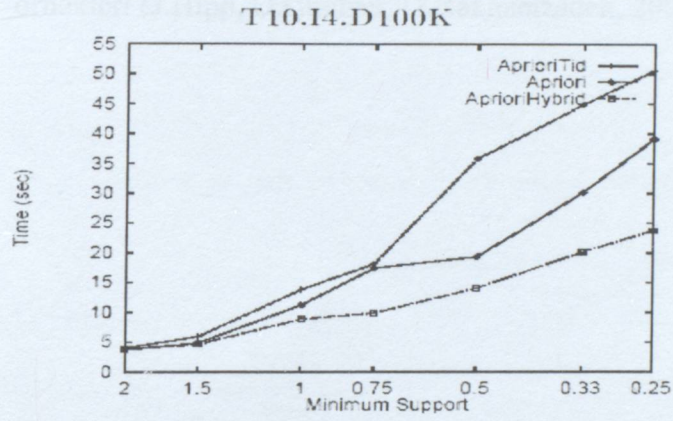
Şekil 4.11 (a) T5.I2.D100K verisine ait karşılaştırma grafiği



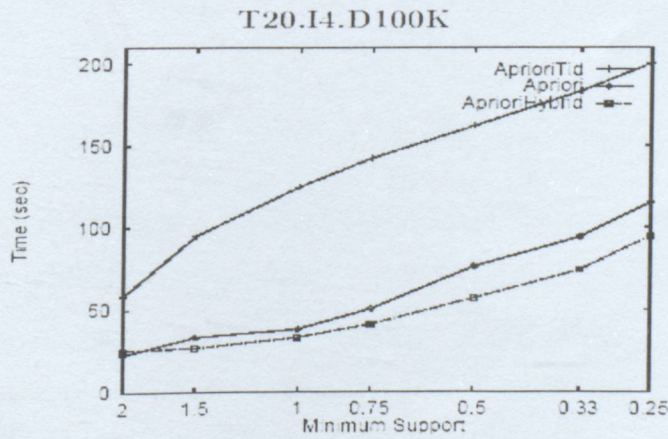
Şekil 4.11 (b) T10.I2.D100K verisine ait karşılaştırma grafiği



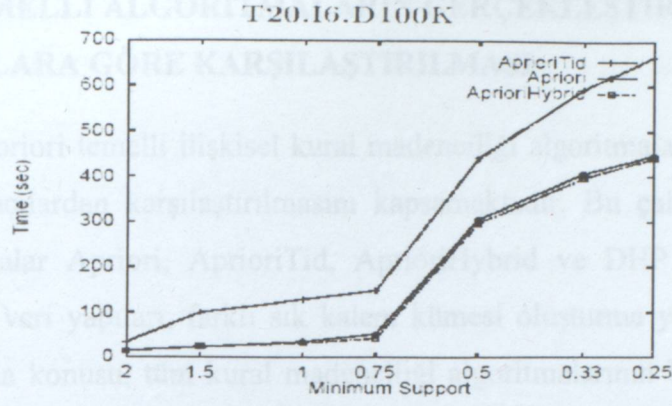
Şekil 4.11 (c) T10.I2.D100K verisine ait karşılaştırma grafiği



Şekil 4.11 (d) T10.I4.D100K verisine ait karşılaştırma grafiği



Şekil 4.11 (e) T20.I4.D100K verisine ait karşılaştırma grafiği



Şekil 4.11 (f) T20.I6.D100K verisine ait karşılaştırma grafiği

Şekil 4.11 Farklı veri gruplarına göre farklı ilişkisel kural algoritmalarının karşılaştırma örnekleri (J.Hipp, U.Güntzer, G.Nakhaeizadeh, 2000)

4.1 Apriori Temelli Algoritmaların Yapısını İnceleme

Çapel bir $A \Rightarrow B$ ilişkisel kuralı için daha önce ayrıntılı olarak anlatılan destek ve güven değerleri hesaplanır. Bunlar:

- destek $s: P(A \cup B)$
- güven $c: P(B|A)$

İfadeleridir. İlişkisel kural madenciliği aşağıdaki iki temel adımdan oluşur:

- Sık kalem kümelerini bulma: Bir sık kalem kümesinin alt küresi de sıkır. Örneğin, $\{AB\}$ sık ise, hem $\{A\}$ hem de $\{B\}$ sık olmalıdır. İteratif olarak sık, sık, sık kümeleri bulunur.
- Sık kalem kümeleri kullanılarak ilişkisel kurallar üretilir.

İlişkisel kural madenciliğinin tüm performansı en önemli olarak ilk adımda belirlenir. Sık kalem kümelerinin belirlenmesinden sonra ilişkisel kurallar kolayca belirlenir. Bu amaçla kullanılan birçok metod vardır. Ancak en temel metod Apriori algoritmasıdır. Apriori'nin türetilen AprioriTID ve AprioriHybrid algoritmaları ise Apriori'ye benzer DFP algoritmasının en performanslı 2. ilki aynısı gerçekleştirilmesine bağlıdır. Bu nedenle bu çalışmada her algoritmanın yukarıdaki ilk adım nasıl uyguladığını incelenip buna göre karşılaştırmaları yapılmıştır. Daha sonra tüm bu algoritmalar tarafından ortak olarak kullanılan sık kalem kümelerinden ilişkisel kural öğrenme algoritması uygulanmıştır.

5. APRIORI TEMELLİ ALGORİTMALARIN GERÇEKLEŞTİRİLEN UYGULAMALARA GÖRE KARŞILAŞTIRILMASI

Bu tez çalışması, Apriori temelli ilişkisel kural madenciliği algoritmalarının uygulanması ve birbirleriyle farklı açılardan karşılaştırılmasını kapsamaktadır. Bu çalışmada incelenen ve kıyaslanan algoritmalar Apriori, AprioriTid, AprioriHybrid ve DHP algoritmalarıdır. Bu algoritmalar benzer veri yapıları, farklı sık kalem kümesi oluşturma yöntemleri kullanırlar. Burada asıl araştırma konusu, tüm kural madenciliği algoritmalarının birbirinden farkı olan cas sayma işlemidir. Çünkü kural üretme prosedürü bütün algoritmalar için ortaktır. Cas; sık kalem kümelerini ifade eder. Örneğin; A'nın farklı semboller kümesi olduğunu düşünelim. A'nın herhangi bir alt kümesi, sembol isimlerinin sözlükteki sırasına göre sıralanmış bir diziyi ifade edebilir. Örneğin {a,c} ve {c,a}, {a,b,c} kümesine ait aynı alt kümeleri ifade eder ve ac dizisi olarak ifade edilir. Bu dizi cas olarak tanımlanır.

5.1 Apriori Temelli Algoritmaların Yapısının İncelenmesi

Genel bir $A \Rightarrow B$ ilişkisel kuralı için daha önce ayrıntılı olarak anlatılan destek ve güven değerleri hesaplanır. Bunlar;

- destek s: $P(A \cup B)$
- güven c: $P(B | A)$

ifadeleridir. İlişkisel kural madenciliği aşağıdaki iki temel adımdan oluşur:

- Sık kalem kümelerini bulma: Bir sık kalem kümesinin alt kümesi de siktir. Örneğin, {AB} sık ise, hem {A} hem de {B} sık olmalıdır. İteratif olarak birli, ikili, üçlü kümeleri bulunur.
- Sık kalem kümeleri kullanılarak ilişkisel kurallar üretme.

İlişkisel kural madenciliğinin tüm performansı asıl olarak ilk adımda belirlenir. Sık kalem kümelerinin belirlenmesinden sonra ilişkisel kurallar kolayca belirlenir. Bu amaçla kullanılan birçok metod vardır. Ama en temel metod Apriori algoritmasıdır. Apriori'nin türleri olan AprioriTID ve AprioriHybrid algoritmaları ile Apriori'ye benzer DHP algoritmasının asıl performansları da ilk adımı gerçekleştirmelerine bağlıdır. Bu nedenle bu çalışmada her algoritmanın yukarıdaki ilk adımı nasıl uyguladıkları incelenip buna göre karşılaştırmaları yapılmıştır. Daha sonra tüm bu algoritmalar tarafından ortak olarak kullanılan, sık kalem kümelerinden ilişkisel kural üretme algoritması uygulanmıştır.

5.1.1 AIS

AIS (Agrawal,Imielinski,Swami,1993), 2^A 'daki cas'ları genişleterek en hızlı şekilde sayar. $L[i]$ i'nci adımda bulunan σ -sık cas'ların kümesi olmak üzere algoritma önce 1-cas'ların kümesi olan $L[1] \subset A$ ile başlar ve ardından $L[q] = \emptyset$ olan bir q adımına kadar

- $L[2] \subset L[1] \times A, \dots, L[i] \subset L[i-1] \times A$

ifadesini hesaplar. Her adımda $L[i]$ 'deki her s cas'ını $L[i-1]$ 'deki bir u cas'ından, u'yü A'daki bir özellik ile genişleterek ($s = u.a$ ve $a > A[i-1]$) elde eder. Elde edilen s bir σ -sık kalem kümesidir. Prosedür tarafından üretilen her u cas'ı bir sayaç ile birlikte üretilir. Bu prosedür sayacın değerini sıfırlayan `init_count(u)`, arttıran `increment_count(u)` ve geri döndüren `get_count(u)` olmak üzere üç fonksiyon kullanır. AIS algoritması Çizelge 5.1'deki gibi çalışır.

Çizelge 5.1 AIS algoritması (Adamo,2000)

```

1  SetofCass L[N]; / N yığın boyutudur.
2.  L[1] = { $\sigma$ -sık 1-cas};
3.  For (int i = 2; L[i-1] != {}; i++) {
4.    SetofCass C = {};
5.    for all transactions (r,s) in D such that |s| >= i
6.      for all cass u in L[i-1]
7.        if (u < s)
8.          for all cass u.a such that a > A[i-1]
9.            if (u.a <= s)
10.             if (u.a is in C)
11.               increment_count (u.a);
12.             else {
13.               C = C  $\cup$  {u.a};
14.               init_count (u.a);
15.             }
16.    L[i] = {};
17.    For all cass u in C {
18.      if (get_count (u) >=  $\sigma$ )
19.        L[i] = NL[i]  $\cup$  {u};
20.    }
21.  }
```

5.1.2 SETM

SETM (Houtsma,Swami,1995) σ -sık cas'ların hesaplanmasında SQL kullanılması amacıyla tasarlanmıştır. Adayları oluşturmak için standart birleştirme işlemini kullanmak üzere, SETM cas oluşturma işlemini sayma işleminden ayırır. Ama, SETM esas olarak AIS'den çok farklı değildir: $L[i]$ 'deki cas'lar $L[i-1]$ 'deki mümkün olan tüm cas'ların yeni eklerle

birleştirilmesiyle elde edilir. Bu nedenle SETM burada detaylı olarak anlatılmamıştır (J.Adamo,2001).

5.1.3 Apriori Algoritması

İlişkisel kural tanımı için en iyi bilinen algoritmalar arasında Apriori yer almaktadır. Bu algoritma her sık kalem kümesinin herhangi bir alt kümesinin yine bir sık kalem kümesi olduğu prensibine dayanır. Tüm Apriori temelli algoritmalar gibi birinci adımda sık kalem kümelerinin belirlenmesi, ikinci adımda da bu sık kalem kümelerinden ilişkisel kurallar üretilmesi işlemlerini yapar.

5.1.3.1 Sık Kalem Kümelerinin Madenlenmesi (Temel Adım)

Aşağıda Apriori algoritmasının temel adımları bulunmaktadır. Algoritmanın ilk adımında sık 1-kalem kümeleri (L_1) belirlenir. Bir sonraki geçişte (k 'nci geçiş) iki evre söz konusudur. Önce apriori-gen fonksiyonu kullanılarak C_k aday kalem kümelerini oluşturmak amacıyla ($k-1$)'nci geçişte sık L_{k-1} kalem kümeleri bulunur. Daha sonra veri tabanı taranır ve C_k 'daki adayların destek değerleri hesaplanır. Bu hesaplamanın hızlı yapılabilmesi için verilen bir t işleminde bulunan C_k adayları etkili bir şekilde belirlenmelidir.

Algoritma aşağıdaki adımları gerçekleştirir:

- Önce L_1 bulunur (L_1 : birli sık kalem kümeleri).
- Buradan C_2 oluşturulur (C_2 : ikili aday kalem kümeleri).
- C_2 'den L_2 ikili sık kalem kümeleri bulunur.
- L_2 'den C_3 üçlü aday kalem kümeleri bulunur.
- C_3 'den L_3 üçlü sık kalem kümeleri....[5]

Apriori önce $L[1]$ 'deki σ -sık 1-cas kümelerini oluşturur ($Cas, \sup(s) \geq \sigma$ şartı sağlandığı zaman σ -sık olarak ifade edilir. Burada σ , kullanıcı tanımlı eşik değeridir). Ardından, her i geçişinde C olarak gösterilen bir HDA ağacındaki tüm i -cas adayları oluşturulup, veri tabanı taranır. Böylece cas adayının destek değeri hesaplanır.

İlk işlemde (generate_candidates prosedürü) $L[i-1]$ 'de bulunan ($i-1$)-cas'ların mümkün olan tüm yollarla birleştirilmesiyle i -cas aday kalem kümeleri elde edilir ve ardından oluşturulan i -cas'lardan tüm ($i-1$)-alt kalem kümeleri $L[i-1]$ 'de bulunanlar C adlı HDA tablosuna yerleştirilir.

İkinci işlemde, veri tabanı taranır ve tüm (r,s) işlemleri için, s 'nin C HDA tablosunda bulunan

tüm alt kalem kümelerinin sayacı arttırılır (procedure count_candidates). Bu işlem C'nin bir HDA tablosu yapısında olması nedeniyle daha etkin olur. C'de oluşturulan tüm u kalem kümeleri bir sayaç (counter) ile yaratılır. Aşağıdaki prosedürlerde sayacı sıfırlayan init_count(u), arttıran increment_count(u) ve geri döndüren get_count(u) olmak üzere üç temel fonksiyon mevcuttur.

Sayma sürecinin ardından algoritma en sonunda C'de σ -sık bulunan tüm kalem kümelerini L[i]'ye yerleştirir. Tüm sayım süreci, bir t adımımda L[t] boş olduğunda tamamlanır.

Çizelge 5.2 Apriori algoritması (Adamo, 2000)

```

1.  SetofCass L[N];    //N yığın boyudur.
2.  L[1]={  $\sigma$ -frequent 1-cass};
3.  for(int i=2;L[i-1]!={};i++) {
4.      CandTablePtr C=generate_candidates(L[i-1]);
5.      for all transactions (r,s) in D such that |s|>=i
6.          count_candidates(C,s,1,i);
7.      L[i]={};
8.      for all cass u in C {
9.          if (get_count(u)>=  $\sigma$ )
10.             L[i]=L[i] U {u};
11.      }
12.  delete C;
13.  }
```

5.1.3.2 Aday Kalem Kümeleri Oluşturma Adımı

Aday kalem kümeleri oluşturma prosedürü çizelge 5.3'de verilmiştir.

Çizelge 5.3 Apriori-aday kalem kümesi oluşturma prosedürü (Adamo, 2000)

```

1.  CandTablePtr generate_candidates(SetofCass L) {
2.      CandTablePtr C=new CandTable[CandTableSize];
3.      let i-1 be the length of cass found in L;
4.      for all pairs (s.a,s.b) in L2 such that a<b {
5.          u=s.a.b;
6.          if (all (i-1)-subcass of u are in L)
7.              insert(C,u,1);
8.      }
9.      return C;
10. }
```


Çizelge 5.3'de geçen `CandTableSize` bir sabittir. Son prosedür, C olarak gösterilen HDA tablosu ağacına cas'ların yinelemeli olarak eklenmesini sağlayan, çizelge 5.4'deki Apriori-aday kalem kümelerini ekleme prosedürüdür.

HDA tablolarının her girişi bir bayrak (flag) ve bir göstergeden (pointer) oluşan iki veri üyesine sahip bir yapı içerir. Bayrak, yaprak (leaf) ve düğüm (node) olmak üzere iki değere sahip olabilir. Bayrak değerine bağlı olarak gösterge de bir yaprak veya bir düğüm (örneğin bir HDA tabloyu gösteren gösterge) göstergesidir.

Her giriş bayrak değeri yaprak ve gösterge değeri de NULL olarak yaratılır. Bir yaprak bir cas listesi ve listenin o andaki boyutu olmak üzere iki veri üyesine sahiptir. Bu da boş liste ve 0 değerleriyle yaratılır. `LeafSize` önceden tanımlanmış bir sabit olup yapraklardaki listelerin olabilecek en büyük boyutunu gösterir.

C olarak gösterilen HDA ağaç yapısına bir s cas'ının eklenmesi işleminde, h olarak gösterilen HDA fonksiyonu kullanılarak s cas'ı birinci derinlikteki HDA tablosundan (root table) başlanarak yinelemeli olarak C'yi oluşturan HDA tablolarda aranır. Örneğin **int h(Attribute a)** fonksiyon çağrısı a sembolüne ait ekleme rankını döndürür. Herhangi bir j döngüsünde, HDA işlemi s'nin j'ninci sembolüne yapılır. **get_attribute(Cas s, int k)** fonksiyonu s'nin k'nıncı sembolünü döndürür. Yinelemeli iniş bir yaprağa ulaşıldığı veya yeni bir yaprağın yaratılabileceği en yakın j derinliğinde sonlanır ve s eklenir. s'nin eklenmesi ona atanan sayacın yaratılıp sıfırlanmasına neden olur. Eğer liste, eklemenin üstüne taşarsa C'de yeni bir (j+1) derinlikli HDA tablosu yaratılır. Uygun girişteki bayrak ve gösterge başlangıçta sıfırlanır. s cas'ı ve aynı zamanda listenin tüm elemanları (j+1)'nci sembollerine göre adreslenerek yaratılan yeni tabloya yeniden eklenirler ve önceki liste silinir.

Çizelge 5.4 Apriori-aday kalem kümelerini ekleme prosedürü (Adamo, 2000)

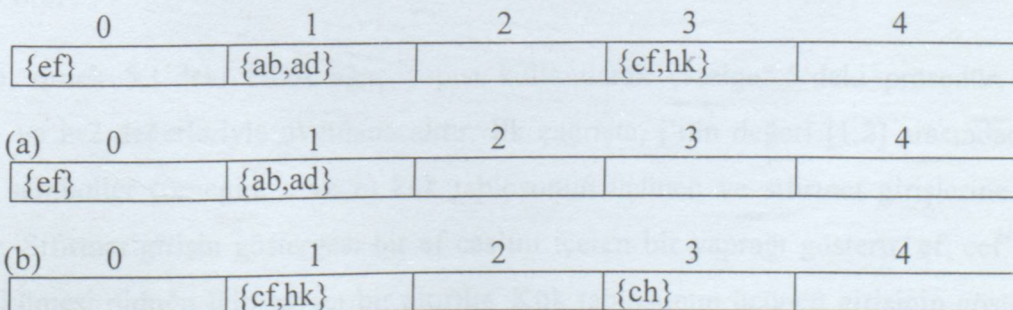
```

1. void insert (CandTablePtr hash_table_ptr, Cas s, int depth) {
2.   int rank=h(get_attribute(s,depth));
3.   if (hash_table_ptr[rank].type==Leaf) {
4.     LeafPtr leaf_ptr=hash_table_ptr[rank].ptr;
5.     if(leaf_ptr==NULL) {
6.       leaf_ptr=new Leaf (LeafSize);
7.       hash_table_ptr[rank].ptr=leaf_ptr;
8.       append s to leaf_ptr->list;
9.     }else {
10.      if (Leaf_ptr->size<LeafSize)
11.        append s to leaf_ptr->list;
12.      else {
13.        if (level==|s|)
14.          handle_insertion_anomaly(leaf_ptr,s);
15.        else {
16.          hash_table_ptr[rank].flag=Node;
17.          hash_table_ptr[rank].ptr=new CandTable[CandTableSize];
18.          for all cass u in leaf_ptr->list
19.            insert (hash_table_ptr[rank].ptr,u,depth+1);
20.          delete Leaf_ptr->list;
21.          insert (hash_table_ptr[rank].ptr,s,depth+1);
22.        }
23.      }
24.    }
25.  }else
26.    insert (hash_table_ptr[rank].ptr,s,depth+1);
27. }

```

Örnek 5.1:

A veri tabanı [a,t] aralığında bir alfabenin elemanlarından oluşmaktadır. CandTableSize ve LeafSize sırasıyla 5 ve 2 olarak verilmiştir. h HDA fonksiyonu “attribute rank mod 5” (attribute rank’ları 1’den başlar) döndürür ve HDA ağacının şu andaki durumunun Şekil 5.1 (a)’deki gibi olduğu varsayalım.



Şekil 5.1 HDA ağacı: (a) şimdiki durum (b) yeni durum (Adamo, 2000)

Yeni ch cas'ı eklendikten sonra, HDA ağacının yeni durumu Şekil 5.1 (b)'deki gibi olur.

Çizelge 5.5 Apriori'nin aday kalem kümesi oluşturma prosedürüyle oluşturulan cas'ların veri tabanından hızlı bir şekilde sayılmasını sağlar. Bu fonksiyon bir önceki fonksiyonun ürettiği C olarak gösterilen HDA ağaç yapısını kullanır.

Çizelge 5.5 Apriori-aday kalemlerini sayma prosedürü (Adamo, 2000)

```

1. void count_candidates (CandTablePtr hash_table_ptr, Cas s, int start, int i) {
2.   for( int j=start; j<=|s|-i+1; j++) {
3.     int rank= h (get_attribute(s,j));
4.     if (hash_table_ptr[rank].type==Leaf) {
5.       LeafPtr leaf_ptr=hash_table_ptr[rank].ptr;
6.       if (leaf_ptr != NULL)
7.         for all cass u in leaf_ptr->list
8.           if (u<=s)
9.             increment_count (u);
10.    }else
11.      count_candidates (hash_table_ptr, s, j+1, i-1);
12.      // (s, j+1) s'nin (j+1)'nci sembolünden başlayan kuyruğunu gösterir.
13.    }
14.  }

```

count_candidates fonksiyonu şu şekilde çalışır: s ; D veri tabanında şu anda incelenmekte olan işlem olmak üzere s cas'ı ve onun tüm alt kalem kümeleri (subcas) C 'yi oluşturan HDA tablolara adreslenir. Bu işlem C 'de birinci derinlikteki tablodan başlanarak yinelemeli olarak yapılır. Yinelemeli inişin herhangi bir seviyesinde HDA işlemi, inişin bir önceki seviyesindeki üzerinde HDA yapılmış olan sembolü takip eden diğer tüm sembollere uygulanır. Her yinelemeli iniş bir yaprağa ulaşıldığında sonlanır. O zaman yapraktaki s 'nin alt kalem kümeleri olan tüm kalem kümelerinin sayısı artırılır (Adamo, 2000).

Örnek 5.2:

Burada, örnek 5.1'deki HDA ağaç yapısı kullanılarak çizelge 5.5'deki prosedür, $s = cef$, $start=1$ ve $i=2$ değerleriyle uygulanacaktır. İlk çağrışta, j 'nin değeri $[1,2]$ arasındadır. Buna uygun semboller (örneğin, c ve e) kök tablosunun üçüncü ve sıfırıncı girişlerine sırasıyla eklenir. Sıfırıncı girişin göstergesi bir ef cas'ını içeren bir yaprağı gösterir. ef , cef 'in bir alt kalem kümesi olduğu için sayacı bir attırılır. Kök tablosunun üçüncü girişinin göstergesi de yeni bir HDA tablosunu gösterir. O zaman count_candidates fonksiyonu $s=cef$, $start=2$ ve $i=1$

değerleri ile yinelemeli olarak çalışır. Şu andaki seviyede j , $[2,3]$ aralığındadır. Buna uygun semboller (örneğin, e ve f), yeni tablonun sırasıyla sıfıncı ve birinci girişlerine adreslenir. Yeni tablonun null gösterge içeren sıfıncı girişi için birşey yapmaya gerek yoktur. Birinci giriş cf ve hk kalem kümelerini içeren bir yaprağa yönlendir. cf , cef 'in bir alt kalem kümesi olduğu için sayacı bir arttırılır. Bu örnek şekil 5.1 üzerinde daha anlaşılır bir şekilde görülebilir.

5.1.3.3 Aday Kalem Kümesi Oluşturma Örnekleri

Aday kalem kümeleri bir önceki adımda belirlenen sık kalem kümelerinden belirlenir. Örneğin üçlü sık kalemelerinden (ör: $L3=\{abc, abd, acd, ace, bcd\}$) 4-lü aday kalemler (ör: $C4=\{abcd\}$) oluşturulması iki adımda gerçekleşir. Bunlar;

- **Kendisiyle birleştirme adımı:** $L3 * L3$. Örneğin, abc ve abd den $abcd$ veya acd ve ace den $acde$ elde edilmesi.
- **Kısaltma adımı:** Örneğin, $acde$ atılır çünkü ade $L3$ ' ün içinde değildir.

Örnek 5.3 [6]:

Çizelge 5.6'daki örnek veri tabanında 9 işlem mevcuttur.

- Minimum işlem destek sayısı = 2
- Minimum_destek = $2 / 9 = \% 22$
- Minimum_güven = $\% 70$

olarak kabul edilmiştir.

Çizelge 5.6 Örnek 5.3'de kullanılan veri tabanı örneği

Kalem	ID'lerinin TID listesi
T100	1 2 5
T200	2 4
T300	2 3
T400	1 2 4
T500	1 3
T600	2 3
T700	1 3
T800	1 2 3 5
T900	1 2 3

Algoritmanın 1. İterasyonu :

C1: kalem kümesi	destek sayısı		L1:kalem-kümesi	destek sayısı
1	6		1	6
2	7		2	7
3	6	→	3	6 →
4	2		4	2
5	2		5	2
C2:L1 * L1, kalem kümesi	destek_sayısı		L2 kalem-kümesi	destek sayısı
1 2	4		1 2	4
1 3	4		1 3	4
1 4	1		1 5	2
1 5	2		2 3	4
2 3	4		2 4	2
2 4	2	→	2 5	2
2 5	2			
3 4	0			
3 5	1			
4 5	0			

L2: sık 2 li kalem kümeler
C2'deki minimum destek şartını sağlayan kalem kümeleri

Yukarıdaki algoritma çalıştırılarak örnek 5.3 için elde edilen sonuçlar çizelge 5.7'de verilmiştir.

Çizelge 5.7 Örnek 5.3'de elde edilen sonuçlar

C3: L2 join L2	[1 2 3], [1 2 5], [1 3 5], [2 3 4], [2 3 5], [2 4 5]
[1 2 3]' un 2'li kalem alt kümeleri	[1 2], [1 3], [2 3]. Bütün 2'li kalem kümeleri L2 nin içindedir ve bu nedenle [1 2 3], L3 ün içinde yer alır.
[1 2 5]'in 2'li kalem alt kümeleri	[1 2], [1 5], [2 5]. Bütün 2'li kalem kümeleri L2 icindedir ve [1 2 5], L3 içinde yer alır.
[1 3 5]'ün 2'li at kümeleri	[1 3], [1 5], [3 5]. Burada [3 5], L2 icinde yer almaz ve [1 25], L3'ün içinde yer alamaz.
[2 3 4]'ün 2'li alt kümeleri	[2 3], [2 4], [3 4].Burada [3 4], L2 icinde yer almaz .Bu nedenle [2 3 4], L3 icinde yer alamaz.
[2 3 5]'ün 2'li alt kümeleri	[2 3], [2 5], [3 5].Burada [3 5], L2 nin içinde yer almaz. Bu nedenle [2 35], L3 e alt olamaz
[2 4 5]'ün 2'li alt kümeleri	[2 4], [2 5], [4 5].Burada [4 5], L2'nin ve [2 4 5], L3'ün içinde yer alamaz.
C3	Kısaltma (prining) işleminin ardından [1 2 3], [1 2 5] olur.
L3 * L3 (join)	L3 * L3 ile C4 oluşturulur.ör: [1 2 3 5]. Bu arada [2 3 5], L3 içinde yoktur

C4

∅ Algoritma sonlanır.

5.1.3.4 Sık Kalem Kümelerinden İlişkisel Kurallar Üretme

Kuvvetli kurallar, minimum destek ve güven şartını sağlayan kurallardır. Minimum destek şartını sağlayan bütün sık kalem kümeleri için, minimum güven şartını da test etmek gerekir.

$$\text{Güven } (A \Rightarrow B) = P(B | A): \frac{\text{destek}(A \cup B)}{\text{destek}(A)} \quad (5.1)$$

Örnek 5.4:

Sık ürün kümesi olan [1 2 5] için boş olmayan alt kümeler [1 2], [1 5], [2 5], [1], [2], [5] dir.

Aday ilişkisel kurallar ise aşağıda listelenmiştir:

- $1 \wedge 2 \Rightarrow 5$ conf: $2/4 = \%50$
- $1 \wedge 5 \Rightarrow 2$ conf: $2/2 = \%100$
- $2 \wedge 5 \Rightarrow 1$ conf: $2/2 = \%100$
- $1 \Rightarrow 2 \wedge 5$ conf: $2/6 = \%33$
- $2 \Rightarrow 1 \wedge 5$ conf: $2/7 = \%29$
- $5 \Rightarrow 1 \wedge 2$ conf: $2/2 = \%100$

Burada minimum destek $\%70$ olarak kabul edilirse ikinci, üçüncü ve son kurallar kuvvetlidir [5].

5.1.4 En İyileleme (Optimizasyon)

Burada Apriori'den türetilen ve farklı yöntemler kullanan AprioriTid ve AprioriHybrid algoritmaları ayrıntılı olarak anlatılacaktır.

5.1.4.1 AprioriTid Algoritması

AprioriTid, Apriori'nin bir türü olup, destek miktarını hesaplamak için tüm veri tabanını ilk kez taradıktan sonra üstünden ikinci kez geçmemeyi amaçlar. Bu amaçla *counting_base* olarak adlandırılan ek bir küme kullanır.

Birinci adımda *counting_base* bir (r, S_1) çiftinden oluşur. r , D veri tabanındaki (r, s) işleminin ilk elemanını ve S_1 de s 'deki tüm özelliklerin kümesini gösterir.

$i > 1$ adımında, *counting_base* bir (r, S_i) çiftinden oluşur. Burada r , D veri tabanındaki (r, s)

işleminin sırasını ve S_i aşağıda gösterildiği gibi s 'den elde edilen bir kümeyi gösterir. S_i aşağıdaki özelliği taşıyan s 'ye ait tüm i -alt kalem kümelerini kapsar.

Özellik: S_i 'deki herhangi bir u , “ $u.ab$ ” biçimindedir. Fakat $u.a$ ve $u.b$, (r, S_{i-1}) işlemindeki S_{i-1} 'e ait olmalıdır ve her iki $u.a$ ve $u.b$ σ -sık $(i-1)$ -cas olmalıdır ($i = 2$ olduğunda $u = \emptyset$ olmaktadır).

Eğer S_i boş olursa o zaman counting_base kümesinde hiç (r, S_i) çifti olmaz. AprioriTid her adımda Apriori'nin yaptığı gibi yeni bir C aday kümesi oluşturur. Sonra destek değerlerini hesaplamak üzere önceki adımda oluşturulan counting_base kümesi tüm D veri tabanının yerine kullanılır. Sayma işlemi esnasında, AprioriTid aynı zamanda o adımla ilişkili counting_base kümesini de oluşturur. Counting_base kümesini kullanmak doğrudur. Çünkü herhangi bir r için S kümesinden i -cas'ının yukardaki özelliği artık sağlamaması durumu hariç, eksilmesi mümkün değildir. Yukardaki özelliği sağlamayan adaylar σ -sık olamayacağından işleme alınmazlar.

Çizelge 5.8 AprioriTid algoritması (Adamo, 2000)

```

1. L[N]; // N yığın boyutudur.
2. SetofCass L[1]={ $\sigma$ -sık 1-cas};
3. Counting_base counting_base[2];
4. int bin=0; // counting_base'in dönüşüm indeksi
5. counting_base[bin] = {(r,S) | (r,s)  $\in$  D ve S, s'deki özelliklerin kümesidir};
6. for (int i = 2; L[i-1] != {}; i++) {
7.   SetofCass C = generate_candidates(L[i-1]);
8.   // C'deki her u, u.ab biçimindedir ve her iki u.a ve u.b L[i-1]'in içindedir.
9.   int previous_bin = bin;
10.  bin = (bin + 1) % 2;
11.  counting_base[bin] = {};
12.  for all cass u in C
13.    init_count(u);
14.  for all (r,S) in counting_base[previous_bin] {
15.    SetofCass S' = {};
16.    for all cass u in C such that u==u.ab and both u.a and u.b are in S {
17.      S' = S'  $\cup$  {u};
18.      increment_count(u);
19.    }
20.    //S' içindeki her u, u.ab biçimindedir ve
21.    // her iki u.a ve u.b eşzamanlı olarak L[i-1] ve S'nin içindedir.
22.    if (S' != {})
23.      counting_base[bin] = counting_base[bin]  $\cup$  {(r,S')};
24.  }
25.  L[i] = {};
26.  for all cass u in C
27.    if (get_count(u)  $\geq \sigma$ )
28.      L[i] = L[i]  $\cup$  {u};
29. }
```


Çizelge 5.9 Aday Oluşturma Prosedürü (Adamo, 2000)

```

1. SetofCass generate_candidates (SetofCass L) {
2.   SetofCass C;
3.   let i-1 be the length of cass found in L;
4.   for all pairs (s.a,s.b) in L2 such that a<b {
5.     u = s.a.b;
6.     if (all (i-1)-subcass of u is in L)
7.       C = C ∪ u;
8.   }
9.   return C;
10. }

```

Algoritmayı daha basit hale getirmek için burada kullanılan aday oluşturma prosedürü, Apriori'ninkine göre daha basit bir yöntem kullanır. Bu prosedür geriye bir HDA ağacı yerine bir küme döndürür.

Örnek 5.5: D veri tabanı aşağıdaki işlemlerin kümesinden oluşmaktadır:

- { (1,acd), (2,bce), (3,abce), (4,be) }

Destek oranı eşik değeri $\sigma = 2 / 4$ verilmiştir. AprioriTid algoritması bu veriye uygulanırsa aşağıdaki kümeler oluşacaktır.

1.Adım

counting_base = { (1,{a,c,d}),
 (2,{b,c,e}),
 (3,{a,b,c,e}),
 (4,{b,e})
 }

L[1] = {a,b,c,e}

2.Adım

counting_base = { (1,{ac}),
 (2,{bc,be,ce}),
 (3,{ab,ac,ae,bc,be,ce}),
 (4,{be})
 }

C = {ab,ac,ae,bc,be,ce}

L[2] = {ac,bc,be,ce}

3. Adım

counting_base = { (2,bce),
 (3,{bce})
 }

C = {bce}

L[3] = {bce}

Şekil 5.2. AprioriTid algoritmasının bir çalışma anındaki görüntüsü (Adamo, 2000)

5.1.4.2 AprioriHybrid Algoritması

Verinin üstünden tüm geçişlerde aynı algoritmaları kullanmak şart değildir. Apriori ve AprioriTid algoritmalarının verinin üstünden farklı geçişlerinde gereken çalışma süreleri farklıdır. Önceki geçişlerde Apriori'nin AprioriTid'e göre daha hızlı olmasına rağmen sonraki geçişlerde de AprioriTid Apriori'yi geçer. Bunun nedeni şudur: Apriori ve AprioriTid aynı aday üretme prosedürünü kullandıkları için aynı kalem kümelerini sayarlar. Sonraki geçişlerde aday kalem kümelerinin sayısı azaldığı halde Apriori veri tabanındaki tüm işlemleri kontrol etmeye devam eder. Diğer taraftan AprioriTid destek sayılarını bulmak için veri tabanını taramak yerine S'yi tarar ve S'nin boyutu veri tabanına göre daha küçüktür.

Bu gözlemler ışığında yeni bir AprioriHybrid algoritması geliştirilmiştir. Bu algoritma önceki geçişlerde Apriori'yi kullanırken S'nin küçüldüğünün ve belleğe sığabileceğinin anlaşılması üzerine AprioriTid'e geçiş yapar. S'nin bir sonraki geçişte belleğe sığıp sığmayacağı şu yöntemden anlaşılır: "O andaki geçişin sonunda S deki adayların sayısı bilinir ve bundan eğer oluşturulmuş olsaydı S'nin boyutunun ne olabileceğini tahmin edilebilirdi". Eğer S' bu geçişte veri tabanına sığabilecek kadar küçükse ve adayların sayısı bu geçişte bir önceki geçişe göre daha azsa AprioriTid'e geçiş yapılır. Kontrol edilmesi gereken bir diğer bir durum da o andaki geçişte üretilen C'nin belleğe sığabildiği halde bir sonraki geçişteki C'nin belleğe sığmaması durumudur.

Apriori'den AprioriTid'e geçiş bir maliyet getirir. Apriori'den AprioriTid'e geçişin verinin üstünden k'nıncı geçişin sonunda olduğu varsayılın. (k+1)'inci geçişte bir işlemde bulunan aday kalem kümelerinin bulunmasından sonra onların ID'lerinin de S'_{k+1} e eklenmesi gerekir. Bu nedenle bu geçişte ekstra bir maliyet ortaya çıkar. Ancak (k+2)'inci geçişte AprioriTid çalıştırılmaya başlanır. Eğer (k+1)-sık kalem kümesi veya (k+2)-aday kalem kümesi yoksa AprioriTid'i kullanmanın hiçbir avantajı olmayacağı gibi Apriori'den AprioriTid'e geçiş yapmanın da büyük bir maliyeti olur.

AprioriHybrid hemen hemen tüm olaylarda Apriori'den daha iyi performans gösterir. Ancak Apriori'den AprioriTid'e geçiş veri geçişinin son adımında gerçekleşirse, geçiş maliyetinden ötürü AprioriHybrid'in performansı kötüleşir. AprioriHybrid'in Apriori'ye göre avantajı S'nin boyutunun sonraki geçiş aşamalarında azalmasına bağlıdır. Eğer S' uzun bir süre büyük kalıp son adımda ani bir düşüşe uğrarsa AprioriTid'i yalnızca kısa bir süre için kullanacağımız için AprioriHybrid'i kullanmak kazanç getirmez. Diğer taraftan S'nin boyutunda büyük ölçüde azalma olursa AprioriTid'in bir süre kullanılmasının ardından AprioriHybrid'in çalışma süresinde önemli bir kazanç sağlanabilir (Agrawal, Srikant, 1994).

5.1.5 Doğrudan HDA ve Kısaltma (DHP)

DHP algoritması ek kontroller yapılarak Apriori'den elde edilebilir. Bu amaçla, C adlı HDA tablosunda aday kalem kümelerinin oluşturulmasını mümkün olduğunca kısıtlamak için, DHP ek bir HDA tablosunun kullanılmasını sağlar. Ek bilgiler, C'de sadece olası σ -sık olan adayları tutmak için yeni HDA tablosunda tutulurlar (Aşağıda yeni HDA tablosu PossFreq olarak adlandırılmıştır). DHP ayrıca işlemlerdeki gereksiz ürünleri kaldırarak veya sonradan yararsız hale gelen kayıtları tamamen silerek veri tabanını kısaltır (Adamo, 2000).

5.1.5.1 Adayları Süzme

$i \geq 2$ için i 'nci iterasyonda kullanılmak üzere $(i-1)$ 'nci iterasyonda DHP tarafından üretilen HDA tablosu PossFreq _{$i+1$} olarak gösterilir. PossFreq _{$i+1$} sayaç dizisidir. PossFreq _{$i+1$} , h_{i+1} (Cass) ile ifade edilen bir HDA fonksiyonuna verilir. HDA tablolar, aşağıdaki şablonlardaki gibi oluşturulur (Adamo, 2000).

0. İterasyon : D veri tabanı bir kez taranarak PossFreq₂ elde edilir

Şablon 1.

1. **for** (int k = 0; k < PossFreqSize; k++)
2. PossFreq2[k] = 0;
3. **for** all transactions (r,s) in D
4. **for** all 2-cass u such that $u \leq s$
5. PossFreq2[h(2,u)]++;

$i \geq 2$ için **(i-1). İterasyon:** PossFreq _{$i+1$} , D'nin bir kere taranmasıyla PossFreq _{i} 'den elde edilir.

Şablon 2.

1. **for** (int k = 0; k < PossFreqSize; k++);
2. PossFreq _{$i+1$} [k]=0;
3. **for** all transactions (r,s) in D
4. **for** all $(i+1)$ -cass u such that $u \leq s$
5. **if** (PossFreq _{i} [h(i,v)] $\geq |D| * \sigma$ **for** all i-cass v such that $v \leq u$)
6. PossFreq _{$i+1$} [h(i+1,u)]++;

$i \geq 2$ ile herhangi bir $i-1$ iterasyonunda sadece PossFreq _{i} ve PossFreq _{$i+1$} adlı iki ardışık HDA tablosuna ihtiyacımız vardır. Bu nedenle DHP'deki HDA tablolar bir flip-flop indeksi ile bir çift tablo olarak kullanılır.

Tanım:

Herhangi bir $i \geq 2$ ve $|u| = i$ olan bir u kalem kümesi için, eğer ve yalnızca $\text{PossFreq}_i[h(i,u)] \geq |D| * \sigma$ şartı sağlanırsa u , olası σ -sık kalem kümesidir.

Gerçekler:

σ -sık ve olası σ -sık kalem kümeleri arasında ilişkiler aşağıdaki gibidir:

- u eğer σ -sık ise o zaman olası σ -sıktır.
- Eğer bir u kalem kümesi olası σ -sık değilse o zaman u σ -sık olamaz.
- PossFreq_i 'nin girişlerindeki mümkün olan çakışmalar nedeniyle, bunun tersi doğru değildir. Sonuç olarak u olası σ -sık olsa bile minimum destek değeri kontrolünün yapılması gerekir.

$\text{PossFreq}_{i+1}[h(i+1,u)]$ 'ı hesaplama işlemi sadece $(i+1)$ kalem kümelerinin σ -sık olup olmadığı hakkında bilgilenmemizi sağlar. Eğer tüm i -cass v 'ler ($v \leq u$) için $\text{PossFreq}_i[h(i,v)] \geq |D| * \sigma$ şartı sağlanmazsa v bazı i -cass v 'ler için olası σ -sık olmaz. İkinci şarta göre böyle bir cas σ -sık olamaz. Bu nedenle, u da σ -sık olamaz.

5.1.5.2 Veri tabanını Kısaltma (Trimming)

DHP, aşağıdaki olgu ve öneriler ışığında veri tabanını büyük ölçüde kısaltır.

Olgu 1:

$|s| < i$ olan bir (r,s) işlemiyle hiçbir i -cas oluşturulamaz. Bu dedenle $|s| = j$ ise j -cas oluşturma adımı tamamlanır tamamlanmaz (r,s) işlemi D 'den hemen silinebilir.

Öneri:

$|s| \geq i+1$ olan herhangi bir (r,s) işlemi için eğer bir a sembolü s 'nin bazı σ -sık $(i+1)$ -alt kalem kümelerine aitse, a s 'nin en az i σ -sık i -alt kalem kümesine aittir.

Kanıt:

u , s 'nin bir σ -sık $(i+1)$ -subcas'ı olsun ve a , u 'ya ait bir sembol olsun. Şimdi $(u-a)$ cas'ını düşünelim. s 'nin $(u-a)$ ile oluşturulabilecek i adet $(i-1)$ -subcass'ı vardır.

Bunları $v_1, \dots, v_i$ subcas'ları olarak gösterelim. $v_1 + a, \dots, v_i + a$, u 'nun i subcas'larıdır. u σ -sık olduğundan, tüm bu i -subcas'lar da σ -sık olur. Sonuç olarak kanıt tamamlanmış olur.

Sonuç1:

$|s| \geq i+1$ olan herhangi bir (r,s) işlemi için eğer s 'nin bir a sembolü en az i σ -sık i -subcas'na ait değilse o zaman a , s 'nin hiçbir σ -sık $(i+1)$ -subcas'na ait olamaz.

Bu sonuca göre veri tabanını kısaltma işlemi aşağıdaki gibi gerçekleşir. (r,s) bir işlem, a s 'nin bir sembolü ve $C_i, L_i, C_i(a)$ ve $L_i(a)$ kümeler olsun.

- C_i : Apriori'nin i 'nci adımda oluşturduğu aday kalem kümerinin kümesi
- L_i : Apriori'nin i 'nci adımda oluşturduğu σ -sık aday kalem kümelerinin kümesi
- $C_i(a)$: s 'nin a 'yı içeren tüm i -subcas'larıyla oluşturduğu C_i 'nin alt kümesi
- $L_i(a)$: s 'nin a 'yı içeren tüm i -subcas'larıyla oluşturduğu L_i 'nin alt kümesi

L_i 'nin, C_i 'nin kısıtlaması olduğuna dikkat edilmelidir; bu nedenle $L_i(a)$, $C_i(a)$ 'nin alt kümesidir.

- $|C_i(a)| < i$ ise $|L_i(a)| < i$

olur. Sonuç olarak, eğer $|C_i(a)| < i$ ise o zaman s 'nin en az i σ -sık i -subcas'na ait olmayan bir a sembolü var demektir. O halde sonuç1'e göre a sembolü s 'nin hiçbir σ -sık $(i+1)$ -subcas'na ait olamaz. Bu nedenle a sembolü sonraki iterasyonlarda yararsız hale gelir ve böylece (r,s) işlemindeki s 'den silinebilir.

Olgu 2:

s , $|s| \geq i+1$ olan rastgele bir cas olsun ve a , s 'nin bir sembolü olsun. s 'nin a içeren tüm $(i+1)$ -subcas'ı olan u için u 'nun en az bir olası σ -sık olamayan i -subcas v 'si vardır. O zaman s 'nin a içeren hiçbir $(i+1)$ -subcas'ı σ -sık olamaz.

Sonuç olarak s 'nin a içeren hiçbir j -subcas'ı ($j \geq i+1$) σ -sık olamaz ve a , o anda incelenen (r,s) işlemindeki s 'den silinebilir.

5.1.5.3 DHP Algoritması:

DHP algoritması kalem kümelerinin oluşturulması esnasında veri tabanını kısaltma işlemi dışında Apriori gibi çalışır. Her adımda kısaltma işlemi olası σ -sık kalem kümelerini saymak için kullanılan HDA tablolara (Prosedürdeki PossFreq dizisi) dayanır. HDA tabloları sayma işlemi Çizelge 5.8 (4-9 ve 14-16 nolu satırlar arası) ve Çizelge 5.11 (6-8 nolu satırlar arası) ile yapılır. HDA tablo sırası, bir tablo çifti (örneğin, PossFreqSize[2][PossFreqSize]) ve bir flip-flop indeksi (örneğin, bin) yoluyla belirlenir. Veri tabanını kısaltma işlemi olgul'e göre

Çizelge 5.10'da (19-21 nolu satırlar arasında), Çizelge 5.12'de (20-24 satırları arası) ve Çizelge 5.13'de (12-16 satırları arası) gerçekleştirilir.

Çizelge 5.10 DHP algoritması (Adamo, 2000)

```

1.  SetofCass L[N];
2.  int PossFreq[2][PossFreqSize];
3.  int bin = 0;
4.  for (int k = 0; k < PossFreqSize; k++)
5.    PossFreq[bin][k] = 0;
6.  for all transactions (r,s) in D {
7.    count all 1-cass in s;
8.    for all 2-cass u in s
9.      PossFreq[bin][h(2,u)]++;
10. }
11. L[1] = { $\sigma$ -frequent 1-cass};
12. for (int i=2; L[i-1] != {}; i++) {
13.   CandTablePtr C = generate_candidates (L[i-1],bin);
14.   bin = (bin+1) % 2;
15.   for (int k = 0; k < PossFreqSize; k++)
16.     PossFreq[bin][k] = 0;
17.   for all transactions (r,s) in D such that |s| >= i {
18.     D=D- {(r,s)};
19.     Cas s_trimmed = count_candidates_and_trim(C,s,1,i);
20.     if (|s_trimmed| > i) {
21.       Cas s_trimmed' = count_PossFreq_and_trim(bin,s_trimmed,i);
22.       if (|s_trimmed'| > i)
23.         D=D U {(r,s_trimmed')};
24.     }
25.   }
26.   L[i] = {};
27.   for all cass u in C {
28.     if (get_count(u) >=  $\sigma$ )
29.       L[i] = L[i] U {u};
30.   }
31.   delete C; 32. }

```

generate_candidates prosedürü Apriori'de tanımlandığı gibidir. Fakat farklı olan tarafı burada C'ye ekleme işleminin PossFreq[h(i,u)] >= |D|.σ koşuluna göre kısıtlanmasıdır. Gerçekten de sadece olası σ -sık aday kalem kümeleri eklenmeye ihtiyaç duyar.

Çizelge 5.11 Aday oluşturma prosedürü(Adamo, 2000)

```

1.  CandTablePtr generate_candidates (SetofCass L, int bin) {
2.      CandTablePtr C = new CandTable[CandTableSize];
3.      let i-1 be the length of cass found in L;
4.      for all pairs (s.a,s.b) in  $L^2$  such that  $a < b$  {
5.          Cas u = s.a.b;
6.          if (PossFreq[bin][h(I,u)]  $\geq |D| * \sigma$ )
7.              insert (C,u,1);
8.      }
9.      return C;
10. }
```

Bir sonraki prosedür Apriori’de tanımlanan count_candidates prosedüründen elde edilmiştir. Ek satırları vardır (2-4, 13 ve 14, 20-24 nolu satırlar). Bu satırlar daha önceden elde edilen sonuç I' ’de temellenmiştir.

Çizelge 5.12 Adayları sayma ve kısaltma prosedürü (Adamo, 2000)

```

1.  Cas count_candidates_and_trim
      (CandTablePtr hash_table_ptr, Cas s, int start, int i) {
2.      attach counter to all attributes a in s;
3.      for all attributes a in s
4.          init_count(a);
5.      for (int j = start; j  $\leq$  |s|-i+1; j++) {
6.          int rank = h(get_attribute(s,j));
7.          if (hash_table_ptr[rank].type == Leaf) {
8.              LeafPtr leaf_ptr = hash_table_ptr[rank].ptr;
9.              if (leaf_ptr != NULL)
10.                 for all cass u in leaf_ptr->list
11.                     if (u  $\leq$  s) {
12.                         increment_count(u);
13.                         for all attributes a in u
14.                             increment_count(a);
15.                     }
16.             } else {
17.                 count_candidates_and_trim (hash_table_ptr, s, j+1, i-1);
18.                 // (s,j+1) s’nin (j+1)’nci sembolünden başlayan kuyruğunu oluşturur.
19.             }
20.             Cas s_trimmed =  $\Phi$ ;
21.             for all attributes a in s
22.                 if (get_count(a)  $\geq$  i)
23.                     s_trimmed = s_trimmed.a;
24.             return s_trimmed;
25. }
```


Sonraki prosedür PossFreq HDA tablosunun girişlerindeki olası σ -sık (i+1)-subcas'larını sayar. Bu prosedürde s'ye ait u adlı (i+1)-subcas'ında bulunan semboller ilerleyerek sayılır (9-10 nolu satırlar). Sonuçta, veri tabanını kısaltma işlemi *olgu 2*'ye göre yapılır (12-16 nolu satırlar).

Çizelge 5.13 PossFreq dizisini sayma ve kısaltma (Adamo, 2000)

```

1.  Cas count_PossFreq_and_trim (int bin, Cas s, int i) {
2.      attach counter to all attributes a in s;
3.      for all attributes a in s
4.          init_count(a);
5.      int previous_bin= (bin+1) % 2;
6.      for all (i+1)-subcass u of s
7.          if (for all i-subcass v of u, PossFreq [previous_bin][h(i,v)] >= |D| *  $\sigma$ ) {
8.              PossFreq[bin][h((i+1),u)]++;
9.              for all attributes a in u
10.                 increment_count (a);
11.          }
12.      Cas s_trimmed =  $\Phi$ ;
13.      for all attributes a in s
14.          if (get_count (a) > 0)
15.              s_trimmed = s_trimmed.a;
16.      return s_trimmed;
17.  }
```

5.2 Veri Hazırlama

Bu bölümde tez çalışmasında kullanılan veri ambarları ve onların özellikleri anlatılmıştır.

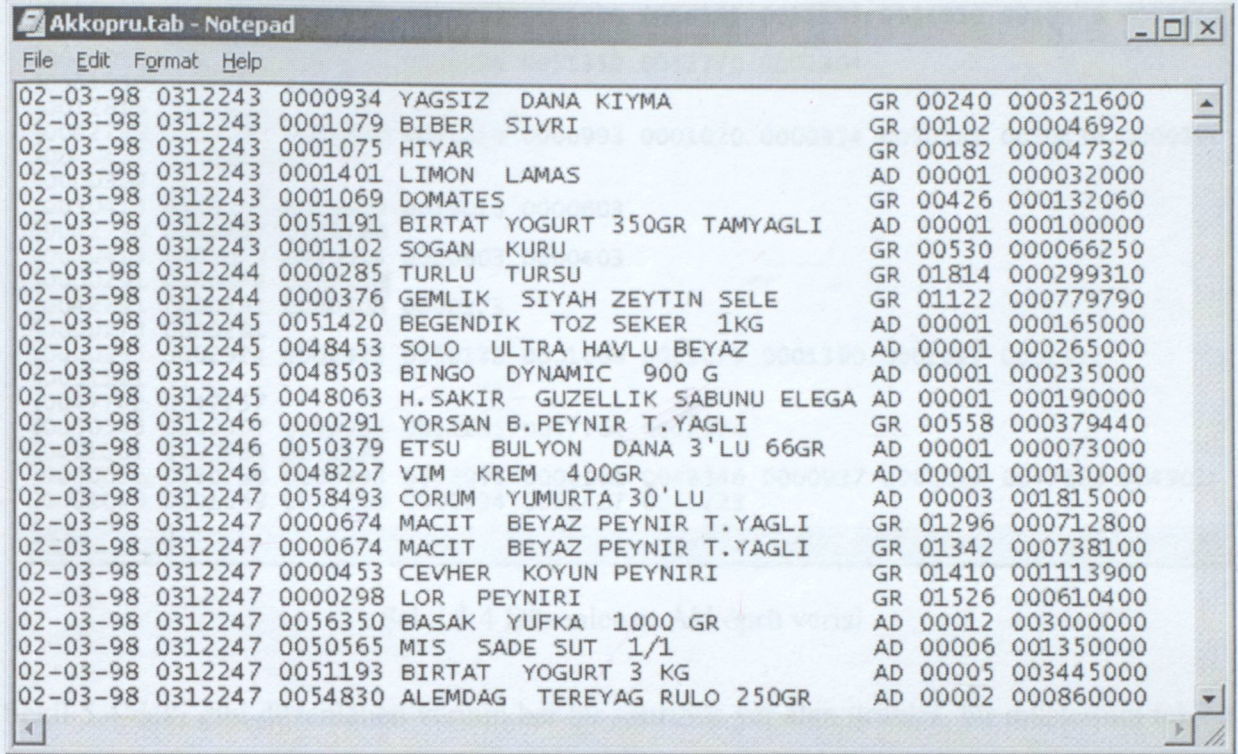
5.2.1 Tez Çalışmasında Kullanılan Veri Ambarları

Bu çalışmada; Akköprü, BMS-WebView-1 ve BMS-WebView-2 adlı üç çeşit veri ambarı kullanılmıştır. Akköprü verisi Migros Mağazalarından tez çalışmasında kullanılmak amacıyla izinli olarak elde edilmiştir. BMS-WebView-1 ve BMS-WebView-2 verileri de internette akademik çalışmalara destek vermek amacıyla serbest bırakılan kaynaklardan sağlanmıştır.

5.2.1.1 Akköprü Verisi

Akköprü verisi, Migros Mağazalarının Ankara Akköprü şubesinden elde edilen bir haftalık market alışveriş verisini içermektedir. Akköprü, 11432 adet işlem ve 4881 çeşit ürün içeren,

7,31 MB büyüklüğünde bir metin dosyasında yer almaktadır. Bu veriden alınan küçük bir örnek şekil 5.3'de gösterilmektedir:

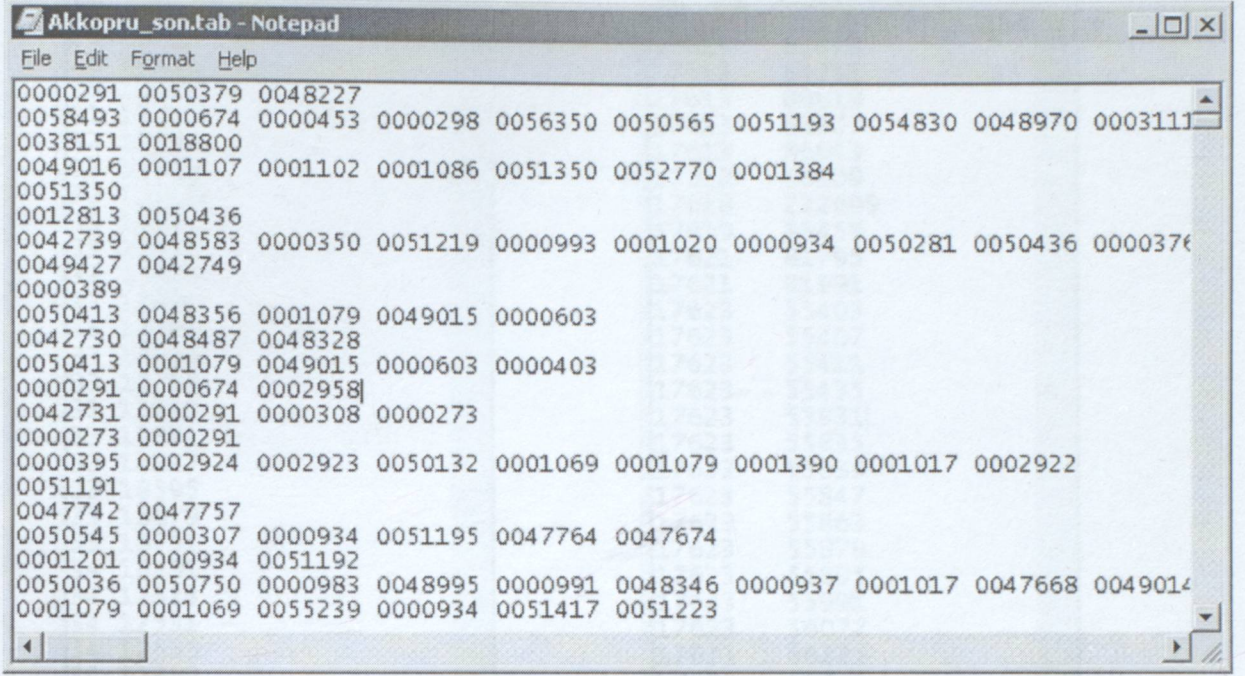


Date	Code	Item Name	Unit	Price
02-03-98	0312243	0000934 YAGSIZ DANA KIYMA	GR 00240	000321600
02-03-98	0312243	0001079 BIBER SIVRI	GR 00102	000046920
02-03-98	0312243	0001075 HIYAR	GR 00182	000047320
02-03-98	0312243	0001401 LIMON LAMAS	AD 00001	000032000
02-03-98	0312243	0001069 DOMATES	GR 00426	000132060
02-03-98	0312243	0051191 BIRTAT YOGURT 350GR TAMYAGLI	AD 00001	000100000
02-03-98	0312243	0001102 SOGAN KURU	GR 00530	000066250
02-03-98	0312244	0000285 TURLU TURSUSU	GR 01814	000299310
02-03-98	0312244	0000376 GEMLIK SIYAH ZEYTIN SELE	GR 01122	000779790
02-03-98	0312245	0051420 BEGENDIK TOZ SEKER 1KG	AD 00001	000165000
02-03-98	0312245	0048453 SOLO ULTRA HAVLU BEYAZ	AD 00001	000265000
02-03-98	0312245	0048503 BINGO DYNAMIC 900 G	AD 00001	000235000
02-03-98	0312245	0048063 H.SAKIR GUZELLIK SABUNU ELEGA	AD 00001	000190000
02-03-98	0312246	0000291 YORSAN B.PEYNIR T.YAGLI	GR 00558	000379440
02-03-98	0312246	0050379 ETSU BULYON DANA 3'LU 66GR	AD 00001	000073000
02-03-98	0312246	0048227 VIM KREM 400GR	AD 00001	000130000
02-03-98	0312247	0058493 CORUM YUMURTA 30'LU	AD 00003	001815000
02-03-98	0312247	0000674 MACIT BEYAZ PEYNIR T.YAGLI	GR 01296	000712800
02-03-98	0312247	0000674 MACIT BEYAZ PEYNIR T.YAGLI	GR 01342	000738100
02-03-98	0312247	0000453 CEVHER KOYUN PEYNIRI	GR 01410	001113900
02-03-98	0312247	0000298 LOR PEYNIRI	GR 01526	000610400
02-03-98	0312247	0056350 BASAK YUFKA 1000 GR	AD 00012	003000000
02-03-98	0312247	0050565 MIS SADE SUT 1/1	AD 00006	001350000
02-03-98	0312247	0051193 BIRTAT YOGURT 3 KG	AD 00005	003445000
02-03-98	0312247	0054830 ALEMDAG TEREYAG RULO 250GR	AD 00002	000860000

Şekil 5.3 Akköprü verisi

Akköprü verisinin her satırında ürünün satış tarihi, kodu, adı, miktarı, fiyatı ve fiş numarası bilgileri yer almaktadır.

Akköprü verisinden anlamlı ilişkisel kurallar üretilebilmesi için, her müşteri tarafından tek bir alışverişte satın alınan ürünlerin kodları bilgisine gerek vardır. Bu nedenle bu verinin bütün algoritmalara aynı formatta girdi olabilmesi için düzenlenmesi gerekmektedir. Bu amaçla aynı fiş numarasına sahip işlemlerde (tek bir müşterinin aynı anda satın aldığı ürünleri içeren işlemler), satın alınan ürünlerin kodları, aralarında birer boşluk olacak şekilde tek bir satırda yazılmıştır. Şekil 5.4, hazırlanan veriden küçük bir örnek içermektedir.

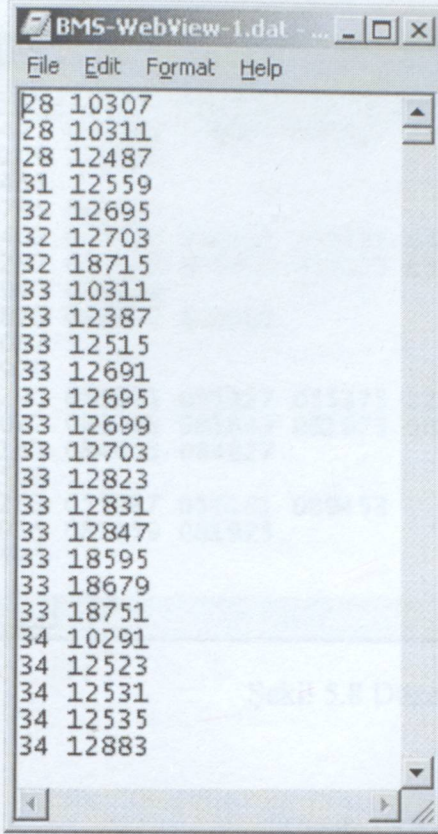


Şekil 5.4 Düzenlenen Akköprü verisi

Şekil 5.4'deki gibi düzenlenen verinin her bir satırında yer alan ürünler, bir müşterinin tek bir seferdeki alışveriş bilgisini içermektedir. Verinin bu şekilde düzenlenmesinin amacı ise hangi ürünlerin bir arada sık olarak satıldığının araştırılması ve bununla ilgili ilginç ilişkisel kuralların ve istatistiksel sonuçların elde edilmesidir.

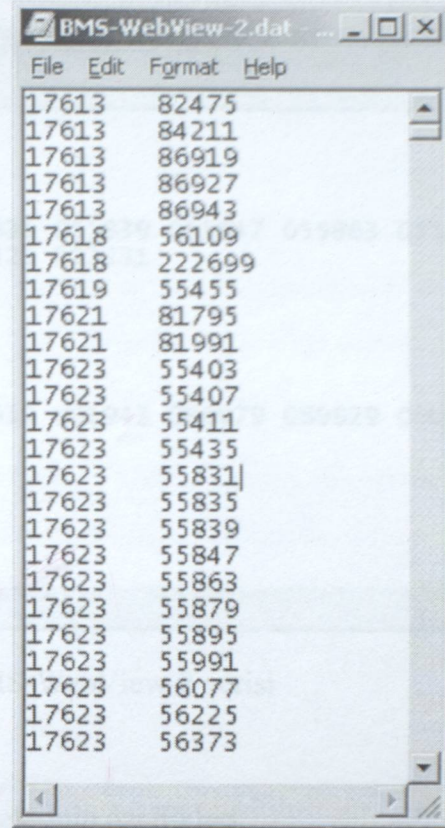
5.2.1.2 BMS-WebView-1 ve BMS-WebView-2 Verileri

BMS-WebView-1 ve BMS-WebView-2, veri madenciliği alanında yapılan akademik çalışmalara destek vermek amacıyla internette serbest bırakılan, iki e-ticaret web sitesinden elde edilen birkaç aylık satış verileridir. Her iki veri de yukarıda anlatılan Akköprü verisinden biraz farklı olarak, bir satırda sadece fiş numarası ve ürün kodu bilgilerini içeren metin dosyalarında tutulmaktadır. Her iki veri kümesi için de amaç, ziyaretçiler tarafından web sitesine yapılan tek bir ziyarette incelenen ürünler arasındaki ilişkiyi bulmaktır. Şekil 5.5 ve şekil 5.6'da bu iki veri kümesine ait küçük birer örnek verilmiştir.



ID	Value
28	10307
28	10311
28	12487
31	12559
32	12695
32	12703
32	18715
33	10311
33	12387
33	12515
33	12691
33	12695
33	12699
33	12703
33	12823
33	12831
33	12847
33	18595
33	18679
33	18751
34	10291
34	12523
34	12531
34	12535
34	12883

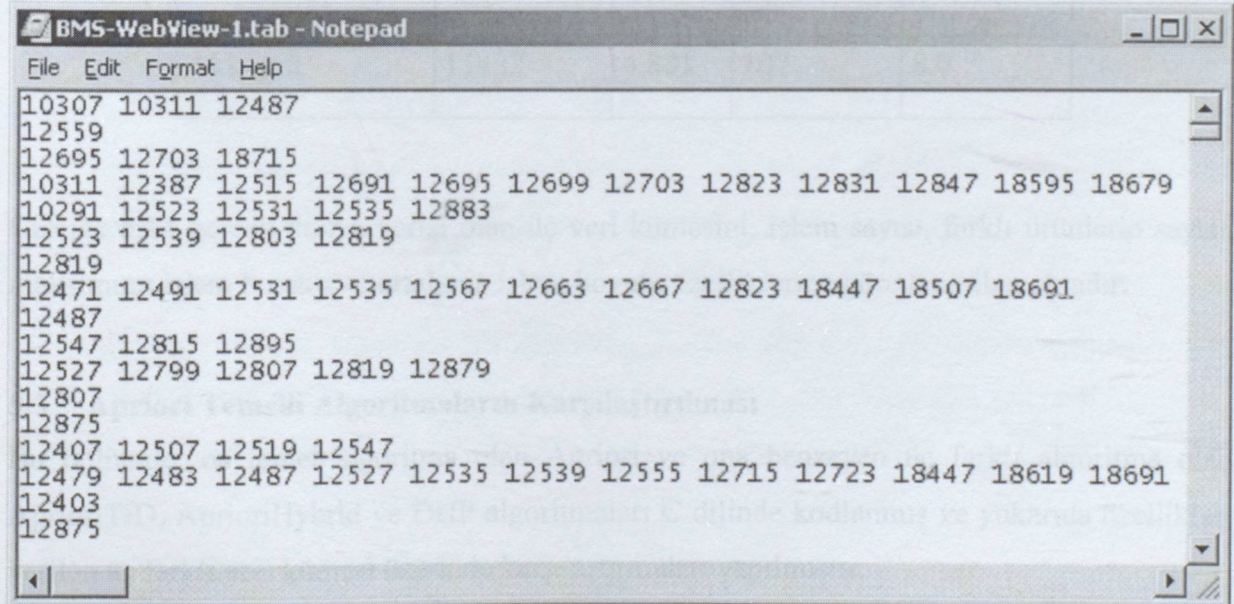
Şekil 5.5 BMS-WebView-1 verisi



ID	Value
17613	82475
17613	84211
17613	86919
17613	86927
17613	86943
17618	56109
17618	222699
17619	55455
17621	81795
17621	81991
17623	55403
17623	55407
17623	55411
17623	55435
17623	55831
17623	55835
17623	55839
17623	55847
17623	55863
17623	55879
17623	55895
17623	55991
17623	56077
17623	56225
17623	56373

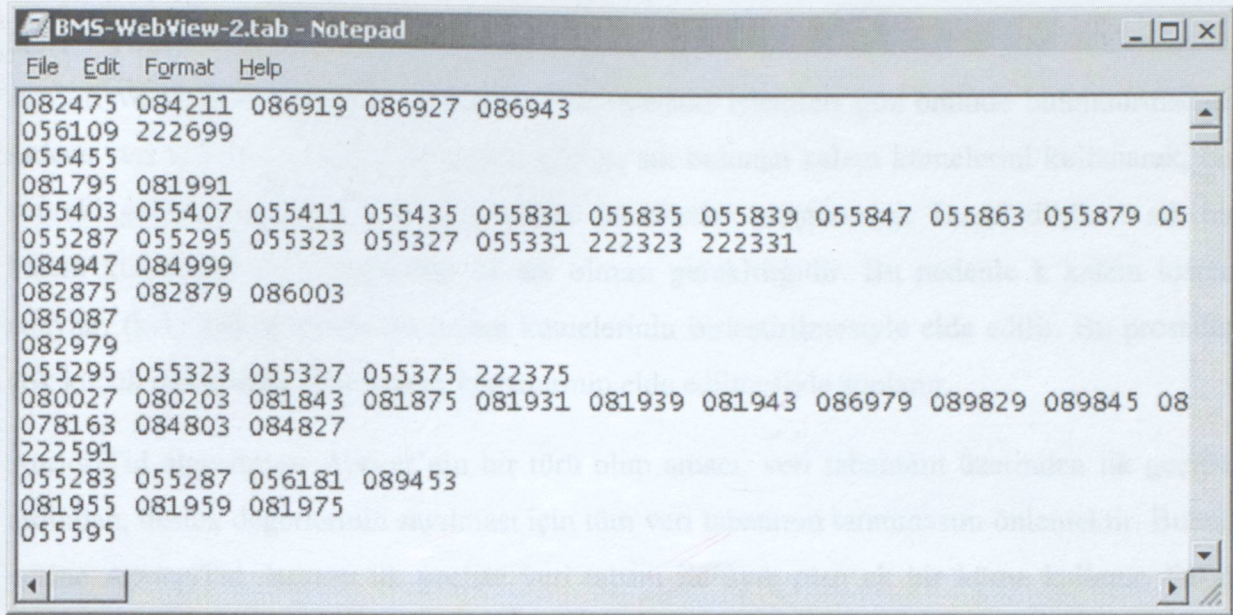
Şekil 5.6 BMS-WebView-2 verisi

Şekil 5.7 ve şekil 5.8’de de bu iki veri kümesinin algoritmalarla uygun girdi haline getirilmesi için düzenlenen son durumlarına ait birer örnek verilmiştir.



ID	Value
10307	10311 12487
12559	
12695	12703 18715
10311	12387 12515 12691 12695 12699 12703 12823 12831 12847 18595 18679
10291	12523 12531 12535 12883
12523	12539 12803 12819
12819	
12471	12491 12531 12535 12567 12663 12667 12823 18447 18507 18691
12487	
12547	12815 12895
12527	12799 12807 12819 12879
12807	
12875	
12407	12507 12519 12547
12479	12483 12487 12527 12535 12539 12555 12715 12723 18447 18619 18691
12403	
12875	

Şekil 5.7 Düzenlenen BMS-WebView-1 verisi



Şekil 5.8 Düzenlenen BMS-WebView-2 verisi

Çizelge 5.14 Veri kümelerinin özellikleri

	İşlem Sayısı	Farklı Ürünler	Maksimum İşlem Boyu	Ortalama İşlem Boyu
BMS-WebView-1	59,602	497	267	2.5
BMS-WebView-2	77,512	3,340	161	5.0
Akköprü	11432	4,881	107	8.0

Çizelge 5.14 gerçek dünya verisi olan üç veri kümesini, işlem sayısı, farklı ürünlerin sayısı, maksimum işlem boyutu ve ortalama işlem boyutu özelliklerine göre tanımlamaktadır.

5.3 Apriori Temelli Algoritmaların Karşılaştırılması

Bu bölümde, en temel algoritma olan Apriori ve ona benzeyen üç farklı algoritma olan AprioriTID, AprioriHybrid ve DHP algoritmaları C dilinde kodlanmış ve yukarıda özellikleri verilen üç farklı veri kümesi üzerinde karşılaştırmaları yapılmıştır.

5.3.1 Yöntem Farklılıkları

Apriori ve AprioriTID algoritmaları veri tabanındaki işlemleri göz önünde bulundurmadan, sadece veri tabanı üzerinden bir önceki geçişte sık bulunan kalem kümelerini kullanarak, bir sonraki geçişte sayılacak olan aday kalem kümelerini oluştururlar. Temel düşünce sık bir kalem kümesinin alt kümelerinin de sık olması gerekliliğidir. Bu nedenle k kalem içeren adaylar, $(k-1)$ kalem içeren sık kalem kümelerinin birleştirilmesiyle elde edilir. Bu prosedür çok küçük miktardaki aday kalem kümelerinin elde edilmesiyle sonlanır.

AprioriTid algoritması Apriori'nin bir türü olup amacı, veri tabanının üzerinden ilk geçişin ardından, destek değerlerinin sayılması için tüm veri tabanının taranmasını önlemektir. Bunun yerine AprioriTid, boyutu ilk geçişte veri tabanı ile aynı olan ek bir küme kullanır. Fakat sonraki geçişlerde bu kümenin boyutu veri tabanından daha küçük olacağından okuma gücünden kazanç sağlanacaktır.

AprioriHybrid algoritması ise bu iki algoritmanın (Apriori ve AprioriTid'in) en iyi özelliklerinin birleştirilmesiyle oluşturulmuştur. Veri tabanından ilk geçişlerde Apriori daha iyi performans verirken sonraki geçişlerde AprioriTid daha iyidir. Her iki algoritma da aynı aday oluşturma prosedürünü kullandıkları için aynı kalem kümelerini sayarlar. Sonraki geçişlerde aday kalem kümelerinin sayısının azalmasına rağmen Apriori veri tabanındaki her kaydı kontrol etmeye devam eder. Diğer taraftan AprioriTid'in çalışma hızı, gerçek veri tabanının yerine kullandığı ek kümenin boyutunun zamanla daha çok küçülmesinden ötürü artar. AprioriHybrid algoritması ise bu iki algoritmanın iyi özelliklerini birleştirir ve ilk geçişlerde Apriori gibi, sonraki geçişlerde de AprioriTid gibi çalışır.

DHP algoritması ek kontrollerle Apriori'den elde edilebilen bir algoritmadır. DHP, aday kalemlerinin C olarak gösterilen HDA tablosunda oluşturulmasını mümkün olduğunca kısıtlamak için ek bir HDA tablosu kullanır. Bu ek tabloda sadece kısmi σ -sık kalem kümelerini bulundurur. Böylece tüm adayların C HDA tablosunda tutulması önlenmiş olur. DHP ayrıca ilerleyen bir şekilde veri tabanının işlemlerindeki özellikleri (örneğin ürünleri) veya yararsız hale gelen tüm işlemleri eleyerek veri tabanını da kısaltır.

5.3.2 Performans Farklılıklarına Göre Karşılaştırma

Bu tez çalışmasında incelenen dört algoritmanın her birinin ayrı yöntemler kullanmasından dolayı farklı çalışma süreleri vardır. Her birinin veri tabanını kullanma ve tarama şekli ile aldıkları parametreler birbirinden farklıdır. Sadece kullandıkları veri yapıları benzerdir ve hepsinde HDA ağaç yapısı kullanılmıştır. Aynı veri yapısının kullanılması birbirinden farklı

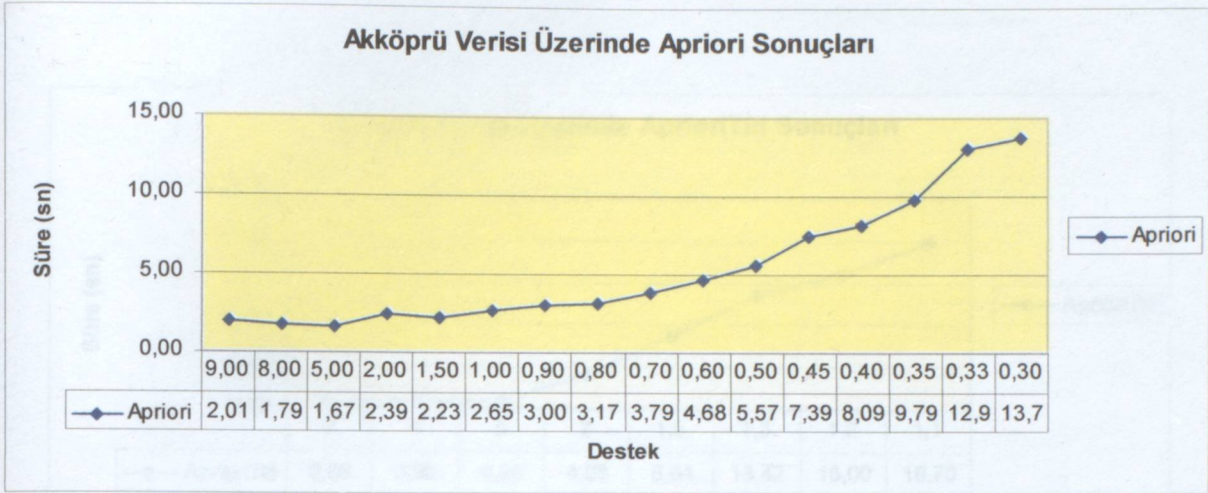
bu dört algoritmanın programlanmasını biraz daha kolaylaştırmıştır. Her algoritmanın aynı veri kümesi üzerindeki sonuçları aşağıdaki tablolarda gösterilmiştir.

5.3.2.1 Akköprü Verisi Üzerinde Performans Farklılıklarına Göre Karşılaştırma

Aşağıda Apriori algoritmasının, Migros Akköprü Şubesi'nden alınan Akköprü verisi üzerindeki sonuçları verilmiştir:

Çizelge 5.15 Akköprü verisi üzerinde Apriori sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kalem Sayısı
656	0,3	13,72	2132
590	0,33	12,98	1809
535	0,35	9,797	1559
469	0,4	8,094	1225
410	0,45	7,39	971
352	0,5	5,58	808
284	0,6	4,68	585
229	0,7	3,80	436
195	0,8	3,17	354
163	0,9	3	281
145	1	2,66	232
92	1,5	2,23	119
61	2	2,39	72
10	5	1,67	11
3	8	1,79	3
1	9	2,01	1

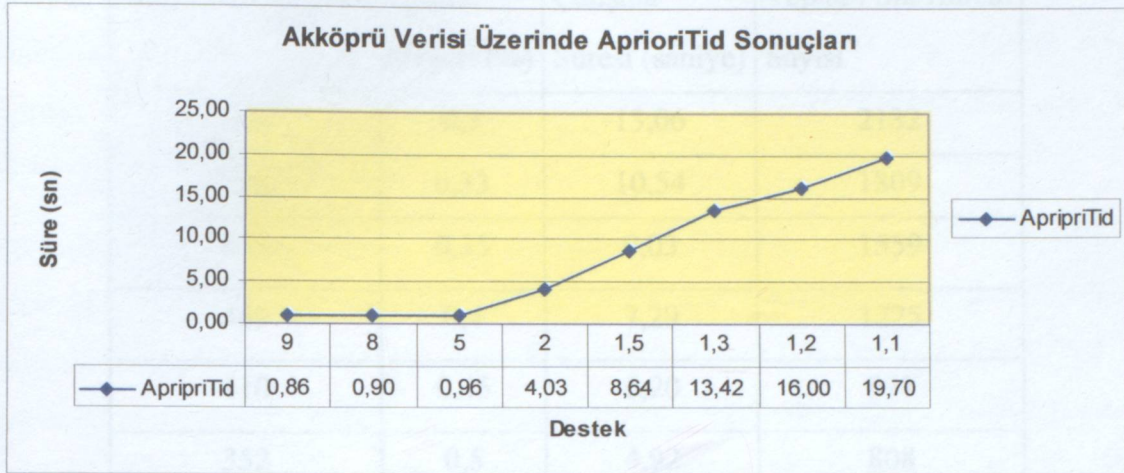


Şekil 5.9 Akköprü verisi üzerinde Apriori sonuçları

Çizelge 5.15'deki sonuçlara göre, destek değerlerinin küçülmesiyle orantılı olarak, sık 1-kalem kümelerinin sayısının da arttığı görülür. Küçük destek şartını sağlayan daha çok ürün olacağı için bu sonuç normaldir. Sık kalem kümelerinin sayısındaki bu artışla birlikte çalışma süreleri de artmaktadır. Destek değeri % 9'a vardığında algoritma sonlanır ve bir tane sık kalem kümesi üretilir. Bu sonuçlara göre şekil 5.9'da görüldüğü gibi destek değerleri ile çalışma süreleri arasında ters orantı vardır.

Çizelge 5.16 Akköprü verisi üzerinde AprioriTid sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kalem Sayısı
145	1	35,98	232
130	1,1	19,70	194
122	1,2	16	171
112	1,3	13,42	151
92	1,5	8,64	119
61	2	4,03	72
10	5	0,97	11
3	8	0,90	3
1	9	0,86	1



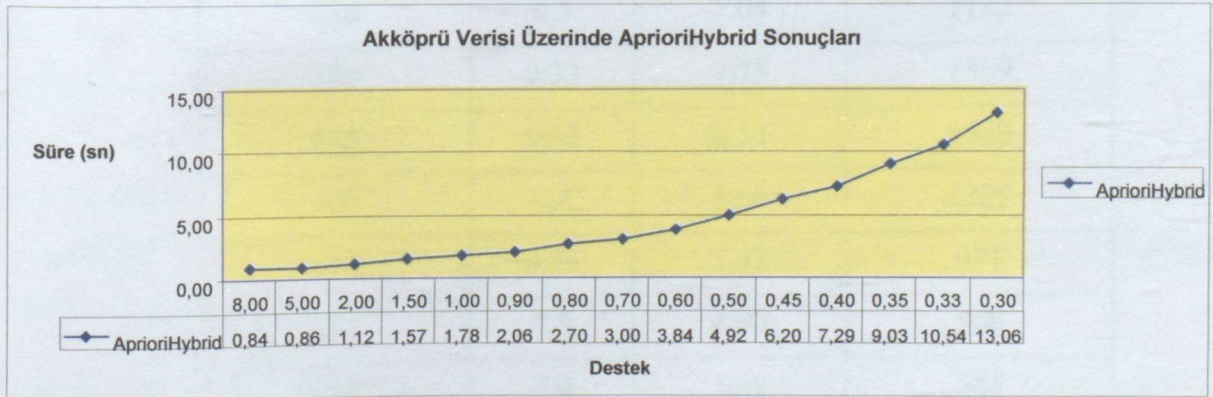
Şekil 5.10 Akköprü verisi üzerinde AprioriTid sonuçları

Çizelge 5.16’da AprioriTid algoritmasının Akköprü verisi üzerindeki sonuçları verilmiştir. Burada Apriori için değerlendirilen tüm destek değerleri AprioriTid için test edilmemiştir. Bunun nedeni çok küçük destek değerleri için AprioriTid’in çalışma süresini çok aşmasıdır. Çünkü daha önce anlatıldığı gibi AprioriTid, veri tabanının ilk taranmasında elde edilen sık 1-kalem kümelerinin tüm ikili alt kümelerini oluşturup onları belleğe yüklemektedir ve bu da bellek yetersizliğine neden olmaktadır. O nedenle algoritma, destek değerinin daha büyük ve sonuçta üretilen sık 1-kalem kümelerinin sayısının küçük olduğu durumlarda sonuç vermektedir.

Bu iki algoritmayı birbirleriyle karşılaştıracak olursak belli bir destek değerinden sonra AprioriTid’in Apriori’ye göre daha iyi performans gösterdiği görülmektedir. Bu destek değeri bu veri kümesi için 5’dir. Akköprü verisinde destek değerinin 5 ve onun üstünde olduğu durumlarda AprioriTid daha iyidir. Görüldüğü gibi AprioriTid yüksek destek değerleri için iyi sonuç vermektedir.

Çizelge 5.17 Akköprü verisi üzerinde AprioriHybrid sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kalem Sayısı
656	0,3	13,06	2132
590	0,33	10,54	1809
535	0,35	9,03	1559
469	0,4	7,29	1225
410	0,45	6,20	971
352	0,5	4,92	808
284	0,6	3,84	585
229	0,7	3	436
195	0,8	2,70	354
163	0,9	2,06	281
145	1	1,78	232
92	1,5	1,57	119
61	2	1,12	72
10	5	0,86	11
3	8	0,84	3



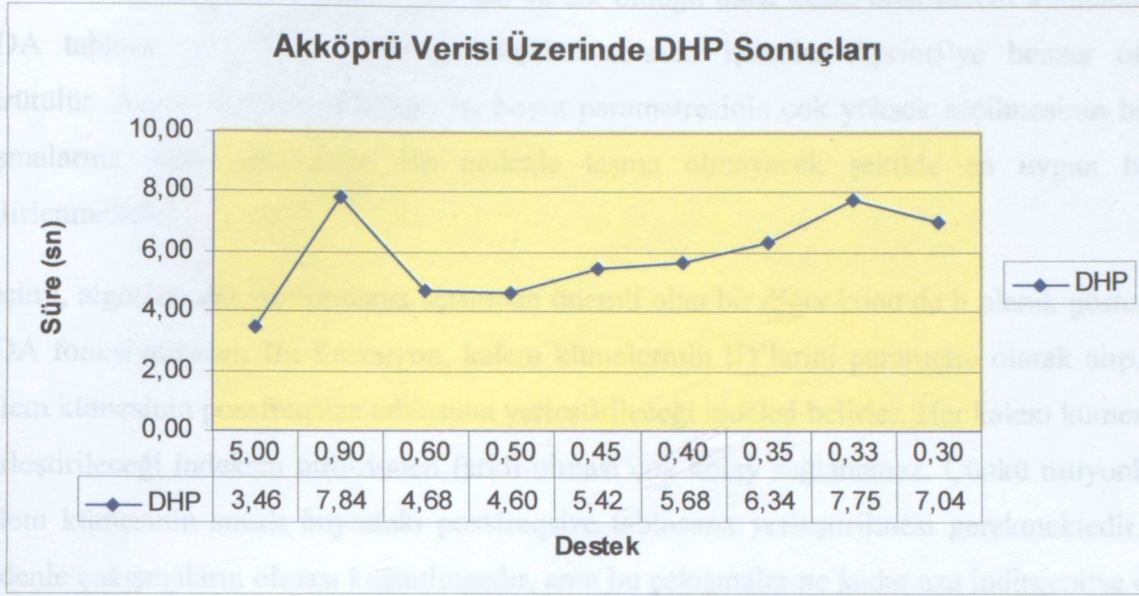
Şekil 5.11 Akköprü verisi üzerinde AprioriHybrid sonuçları

AprioriHybrid algoritması, Apriori ve AprioriTid algoritmalarının iyi özelliklerinin birleştirilmesiyle oluşturulmuştur. AprioriHybrid algoritmasının Akköprü verisi üzerindeki sonuçları ise Çizelge 5.17’de verilmiştir. Bu tablo sonuçlarına göre çalışma sürelerinin Apriori ve AprioriTid’e göre daha kısa olduğu ve algoritmanın diğerlerine göre daha iyi performans gösterdiği görülmektedir.

AprioriHybrid algoritmasının performansını belirleyen en önemli özellik aldığı parametrelerdir. Çalışma süresini etkileyen en önemli parametre; Apriori’nin ürettiği aday kalemlerinin sayısının belli bir sınır değerine ulaşması durumunda AprioriTid’e geçiş yapılmasını sağlayan geçiş sınır parametresidir. Eğer programın sonlanmasına yakın Apriori’den AprioriTid’e geçiş yapılırsa AprioriTid bir defa çalıştırılarak veya hiç çalıştırılmadan program sonlanabilir. O zaman bu geçiş işleminin getirdiği maliyet oldukça yüksek olacaktır ve çalışma süresi de bir hayli artacaktır. Örneğin; destek değeri = % 0.8, sınır parametresi 60000 alındığı bir durumda çalışma süresi 11,26 saniye iken destek değeri = % 0.8 ve sınır parametresi de 6000 alındığında çalışma süresinin 2,70 saniye olduğu görülmüştür. Bu sonuç, yukarıda anlatılan durumu çok iyi örneklemektedir. Bu nedenle sınır parametresinin doğru seçildiği durumlarda, AprioriHybrid algoritmasını aynı destek değerleri için diğer algoritmalarla aynı sayıda sık-kalem kümesi ürettiği; fakat daha hızlı çalıştığı görülmüştür.

Çizelge 5.18 Akköprü verisi üzerinde DHP sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kalem Sayısı
656	0,3	7,04	2132
590	0,33	7,75	1809
535	0,35	6,34	1559
469	0,4	5,68	1225
410	0,45	5,42	971
352	0,5	4,60	808
284	0,6	4,68	585
163	0,9	4	281
145	1	7,84	232
10	5	3,46	11



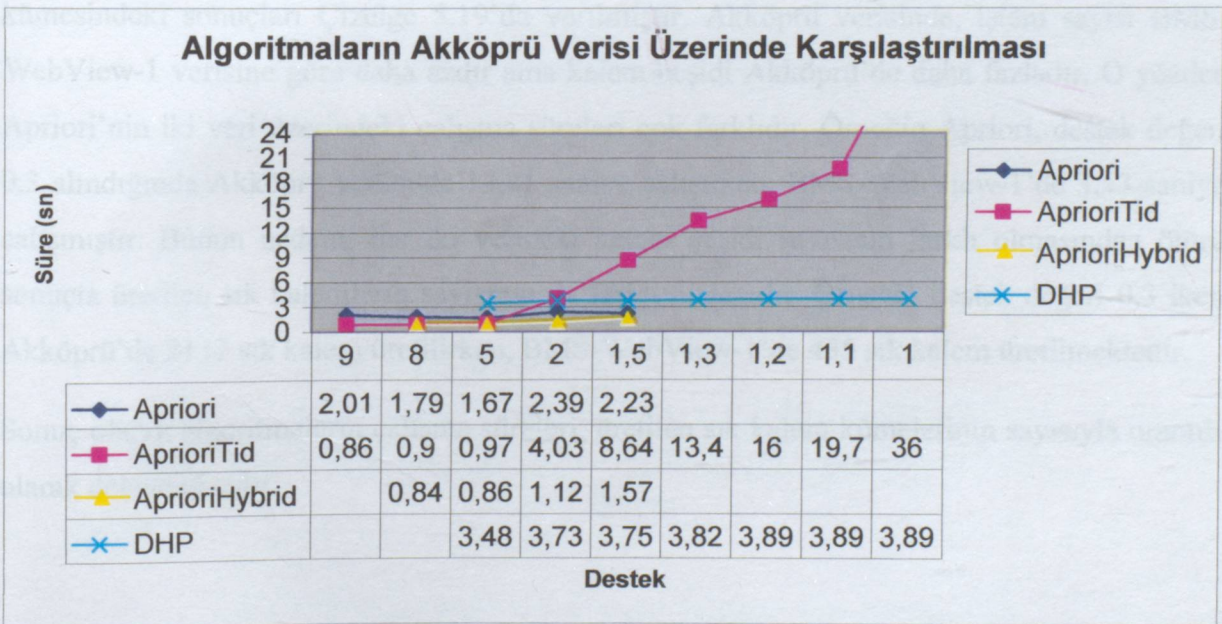
Şekil 5.12 Akköprü verisi üzerinde DHP sonuçları

Çizelge 5.18, DHP algoritmasının Akköprü verisi üzerindeki sonuçlarını göstermektedir. Sonuçlar diğer algoritmalarla karşılaştırılınca destek değerinin küçük olduğu durumlarda DHP'nin daha hızlı olduğu görülür. Örneğin destek değeri %0,3 iken Apriori 13,72 saniyede sonlanırken, DHP 7,04 saniye sürer. Ancak destek değerinin arttığı durumlarda DHP'nin performansı diğer algoritmalarla göre düşer. Örneğin destek değeri %5 alınıncı Apriori 1,67 saniye, DHP ise 3,46 saniyede sonlanır. Bunun nedeni; DHP'nin ilk geçişinin çalışma süresinin, HDA tablosu üretimi için gerekli ekstra taşıma olmasından dolayı Apriori'den biraz daha büyük olmasıdır. Ama sonraki geçişlerde DHP çalışma süresini Apriori'ye göre önemli ölçüde azaltır. Bunun nedeni Apriori'nin her geçişte tüm veri tabanını taraması oysa DHP'nin tüm veri tabanını yalnızca ilk iki geçişte tarayıp sonradan azaltılmış veri tabanını kullanmasıdır. Destek değeri büyük alındığı zaman oluşturulacak olan sık kalem kümelerinin sayısı da az olur, fakat DHP ek kontrollerle aynı işlemleri yapmaya devam eder. Apriori'ye göre ek olarak yapılan bu işlemler sistemi yavaşlatabilir. Ancak destek değerinin küçük ve dolayısıyla üretilecek olan sık kalem kümelerinin sayısının da fazla olduğu durumlarda DHP'nin kullandığı ek kontrollerin avantajından yararlanılabilir.

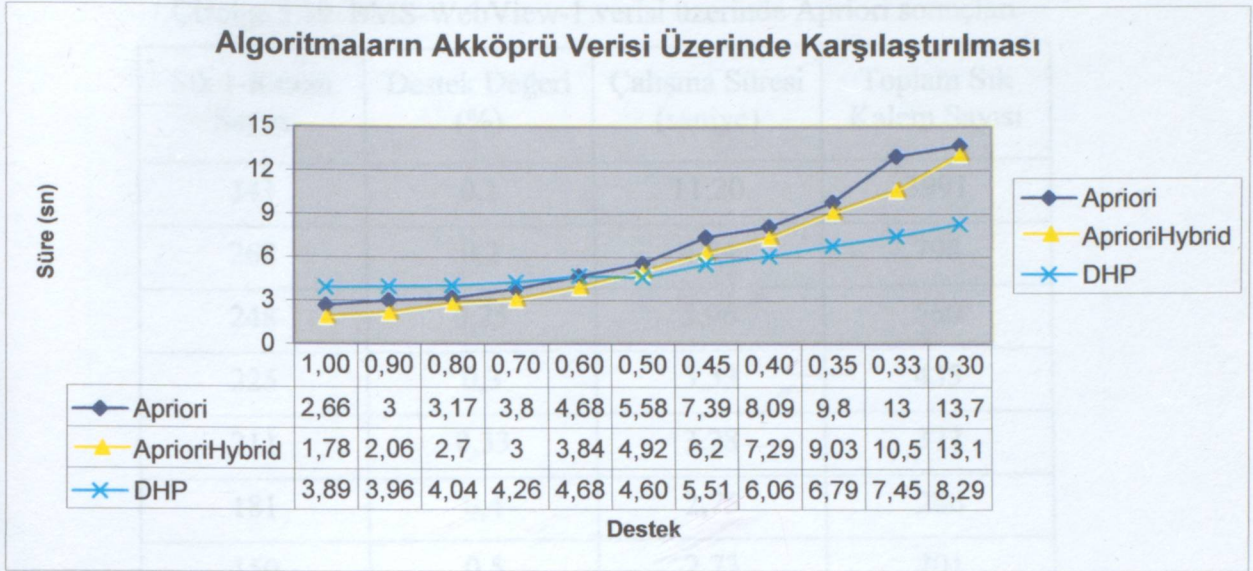
DHP algoritmasının performansını etkileyen en önemli konulardan biri aldığı parametrelerdir. En etkili parametre, C_k HDA tablosunun yerine kullanılan ek possfreqsize tablosunun boyutunu belirleyen parametredir. Bu boyut ne kadar büyük olursa, seçilen HDA fonksiyonuna göre üretilen kalem kümelerinin possfreqsize tablosuna yerleştirilebileceği alan

sayısının fazla olması nedeniyle çakışma sorunu en aza iner. Bu da kalem kümelerinin sık olup olmadığı bilgisini daha iyi doğrular ve sık olduğu daha kesin olan kalem kümeleri asıl HDA tablosu olan C_k 'ya alınır. Bundan sonraki işlemler Apriori'ye benzer olarak yürütülür. Ancak buradaki problem bu boyut parametresinin çok yüksek seçilmesinin bellek taşmalarına neden olmasıdır. Bu nedenle taşma olmayacak şekilde en uygun boyut belirlenmelidir.

Seçimi, algoritmanın performansı açısından önemli olan bir diğer konu da h olarak gösterilen HDA fonksiyonudur. Bu fonksiyon, kalem kümelerinin ID'lerini parametre olarak alıp, her kalem kümesinin possfreqsize tablosuna yerleştirileceği indeksi belirler. Her kalem kümesinin yerleştirileceği indeksin birbirinden farklı olması çok kolay sağlanamaz. Çünkü milyonlarca kalem kümesinin sınırlı boyuttaki possfreqsize tablosuna yerleştirilmesi gerekmektedir. Bu nedenle çakışmaların olması kaçınılmazdır, ama bu çakışmalar ne kadar aza indirgenirse daha sonra HDA süzgeçinden geçirilip asıl HDA tablosuna alınan kalem kümelerin sayısı azaltmış olur. Bu da algoritmanın performansını önemli ölçüde arttırmaktadır. Çünkü çakışmaların çok olduğu durumda, destek değerini sağlayan indekslerdeki kalem kümelerinin sayısının fazla olması nedeniyle bunlar asıl C_k HDA tablosuna aktarılacaktır. Halbuki bunların içinde destek şartını sağlamadığı halde sırf çakışmalar nedeniyle indeksteki eleman sayısının fazla olması sonucunda C_k 'ya alınanlar vardır. Bu nedenle h HDA fonksiyonunu da çakışmaları en aza indirgeyecek şekilde seçmek gerekir.



Şekil 5.13 Algoritmaların Akköprü verisi üzerinde karşılaştırılması



Şekil 5.14 Algoritmaların Akköprü verisi üzerinde karşılaştırılması

Şekil 5.13’de yüksek değerli destek değerleri için, şekil 5.14’de ise düşük değerli destek değerleri için, Akköprü verisi üzerinde incelenen Apriori temelli dört algoritmaya ait karşılaştırma sonuçları verilmiştir.

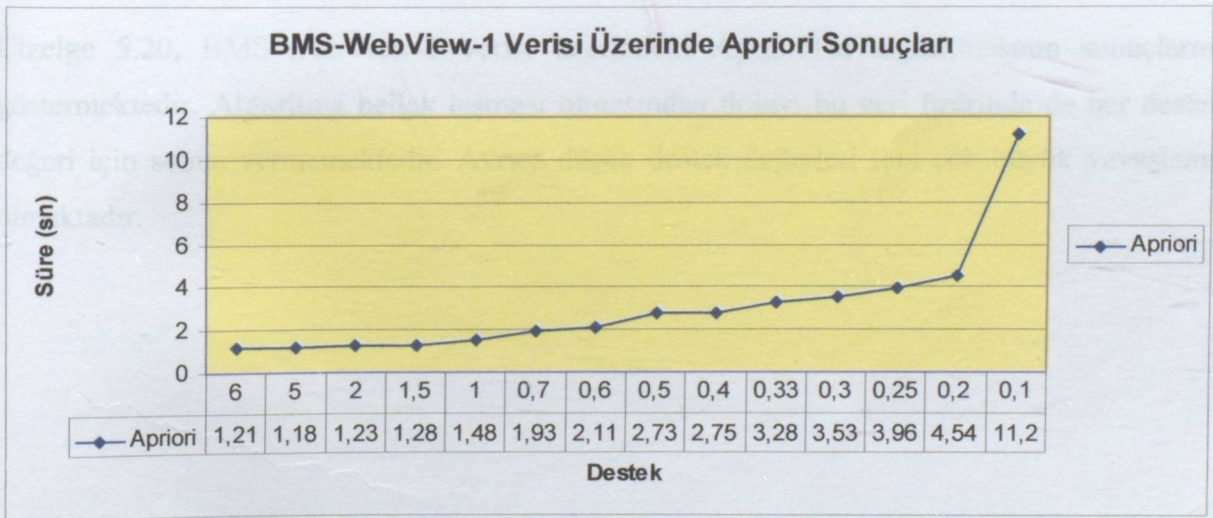
5.3.2.2 BMS-WebView-1 Verisi Üzerinde Performans Farklılıklarına Göre Karşılaştırma

BMS-WebView-1 verisinin özellikleri daha önce anlatılmıştı. Apriori’nin bu veri kümesindeki sonuçları Çizelge 5.19’da verilmiştir. Akköprü verisinde, işlem sayısı BMS-WebView-1 verisine göre daha azdır ama kalem çeşidi Akköprü’de daha fazladır. O yüzden Apriori’nin iki veri üzerindeki çalışma süreleri çok farklıdır. Örneğin Apriori, destek değeri 0,3 alındığında Akköprü verisinde 13,71 saniye çalışırken; BMS-WebView-1’de 3,53 saniye çalışmıştır. Bunun nedeni, her iki verideki kalem çeşidi sayısının farklı olmasından ötürü sonuçta üretilen sık kalemlerin sayısının da farklı olmasıdır. Örneğin destek değeri 0,3 iken Akköprü’de 2132 sık kalem üretilirken, BMS-WebView-1’de 435 sık kalem üretilmektedir.

Sonuç olarak algoritmaların çalışma süreleri, üretilen sık kalem kümelerinin sayısı ile orantılı olarak değişmektedir.

Çizelge 5.19 BMS-WebView-1 verisi üzerinde Apriori sonuçları

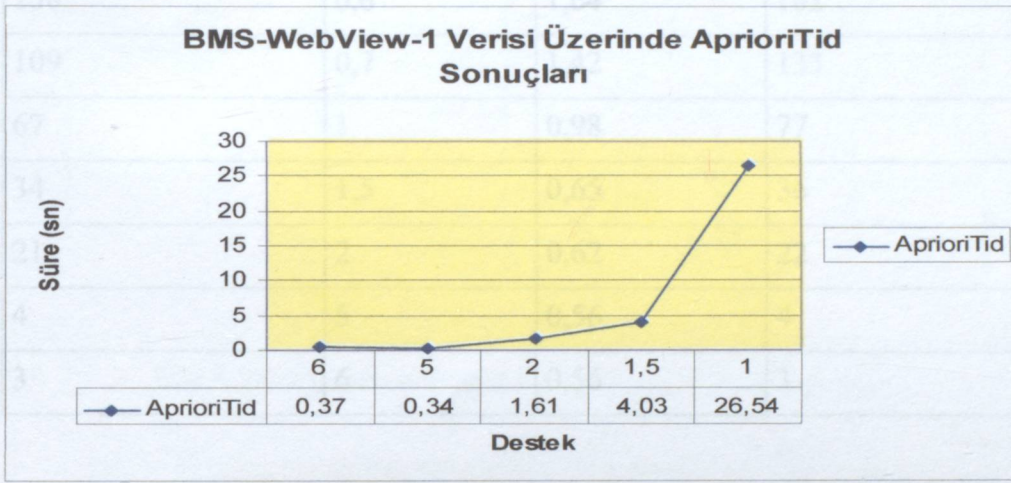
Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kalem Sayısı
343	0,1	11,20	3991
268	0,2	4,54	798
248	0,25	3,96	560
225	0,3	3,53	435
211	0,33	3,28	377
181	0,4	2,75	286
150	0,5	2,73	201
130	0,6	2,11	162
109	0,7	1,93	133
67	1	1,48	77
34	1,5	1,28	36
21	2	1,23	22
4	5	1,18	4
3	6	1,21	3



Şekil 5.15 BMS-WebView-1 verisi üzerinde apriori sonuçları

Çizelge 5.20 BMS-WebView-1 verisi üzerinde AprioriTid sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kelem Sayısı
67	1	26,54	77
34	1,5	2,64	36
21	2	1,61	22
4	5	0,34	4
3	6	0,37	3

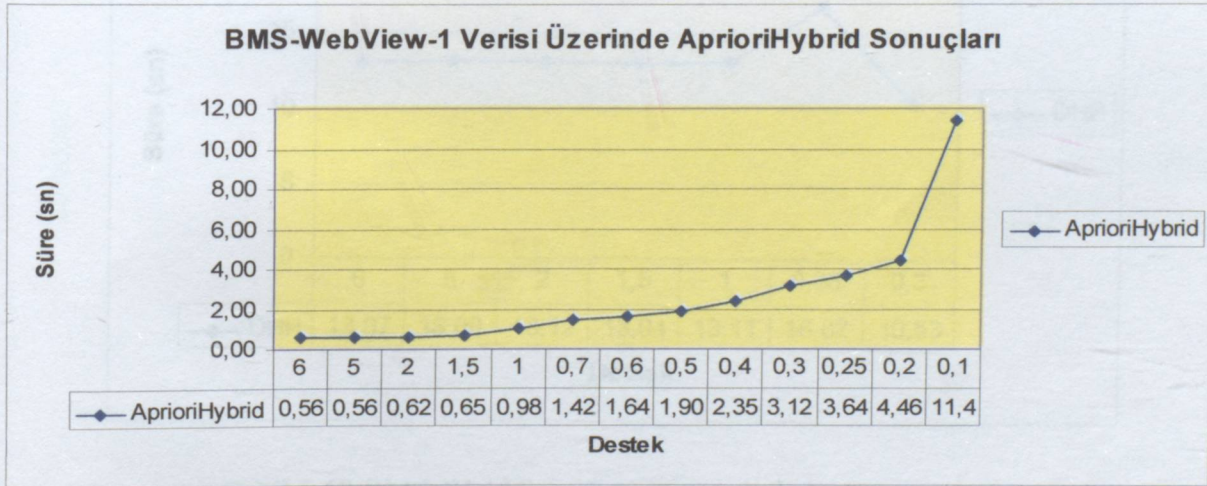


Şekil 5.16 BMS-WebView-1 verisi üzerinde aprioritid sonuçları

Çizelge 5.20, BMS-WebView-1 verisi üzerindeki AprioriTid algoritmasının sonuçlarını göstermektedir. Algoritma bellek taşması olmasından dolayı bu veri üzerinde de her destek değeri için sonuç vermemektedir. Ayrıca düşük destek değerleri için çok büyük yavaşlama olmaktadır.

Çizelge 5.21 BMS-WebView-1 verisi üzerinde AprioriHybrid sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kelem Sayısı
343	0,1	11,42	3991
268	0,2	4,46	798
248	0,25	3,64	560
225	0,3	3,12	435
181	0,4	2,35	286
150	0,5	1,90	201
130	0,6	1,64	162
109	0,7	1,42	133
67	1	0,98	77
34	1,5	0,65	36
21	2	0,62	22
4	5	0,56	4
3	6	0,56	3



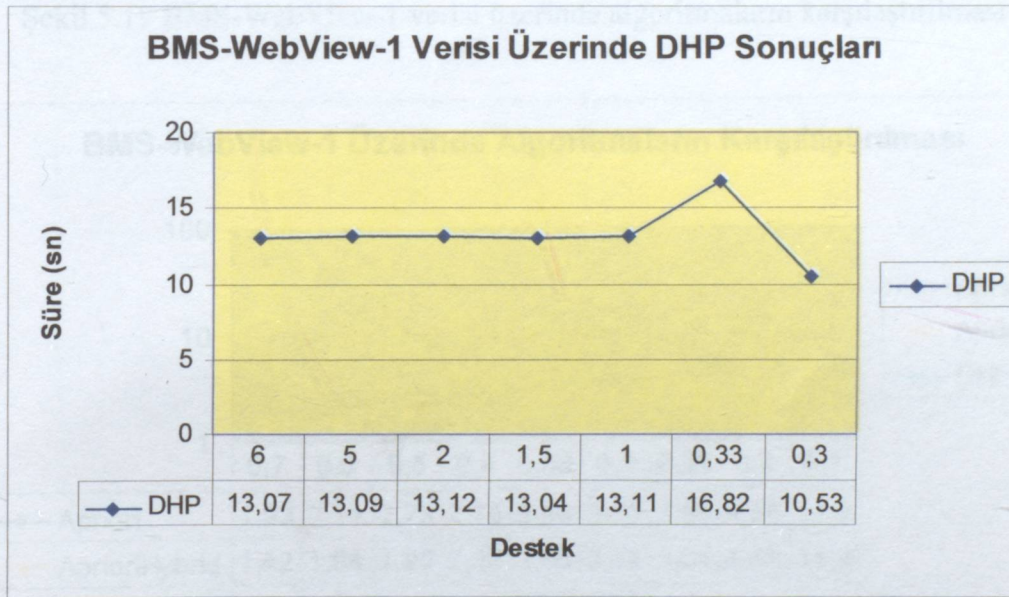
Şekil 5.17 BMS-WebView-1 verisi üzerinde apriorihybrid sonuçları

AprioriHybrid algoritmasının BMS-WebView-1 üzerindeki sonuçları Apriori'ye göre çok iyi değildir. Burada da destek değeri arttıkça AprioriHybrid daha da hızlanmıştır. Bunun nedeni sınır değerinin doğru olarak seçilmesine de bağlıdır. Daha önce anlatıldığı gibi Apriori'den

AprioriTid'e geçiş yapılacak sınır değerinin doğru olarak belirlenememesi, AprioriHybrid'in Apriori'den daha kötü performans göstermesine yol açmaktadır.

Çizelge 5.22 BMS-WebView-1 verisi üzerinde DHP sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kelem Sayısı
225	0,3	10,53	435
211	0,33	16,82	377
67	1	13,11	77
34	1,5	13,04	36
21	2	14,09	22
4	5	13,09	4
3	6	13,07	3



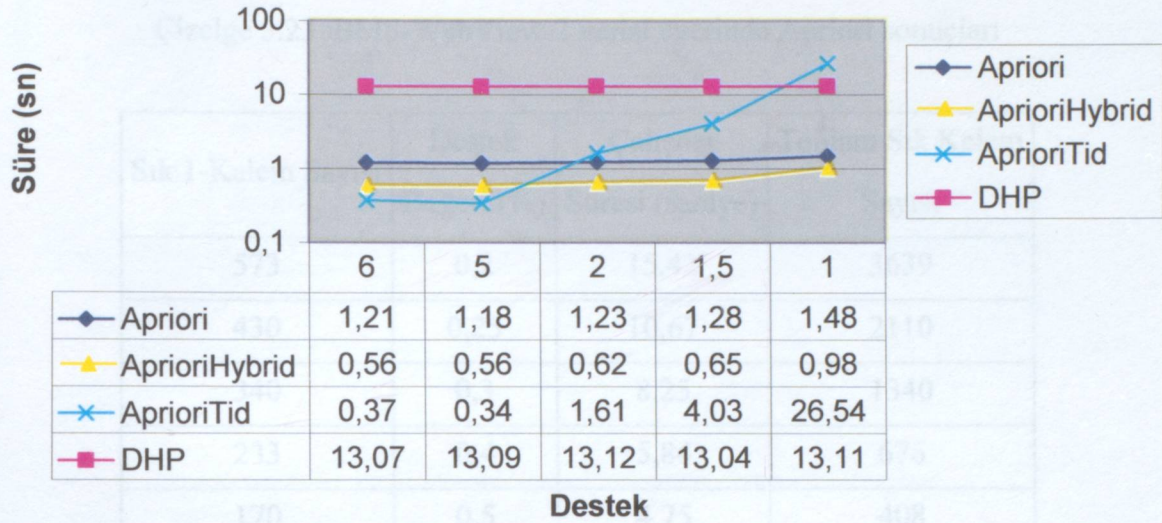
Şekil 5.18 BMS-WebView-1 verisi üzerinde DHP sonuçları

Çizelge 5.22'de DHP'nin BMS-WebView-1'deki sonuçları yer almaktadır. DHP algoritması Akköprü verisinde Apriori'ye göre daha hızlı iken BMS-WebView-1 verisinde oldukça yavaşlamıştır. Bunun nedeni şudur:

“DHP possfreşize adlı ek bir HDA kontrol tablosu kullanır ve bu tablonun boyutunun büyük seçilmesi performans açısından şarttır. Ama BMS-WebView-1 verisinde işlem sayısı çok

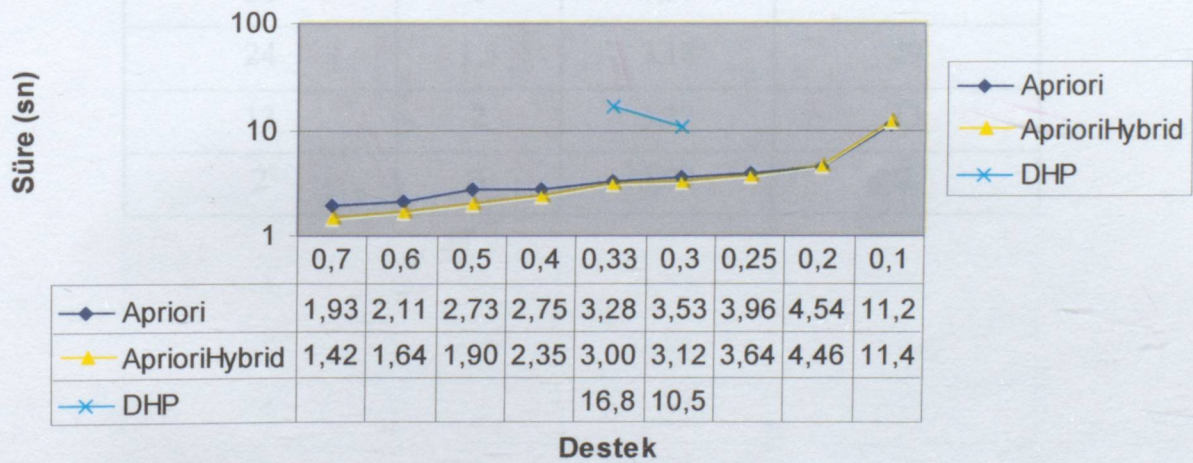
fazla olduğu için bellek fazla kullanılır ve dolayısıyla bu boyut da yeteri kadar büyük seçilemez. Sonuç olarak da hızda düşüş olur.”

BMS-WebView-1 Üzerinde Algoritmaların Karşılaştırılması



Şekil 5.19 BMS-WebView-1 verisi üzerinde algoritmaların karşılaştırılması

BMS-WebView-1 Üzerinde Algoritmaların Karşılaştırılması



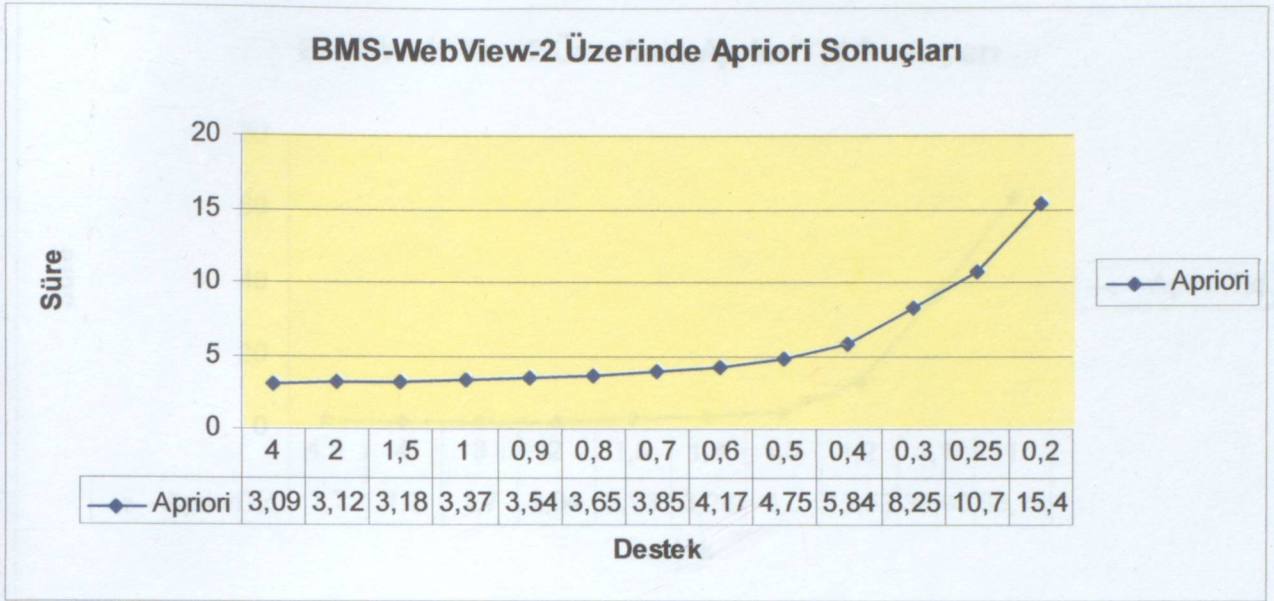
Şekil 5.20 BMS-WebView-1 verisi üzerinde algoritmaların karşılaştırılması

5.3.2.3 BMS-WebView-2 Verisi Üzerinde Performans Farklılıklarına Göre Karşılaştırma

Çizelge 5.23 ve şekil 5.21’de Apriori algoritmasının BMS-WebView-2 verisi üzerindeki sonuçları gösterilmiştir.

Çizelge 5.23 BMS-WebView-2 verisi üzerinde Apriori sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kalem Sayısı
573	0,2	15,42	3639
430	0,25	10,67	2110
340	0,3	8,25	1340
233	0,4	5,84	676
170	0,5	4,75	408
127	0,6	4,17	257
104	0,7	3,85	187
86	0,8	3,65	138
75	0,9	3,54	113
56	1	3,37	81
24	1,5	3,18	29
13	2	3,12	13
2	4	3,09	2



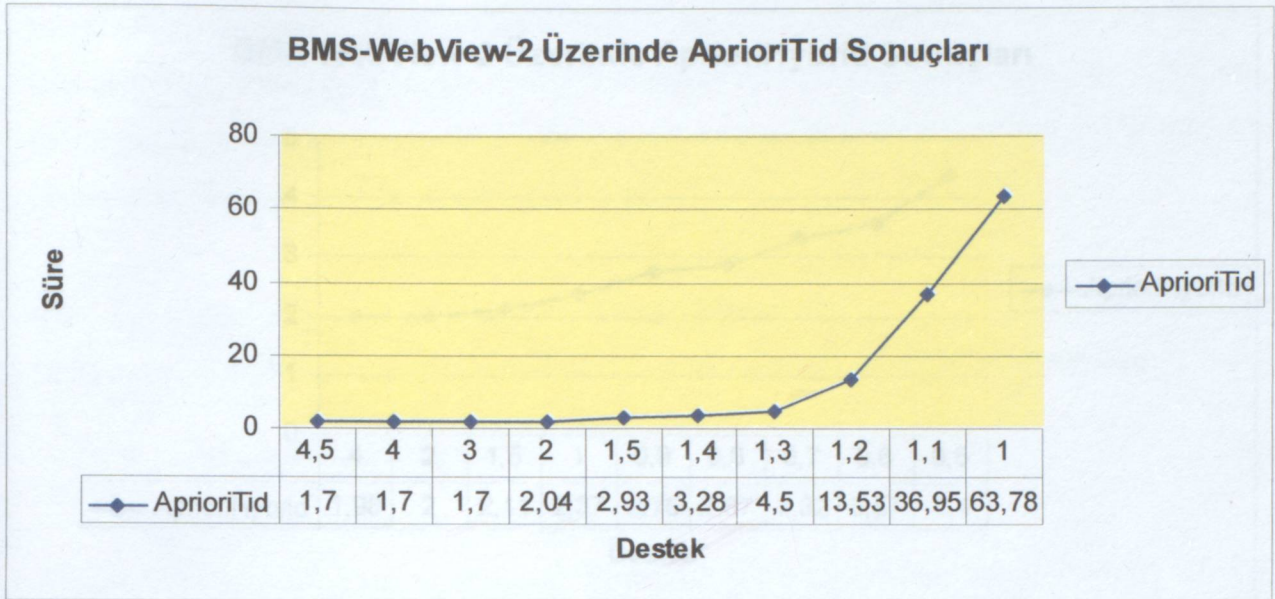
Şekil 5.22 BMS-WebView-2 üzerinde AprioriTid sonuçları

Şekil 5.21 BMS-WebView-2 üzerinde Apriori sonuçları

Çizelge 5.24 ve şekil 5.22’de AprioriTid algoritmasının BMS-WebView-2 verisi üzerindeki sonuçları gösterilmiştir:

Çizelge 5.24 BMS-WebView-2 verisi üzerinde AprioriTid sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kelem Sayısı
56	1	63,78	81
50	1,1	36,95	69
44	1,2	13,53	57
35	1,3	4,50	46
27	1,4	3,28	33
24	1,5	2,93	29
13	2	2,04	13
3	3	1,70	3
2	4	1,70	2
1	4,5	1,70	1

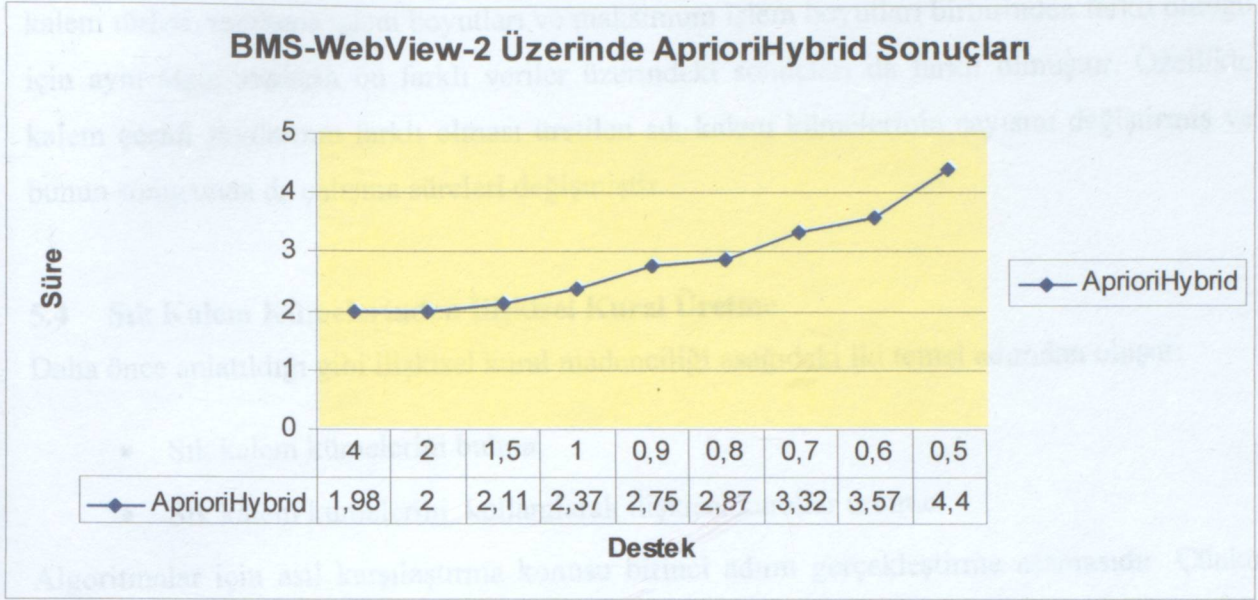


Şekil 5.22 BMS-WebView-2 üzerinde AprioriTid sonuçları

Çizelge 5.25 ve şekil 5.23’de AprioriHybrid algoritmasının BMS-WebView-2 verisi üzerindeki sonuçları gösterilmiştir:

Çizelge 5.25 BMS-WebView-2 Verisi Üzerinde AprioriHybrid Sonuçları

Sık 1-Kalem Sayısı	Destek Değeri (%)	Çalışma Süresi (saniye)	Toplam Sık Kelem Sayısı
170	0,5	4,40	408
127	0,6	3,57	257
104	0,7	3,32	187
86	0,8	2,87	138
75	0,9	2,75	113
56	1	2,37	81
24	1,5	2,11	29
13	2	2	13
2	4	1,98	2



Şekil 5.23 BMS-WebView-2 üzerinde AprioriHybrid sonuçları

5.3.3 Sık Kalem Kümelerine Göre Karşılaştırma

Algoritmaların tümü aynı aday oluşturma prosedürünü kullandığı için, aynı sık kalem kümelerini oluştururlar. Yukarıdaki tabloların sonuçlarına göre aynı veri kümeleri üzerinde aynı destek değerleri için aynı sayıda sık kalem kümeleri oluşturulmuştur. Ancak burada incelenen algoritmaların dışında kalan algoritmalar için aynı şey söylenemez. Çünkü onlar daha farklı sayıda sık kalem kümeleri üretebilir.

5.3.4 Eşik Destek Değerlerine Göre Karşılaştırma

Yukarıda anlatıldığı gibi destek değerine göre çalışma süresi ve üretilen sık kalem kümelerinin sayısı değişmektedir. Destek değeri ne kadar küçük seçilirse, oluşturulacak olan sık kalem kümelerinin sayısı da fazla olur. Her algoritma her destek değerinde çalışmaz. Bu nedenle destek değerinin belirlenmesi de önemli bir konudur. Algoritmalar belleğin yeterli olduğu destek değerleri için sonuç üretmektedir. Örneğin AprioriTid algoritması belleği çok fazla kullanan bir algoritma olduğu için Apriori ve diğer algoritmalar gibi çok düşük destek değerlerinde sonuç vermemektedir. Diğer algoritmalarda da aynı durum söz konusudur. Belleği çok kullanan algoritmalarından biri olan DHP algoritması da, diğer algoritmaların sık kalem kümesi ürettiği bazı destek değerleri için sonuç vermemektedir.

5.3.5 Farklı Veri Kümeleri Üzerinde Karşılaştırma

Bu tez çalışmasında daha önce özellikleri anlatılan üç farklı veri kümesi üzerinde çalışılmış ve her veri kümesi üzerinde farklı sonuçlar elde edilmiştir. Veri kümelerinin işlem sayıları, kalem türleri, ortalama işlem boyutları ve maksimum işlem boyutları birbirinden farklı olduğu

kalem türleri, ortalama işlem boyutları ve maksimum işlem boyutları birbirinden farklı olduğu için aynı algoritmaların bu farklı veriler üzerindeki sonuçları da farklı olmuştur. Özellikle kalem çeşidi sayılarının farklı olması üretilen sık kalem kümelerinin sayısını değiştirmiş ve bunun sonucunda da çalışma süreleri değişmiştir.

5.4 Sık Kalem Kümelerinden İlişkisel Kural Üretme

Daha önce anlatıldığı gibi ilişkisel kural madenciliği aşağıdaki iki temel adımdan oluşur:

- Sık kalem kümelerini bulma,
- Sık kalem kümelerini kullanarak ilişkisel kurallar üretme.

Algoritmalar için asıl karşılaştırma konusu birinci adımı gerçekleştirme aşamasıdır. Çünkü ikinci adımı ortak bir prosedürle yapabilirler. Birinci adımdan elde edilen sık kalem kümeleri bu prosedüre girdi olarak verilir ve ilişkisel kurallar elde edilir.

İncelediğimiz ve uygulamasını gerçekleştirdiğimiz dört algoritmanın aynı veri kümesi üzerinde ürettikleri sık kalem kümeleri aynı olduğundan ilişkisel kuralları da aynıdır. Bu nedenle tek bir ilişkisel kural üretme algoritması bizi amacımıza ulaştırır. Aşağıda ilişkisel kural üretme algoritmasının pseudo kodu bulunmaktadır:

Çizelge 5.26 İlişkisel kural üretme algoritması

```

1. forall large k-itemset  $I_k$ ,  $k \geq 2$  do begin
2.    $H_1 = \{\text{consequents of rules derived from } I_k \text{ with one item in the consequent}\};$ 
3.   call ap-genrules ( $I_k$ ,  $H_1$ )
4. end

procedure ap-genrules ( $I_k$ : large k-itemset,  $H_m$  : set of m-item consequents)
  if ( $k > m+1$ ) then begin
     $H_{m+1} = \text{apriori-gen}(H_m);$ 
    forall  $h_{m+1} \in H_{m+1}$  do begin
       $\text{Conf} = \text{support}(I_k) / \text{support}(I_k - h_{m+1});$ 
      if ( $\text{conf} \geq \text{minconf}$ ) then
        Output the rule  $(I_k - h_{m+1}) \rightarrow h_{m+1}$  with confidence = conf and support support( $I_k$ );
      Else
        Delete  $h_{m+1}$  from  $H_{m+1}$ ;
    End
    Call ap-genrules ( $I_k$ ,  $H_{m+1}$ );
  end

```

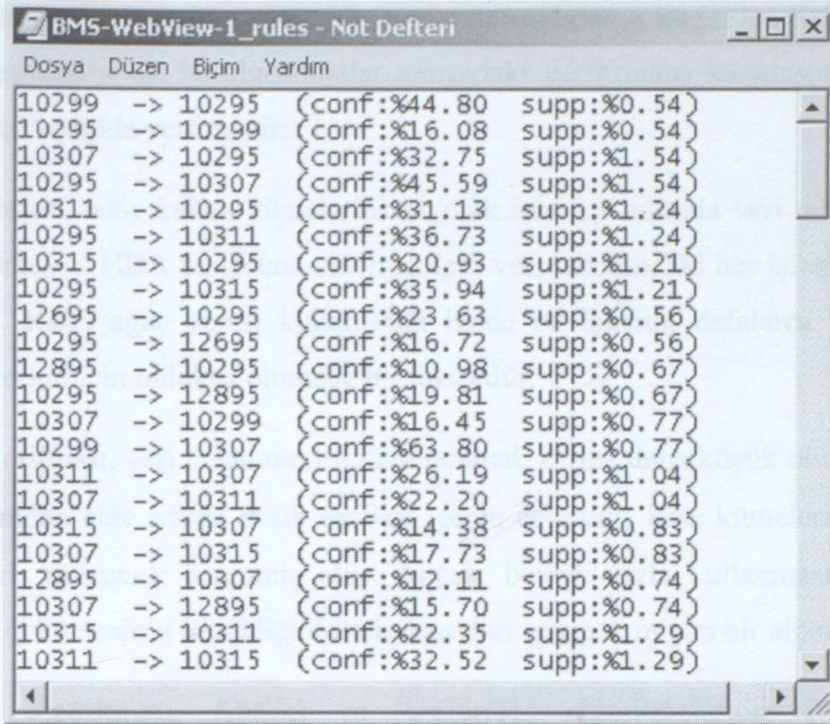
Bu çalışmadaki dört algoritmanın ürettiği ilişkisel kurallar dosya formatında kaydedilmiştir. Akköprü ve BMS-WebView-1 verilerinden elde edilen sonuçların bir kısmı aşağıdaki şekillerde gösterilmiştir. Bu şekillerde görülen $a \rightarrow b$ biçimindeki kurallarda, a ve b ürün kodlarına karşılık gelmektedir. Örneğin $10299 \rightarrow 10295$ (conf:% 44.80 , supp:% 0.54) kuralına göre, 10299 kodlu ürününü satın alan kişiler 10295 kodlu ürünü % 44.80 güven ve % 0.54 destek değerleriyle satın almaktadırlar.

apriori_rules - Not Defteri			
Dosya	Düzen	Biçim	Yardım
291	->	273	(conf:%10.99 supp:%0.55)
273	->	291	(conf:%22.03 supp:%0.55)
273	->	51420	(conf:%25.87 supp:%0.65)
274	->	51420	(conf:%28.23 supp:%0.52)
291	->	283	(conf:%11.17 supp:%0.56)
283	->	291	(conf:%16.62 supp:%0.56)
283	->	1069	(conf:%17.14 supp:%0.58)
283	->	1075	(conf:%15.32 supp:%0.52)
283	->	48435	(conf:%15.32 supp:%0.52)
283	->	51420	(conf:%24.94 supp:%0.84)
307	->	291	(conf:%19.24 supp:%0.58)
291	->	307	(conf:%11.52 supp:%0.58)
403	->	291	(conf:%18.05 supp:%0.63)
291	->	403	(conf:%12.57 supp:%0.63)
934	->	291	(conf:%13.79 supp:%0.90)
291	->	934	(conf:%17.98 supp:%0.90)
997	->	291	(conf:%13.29 supp:%0.59)
291	->	997	(conf:%11.69 supp:%0.59)
291	->	1069	(conf:%17.98 supp:%0.90)
1075	->	291	(conf:%10.22 supp:%0.83)
291	->	1075	(conf:%16.58 supp:%0.83)
291	->	1201	(conf:%11.17 supp:%0.56)

Şekil 5.24 Akköprü verisinden elde edilen ilişkisel kurallar

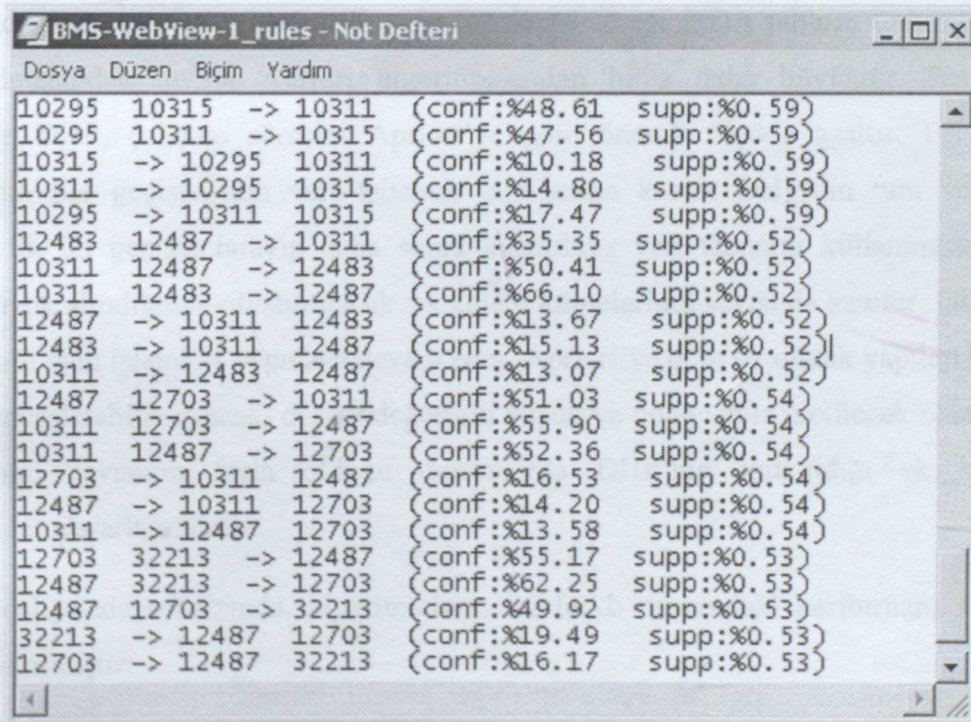
apriori_rules - Not Defteri			
Dosya	Düzen	Biçim	Yardım
1069	1075	1387	-> 1093 (conf:%40.38 supp:%0.55)
1069	1075	1093	-> 1387 (conf:%39.38 supp:%0.55)
1093	1387	->	1069 1075 (conf:%41.45 supp:%0.55)
1075	1387	->	1069 1093 (conf:%32.81 supp:%0.55)
1075	1093	->	1069 1387 (conf:%33.33 supp:%0.55)
1069	1387	->	1075 1093 (conf:%26.25 supp:%0.55)
1069	1093	->	1075 1387 (conf:%25.30 supp:%0.55)
1387	->	1069 1075 1093	(conf:%14.19 supp:%0.55)
1093	->	1069 1075 1387	(conf:%12.96 supp:%0.55)
1075	1099	1384	-> 1069 (conf:%90.62 supp:%0.51)
1069	1099	1384	-> 1075 (conf:%67.44 supp:%0.51)
1069	1075	1384	-> 1099 (conf:%26.01 supp:%0.51)
1069	1075	1099	-> 1384 (conf:%59.18 supp:%0.51)
1099	1384	->	1069 1075 (conf:%47.54 supp:%0.51)
1075	1384	->	1069 1099 (conf:%20.42 supp:%0.51)
1075	1099	->	1069 1384 (conf:%49.15 supp:%0.51)
1069	1384	->	1075 1099 (conf:%15.22 supp:%0.51)
1069	1099	->	1075 1384 (conf:%33.72 supp:%0.51)
1075	1120	1384	-> 1069 (conf:%85.51 supp:%0.52)
1069	1120	1384	-> 1075 (conf:%72.84 supp:%0.52)
1069	1075	1384	-> 1120 (conf:%26.46 supp:%0.52)
1069	1075	1120	-> 1384 (conf:%62.11 supp:%0.52)

Şekil 5.25 Akköprü verisinden elde edilen diğer ilişkisel kurallar



Dosya	Düzen	Biçim	Yardım
10299	->	10295	(conf:%44.80 supp:%0.54)
10295	->	10299	(conf:%16.08 supp:%0.54)
10307	->	10295	(conf:%32.75 supp:%1.54)
10295	->	10307	(conf:%45.59 supp:%1.54)
10311	->	10295	(conf:%31.13 supp:%1.24)
10295	->	10311	(conf:%36.73 supp:%1.24)
10315	->	10295	(conf:%20.93 supp:%1.21)
10295	->	10315	(conf:%35.94 supp:%1.21)
12695	->	10295	(conf:%23.63 supp:%0.56)
10295	->	12695	(conf:%16.72 supp:%0.56)
12895	->	10295	(conf:%10.98 supp:%0.67)
10295	->	12895	(conf:%19.81 supp:%0.67)
10307	->	10299	(conf:%16.45 supp:%0.77)
10299	->	10307	(conf:%63.80 supp:%0.77)
10311	->	10307	(conf:%26.19 supp:%1.04)
10307	->	10311	(conf:%22.20 supp:%1.04)
10315	->	10307	(conf:%14.38 supp:%0.83)
10307	->	10315	(conf:%17.73 supp:%0.83)
12895	->	10307	(conf:%12.11 supp:%0.74)
10307	->	12895	(conf:%15.70 supp:%0.74)
10315	->	10311	(conf:%22.35 supp:%1.29)
10311	->	10315	(conf:%32.52 supp:%1.29)

Şekil 5.26 BMS-WebView-1 verisinden elde edilen ilişkisel kurallar



Dosya	Düzen	Biçim	Yardım
10295	10315	->	10311 (conf:%48.61 supp:%0.59)
10295	10311	->	10315 (conf:%47.56 supp:%0.59)
10315	->	10295	10311 (conf:%10.18 supp:%0.59)
10311	->	10295	10315 (conf:%14.80 supp:%0.59)
10295	->	10311	10315 (conf:%17.47 supp:%0.59)
12483	12487	->	10311 (conf:%35.35 supp:%0.52)
10311	12487	->	12483 (conf:%50.41 supp:%0.52)
10311	12483	->	12487 (conf:%66.10 supp:%0.52)
12487	->	10311	12483 (conf:%13.67 supp:%0.52)
12483	->	10311	12487 (conf:%15.13 supp:%0.52)
10311	->	12483	12487 (conf:%13.07 supp:%0.52)
12487	12703	->	10311 (conf:%51.03 supp:%0.54)
10311	12703	->	12487 (conf:%55.90 supp:%0.54)
10311	12487	->	12703 (conf:%52.36 supp:%0.54)
12703	->	10311	12487 (conf:%16.53 supp:%0.54)
12487	->	10311	12703 (conf:%14.20 supp:%0.54)
10311	->	12487	12703 (conf:%13.58 supp:%0.54)
12703	32213	->	12487 (conf:%55.17 supp:%0.53)
12487	32213	->	12703 (conf:%62.25 supp:%0.53)
12487	12703	->	32213 (conf:%49.92 supp:%0.53)
32213	->	12487	12703 (conf:%19.49 supp:%0.53)
12703	->	12487	32213 (conf:%16.17 supp:%0.53)

Şekil 5.27 BMS-WebView-1 verisinden elde edilen diğer ilişkisel kurallar

5.5 Uygulanan Algoritmaların Karşılaştırma Sonuçlarının Değerlendirilmesi

Kullanılan algoritmalar ve bu algoritmalar arasındaki performans karşılaştırmaları ile ilgili değerlendirmeler aşağıda verilmiştir:

Apriori algoritması, aday kalem kümelerini saymak için her adımda veri tabanını tarar. Her aday kalem kümesini HDA tablosuna eklemek için veri tabanındaki her işlemi kontrol etmek gerekmektedir. HDA ağaç yapısı kullanıldığı halde bu işlemin defalarca tekrar edilmesi, algoritmanın verimi için oldukça olumsuz bir faktördür.

AprioriTid algoritması, veri tabanının tümünü taramak yerine daha küçük olması beklenen ve bir önceki adımdan elde edilen σ -sık cas'ları içeren counting_base kümelerini taramaktadır. Böylece tekrarlı taramalar önlenmiş olur. Ancak, belleği fazla kullanması sebebiyle, çok sayıda sık kalem kümesinin üretildiği destek değerleri için pek uygun bir algoritma değildir.

AprioriHybrid algoritması, Apriori ve AprioriTid algoritmalarının iyi özelliklerinin birleştirilmesiyle elde edilmiştir. Bu nedenle iki algoritma arasındaki geçişi belirleyen sınır parametresinin doğru seçildiği durumlarda her ikisine göre daha iyi performans gösterir.

DHP algoritmasının veri tabanını ilk seferde tarama süresi, HDA tablosu üretimi için ekstra taşıma olmasından dolayı Apriori algoritmasından biraz daha büyüktür. Fakat sonraki geçişlerde DHP, çalışma süresini Apriori'ye göre önemli ölçüde azaltır. Bunun nedeni Apriori'nin her geçişte tüm veri tabanını taramasına karşın DHP'nin tüm veri tabanını yalnızca ilk iki geçişte tarayıp daha sonra kısaltılmış veri tabanını kullanmasıdır. Destek değeri büyük alındığında oluşturulacak sık kalem kümelerinin sayısı da az olur, fakat DHP ek kontrollerle aynı işlemleri yapmaya devam eder. Apriori'ye göre ek olarak yapılan bu işlemler sistemi yavaşlatabilir. Ancak, destek değerinin küçük ve dolayısıyla üretilecek olan sık kalem kümelerinin sayısının fazla olduğu durumlarda DHP'nin kullandığı ek kontrollerin avantajından yararlanılabilir.

Yapılan çalışmada sonucunda, algoritmaların iki farklı durumdaki performans sıralamaları aşağıda verilmiştir:

- **Düşük destek değerleri için:** DHP > AprioriHybrid > Apriori > AprioriTid
- **Yüksek destek değerleri için:** AprioriTid > AprioriHybrid > Apriori > DHP

Yukarıdaki performans sıralamaları algoritmaların her iki veri kümesi üzerinde verdikleri sonuçlardan elde edilmiştir. Buna göre DHP algoritması düşük destek değerlerinde en iyi performansı gösterirken, yüksek destek değerlerinde AprioriTid en iyi performansı

göstermiştir. DHP'nin düşük destek değerlerinde en hızlı olmasının nedeni veri tabanını kısaltması ve tarama yapacağı alanı daraltmasıdır. AprioriTid'in yüksek destek değerlerinde en hızlı olmasının nedeni ise veri tabanını bir kez taradıktan sonra ek bir küme kullanması ve tekrarlı taramaları ortadan kaldırmasıdır. AprioriHybrid algoritması ise Apriori ve AprioriTid algoritmalarının iyi özelliklerinin birleştirilmesiyle elde edildiği için onlara göre genel olarak daha iyi performans göstermiştir. Fakat çok yüksek destek değerlerinde AprioriTid'e göre daha yavaş olduğu görülmüştür. Bunun nedeni iki algoritma arasındaki geçiş maliyetinin yüksek olmasıdır.

Sonuç olarak destek değeri küçüldükçe algoritmaların çalışma süreleri ve ürettikleri sık kalem kümelerinin sayısı artar. Bunun sonucunda da algoritmaların çalışma süreleri arasındaki fark artar. Fakat her destek değeri için algoritmaların ürettikleri sık kalem kümelerinin sayısı aynıdır.

6. SONUÇ VE ÖNERİLER

Bu çalışmanın araştırma konusu olan ilişkisel kural madenciliği, veri tabanlarında bir arada görülen ilişkileri ortaya çıkaran ve özetleyen ilişkisel kuralları bulmayı amaçlayan bir tekniktir. Bir alışveriş sırasında veya birbirini izleyen alışverişlerde müşterinin hangi ürün veya hizmetleri satın almaya eğilimli olduğunun belirlenmesi, müşteriye daha fazla ürünün satılmasını sağlama yollarından biridir. Bu nedenle ilişkisel kural madenciliği son yılların en çok araştırılan konularından biri olmuştur. Bu amaçla bir çok algoritma geliştirilmiş ve daha iyi performans elde etmek için var olan algoritmalarından yeni algoritmalar türetilmiştir. Bu konudaki en önemli algoritmalarından biri de, bir çok algoritma için de temel teşkil eden Apriori algoritmasıdır.

Bu çalışmada öncelikle, veri madenciliği konusunda ön araştırmalar yapılarak; veri madenciliği teknikleri, veri ambarı ve OLAP kavramları ikinci bölümde ayrıntılı olarak anlatılmıştır.

Bu çalışmanın temeli olan ilişkisel kural madenciliği konusu üçüncü bölümde anlatılmıştır. Ayrıca, algoritmaların uygulanması esnasında gerekli olan destek, güven, aday kalem kümesi, sık kalem kümesi, arama alanı ve ilişkisel kural gibi temel kavramlara da bu bölümde ayrıntılı olarak yer verilmiştir.

Dördüncü bölümde, Apriori algoritmasına benzemeyen ve farklı veri yapıları kullanan FP-Growth, Charm, Closet, Eclat, Partioning algoritmaları, kullandıkları yöntem ve yapılara göre incelenmiştir.

Temel olarak Apriori algoritmasının ele alındığı bu çalışmada, ilişkisel kural madenciliği algoritmaları incelenerek, Apriori ve ona benzeyen AprioriTid, AprioriHybrid ve DHP algoritmalarının; yöntem, performans, üretilen sık kalem kümesi sayısı ve destek eşik değeri farklılıklarına göre, algoritmaların uygulandığı bir program vasıtasıyla, farklı veri kümeleri üzerinde karşılaştırmaları yapılmıştır. Uygulanan algoritmalar için HDA veri yapısı kullanılmıştır. Bu algoritmalar, kullanılan farklı veri ambarları ve uygulamaların sonuçları beşinci bölümde ayrıntılı olarak anlatılmıştır.

Bu algoritmalar, C programlama dili ile programlanarak, CPU hızı Pentium IV 1.5 GHz ve RAM kapasitesi 512 MB olan bir bilgisayar ile, Windows 2000 Professional işletim sistemi üzerinde test edilmiştir.

Elde edilen deneyimler ışığında bu çalışmayı daha ileri bir seviyeye taşımak ve burada incelenen algoritmaların performanslarını daha da arttırabilmek için ileride yapılması hedeflenen çalışmalar aşağıda listelenmiştir:

- Farklı veri yapılarının uygulanması ile algoritmaların performansları arasındaki değişimler gözlemlenebilir.
- Elde edilen sık kalem kümelerinin sayısını azaltmak için DHP’de yapılan bazı ek kontrollerin, oluşturulan sık kalem kümelerinin sayısında fazla bir değişiklik yapmadığı gözlemlenmiştir. Bunların kaldırılması hem maliyeti hem de algoritmanın performansını olumlu etkileyecektir.
- Algoritmalar, veri üreteçleri ile rastgele üretilen veri kaynakları üzerinde de uygulanabilir ve buradan elde edilecek sonuçlar ile gerçek dünya verisi üzerinden elde edilen sonuçlar karşılaştırılabilir.
- Apriori’den bağımsız olan diğer algoritmaların da kullanıldığı bir program yazılarak yeni karşılaştırmalar yapılabilir. Hepsinin iyi özellikleri birleştirilerek yeni ve güçlü algoritmalar üretilebilir.

KAYNAKLAR

- Adamo, J., (2000), "Data Mining for Association Rules and Sequential Patterns".
- Aggarwal, C.C., Yu, P.S., (1998), "Mining Large Itemsets For Association Rules", Bulletin Of The IEEE Computer Society Technical Committee On Data Engineering.
- Agrawal, R., Imielinski, T., Swami, A., (1993), "Mining Association Rules between Sets of Items in Large Databases", ACM-SIGMOD Konferansı, Washington, DC.
- Agrawal, R., Srikant, R., (1994), "Fast Algorithms for Mining Association Rules", 20th Int. Conf. Very Large Data Bases, VLDB.
- Agrawal, R., Srikant, R., (1995), "Mining Sequential Patterns", IBM Almaden Research Center, San Jose.
- Alpaydın, E. (2000), "Zeki Veri Madenciliği", Boğaziçi Üniversitesi Bilgisayar Mühendisliği Bölümü, İstanbul.
- Borgelt, C., Kruse, R., (1996), "Induction Of Association Rules: Apriori Implementation", Department Of Knowledge Processing And Language Engineering School Of Computer Science Otto-Von-Guericke-University Of Magdeburg, Germany.
- Cengiz, İ., "Mining Association Rules", Bilkent University, Department Of Computer Engineering & Information Sciences, Ankara.
- Craw, S., Wiratunga, N., (2000), "Applications Of Data Mining-Association Rules", Ref: Written & Frank, 2000, Chapters 4 and 8.
- Goethals, B., (2003), "Survey on Frequent Pattern Mining" HIIT Basic Research Unit, Department of Computer Science, University of Helsinki, P.O. box 26, FIN-00014 Helsinki, Finland.
- Guo, W., "Fp-Growth Algorithm With Candidate Generation And Its Application In Quantitative Association Rules Mining", Computer Science Department North Dakota State University.
- Han, J., Pei, J., Yin, Y., (1999), "Mining Frequent Patterns without Candidate Generation" 2000 ACM SIGMOD Intl. Conference on Management of Data.
- Hipp, J., Güntzer, U., Nakhaeizadeh, G., (2000), "Algorithms for Association Rule Mining- A General Survey and Comparison", 2000 ACM SIGKDD.
- Houtsma, M., Swami, A., (1995), "Set-oriented mining for association rules in relational databases", Int'l. Conference on Data Engineering, Taipei, Taiwan, pp. 25-33.
- Inmon W.H., (1996), "Building the Data Warehouse, J.W & Sons".
- Inmon, W.H., Welch, J.D., Glassey, K.L., (1997), "Managing the Data Warehouse, J. W. & Sons".
- Mannila, H., (1997), "Methods and Problems in Data Mining", Department Of Computer Science University Of Helsinki, Finland.
- Netz, A., Chaudhuri, S., Bernhardt, J., Feyyad, U., (2000), "Integration Of Data Mining And Relational Databases", 26th International Conference On Very Large Databases, Cairo, Egypt.

Özel, S.A., Güvenir, H.A., "An Algorithm For Mining Association Rules Using Perfect Hashing And Database Pruning", Bilkent University, Department Of Computer Engineering, Ankara.

Park, J.S., Chen, M.S., Yu, P.S., (1995), "An Effective Hash-Based Algorithm For Mining Association Rules", Ibm Thomas J. Watson Research Center Yorktown Heights, New York.

Pei, J., Han, J., Mao, R., (2000), "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.

Seno, M., Karypis, G., (2001), "LPMiner: An Algorithm For Finding Frequent Itemsets Using Length-Decreasing Support Constraint", Department Of Computer Science and Engineering, Army HPC Research Center University of Minnesota.

Wijzen, J., Meersman, R., (1997), "On The Complexity of Mining Quantitative Association Rules", Department Informatica, Vrije Universiteit Brussel, Belgium.

Wijzen, J., Meersman, R., (1998), "On The Complexity of Mining Quantitative Association Rules" 1-20 () Kluwer Academic Publishers, Boston. Manufactured in Netherlands, The Data Mining and Knowledge Discovery.

Zaki, M.J., Charm, C.H., (1999), "An Efficient Algorithm for Closed Association Rule Mining.", In Technical Report 99-10, Computer Science, Rensselaer Polytechnic Institute.

Zaki, M.J., (2000), "Scalable Algorithms For Association Mining", IEEE Transactions On Knowledge Data Engineering.

Zaki, M.J., Parthasarathy, S., Ogihara, M., Li, W., (1997), "New Algorithms For Fast Discovery Of Association Rules", The University Of Rochester Computer Science Department Rochester, New York.

Zhang, C., Zhang, S., (2002), "Association Rule Mining Models and Algorithms", Springer-Verlag Berlin Heidelberg.

Zheng, Z., Kohavi, R., Mason, L., (2001), "Real World Performance of Association Rule Algorithms", 7th ACM-SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, New York.

INTERNET KAYNAKLARI

[1]<http://www.cmpe.boun.edu.tr/~ethem>

[2]<http://idari.cu.edu.tr/sempozyum/bil38.htm>

[3]<http://fuzzy.cs.uni-magdeburg.de/~borgelt/software.html>

[4]midas.cs.ndsu.nodak.edu/~ding/datamining/qin_ding.doc

[5]<http://www.cs.bu.edu/faculty/gkollios/ada02/LectNotes/assoc1.ppt>

[6]<http://www.cs.bu.edu/faculty/gkollios/ada02/LectNotes/assoc2.ppt>

[7]http://ir.iit.edu/~dagr/DataMiningCourse/Spring2001/Presentations/Summary_03.pdf

[8]http://www.bilgiyonetimi.org/cm/pages/mkl_gos.php?nt=131

ÖZGEÇMİŞ

Doğum tarihi 04.04.1979

Doğum yeri Rize

Lise 1993-1997 Rize Fener Süper Lisesi

Lisans 1997-2001 İstanbul Üniversitesi Mühendislik Fak.
Bilgisayar Mühendisliği Bölümü

Yüksek Lisans 2002-2004 Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Bölümü

Çalıştığı kurum(lar)

2000-2001 Vestelnet Yazılım Şti.

2002-2003 YTÜ Fen Bilimleri Enstitüsü Araştırma Görevlisi

2003-Devam ediyor Tepe Teknolojik Servisler A.Ş.

