



T.C.

SELÇUK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİ SİSTEMLERİNDE FARK FONKSİYONU TABANLI ÖZELLİK SEÇME YÖNTEMİNİN GELİŞTİRİLMESİ

MEHMET HACIBEYOĞLU

DOKTORA TEZİ

Bilgisayar Mühendisliği Anabilim Dalını

Mart-2012
KONYA
Her Hakkı Saklıdır

ÖZET

DOKTORA TEZİ

BİLGİ SİSTEMLERİNDE FARK FONKSİYONU TABANLI ÖZELLİK SEÇME YÖNTEMİNİN GELİŞTİRİLMESİ

MEHMET HACIBEYOĞLU

Selçuk Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Ahmet ARSLAN

2012, 94 Sayfa

Jüri
Prof. Dr. Ahmet ARSLAN
Prof. Dr. Şirzat KAHRAMANLI
Doç. Dr. Mehmet ÇUNKAŞ
Yrd. Doç. Dr. Mesut GÜNDÜZ
Yrd. Doç. Dr. Gülay TEZEL

Bilgi sistemlerinde özellik seçmenin amacı şart özelliklerinin orijinal kümesi ile aynı sınıflandırma başarısı sağlayan özelliklerin bir veya birkaç tane minimal alt kümesini bulmaktır. Genellikle bir veri kümesi özelliklerin birden fazla minimal alt kümelerine sahip olabilir ve bunların hepsini bulmak bir NP-hard (belirsiz polinomal-zor) problemdir. Bu yüzden bir veri kümesine ait olan özelliklerin bir veya birkaç minimal alt kümesini bulan sezgisel algoritmalar geliştirilmiştir. Fakat bu algoritmaları kullanmak en iyi çözümü kaçırma riskini de beraberinde getirmektedir. Çünkü özelliklerin bulunan alt kümesi bazen en iyi küme olmayabilir. Bu yüzden, özelliklerin en iyi alt kümesini bulmak istendiğinde bu iş için yegâne olan fark fonksiyonu tabanlı özellik seçme yaklaşımı kullanılır. Fakat maalesef bu yaklaşıma dayanan algoritmalar bellek taşmasından dolayı işini bitirmeden sonlanırlar. Bu algoritmalar için bellek taşmasının sebebi fark fonksiyonun disjunktif normal forma çevrilirken üstel olarak artan hafıza karmaşıklığıdır. Bu yüzden, bu çalışmada, disjunktif normal formun orijinal fark fonksiyonundan değil indirgenmiş fark fonksiyonundan elde edilmesi yöntemi geliştirilmiş ve bu yolla karmaşıklık kendi karekökünе kadar azaltılmıştır. Böylece, iki aşamadan oluşan lojik fonksiyon tabanlı bir özellik seçme yöntemi geliştirilmiştir. Birinci aşamada veri kümesinin doğruluk tablo görüntüsü kullanılarak indirgenmiş fark fonksiyonu oluşturulur, ikinci aşamada ise elde edilen indirgenmiş fark fonksiyonu iteratif olarak bölünerek disjunktif normal forma çevrilir ve böylece işlenmekte olan veri kümesine ait özelliklerin minimal alt kümeleri elde edilir. Geliştirilen özellik seçme yöntemi bu özelliği sayesinde diğer özellik seçme yöntemleri ile işlenemeyen veri kümelerini de başarı ile işlenebilmektedir.

Anahtar Kelimeler: Bilgi sistemleri, Boole fonksiyonu, fark fonksiyonu, fonksiyonel parçalama, özellik seçme, özellik indirgeme, veri kümeleri.

ABSTRACT

Ph.D THESIS

DEVELOPMENT OF DISCERNIBILITY FUNCTION BASED FEATURE SELECTION METHOD IN INFORMATION SYSTEMS

MEHMET HACIBEYOĞLU

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCE OF SELÇUK UNIVERSITY
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN COMPUTER ENGINEERING

Advisor: Prof. Dr. Ahmet ARSLAN

2012, 94 Pages

Jury
Prof. Dr. Ahmet ARSLAN
Prof. Dr. Şirzat KAHRAMANLI
Assoc. Prof. Dr. Mehmet ÇUNKAŞ
Asst. Prof. Dr. Mesut GÜNDÜZ
Asst. Prof. Dr. Gülay TEZEL

The goal of feature selection in information systems is to find the minimal subset of condition feature set that has the same classification power as the original condition feature set. Usually one dataset may have a lot of minimal subset of attributes and finding all of them is known as an NP-hard problem. Therefore, when only one minimal subset of attribute is required, some heuristic for finding only one or a small number of possible minimal subset of attributes is used. But in this case there is a risk that the best minimal subset of attributes would be overlooked. When the best solution of a feature selection task is required, the discernibility function-based approach, generating all minimal subset of attributes, is used. But unfortunately discernibility function-based feature selection programs fail due to overflow the computer's memory. This is the intractable space complexity of the conversion of a discernibility function to disjunctive normal form. Thus, in this study, we developed a method that obtained disjunctive normal form from minimized discernibility function which the value computational complexity is reduced the square root of the original conversion. Based on these facts, we developed a two-step logic function–based feature selection method. The first stage, developed method derives minimized discernibility function from the truth table image of a dataset. The second stage finds all minimal subset of attributes of a dataset by iteratively partitioning the discernibility function so that the part to be converted to disjunctive normal form. Due to this property, it can process most of datasets that can not be processed by other discernibility function based programs.

Keywords: Attribute reduction, Boole function, datasets, discernibility function, feature selection, functional partitioning, information systems.

ÖNSÖZ

Doktora çalışmamı yapabilmem için, kapılarını bana güvenle açan, her zaman gönülden destekleyen ve bilgi birikimlerini benimle paylaşarak bana yol gösterici olan saygı değer hocalarım Prof. Dr. Ahmet ARSLAN ve Prof. Dr. Şirzat KAHRAMANLI' YA çok teşekkür ederim.

Tezimi hazırlarken her türlü yardımlarını esirgemeyen eksiklerimi görüp tez çalışmamı iyileştirmemi sağlayan Doç. Dr. Mehmet ÇUNKAŞ'a teşekkür ederim.

Bu tezi hazırlamamda bana çok yardımcı olan eşim ve aileme ayrıca bana yeni fikirler vererek tezimi hazırlamamda yardımları dokunan çalışma arkadaşlarıma teşekkür ederim.

Mehmet HACIBEYOĞLU
KONYA-2012

İÇİNDEKİLER

ÖZET ..... **iv**

ABSTRACT ..... **v**

ÖNSÖZ ..... **vi**

İÇİNDEKİLER ..... **vii**

SİMGELER VE KISALTMALAR ..... **ix**

1. GİRİŞ **1**

2. ÖZELLİK SEÇMENİN LİTERATÜRDEKİ YERİ **4**

3. VERİ KÜMELERİNDE ÖZELLİK SEÇME İŞLEMİNE BOOLE FONKSİYON YAKLAŞIMI **9**

3.1. Veri kümelerine Boole fonksiyonu yaklaşımı 9
3.2. Bir veri kümesinin Boole fonksiyonu olarak yorumlanması 9
3.3. Bir veri kümesinin doğruluk tablo görüntüsünün elde edilmesi 11
3.4. Bir veri kümesi için fark fonksiyonunun elde edilmesi 14
3.4.1. Fark fonksiyonunun indirgenmesi 16
3.5. Bir veri kümesinin doğruluk tablo görüntüsünden indirgenmiş fark fonksiyonunun elde edilmesi 16
3.6. Bit tabanlı ifade formunda bulunan indirgenmiş fark fonksiyonunun disjunktif normal forma çevrilmesi ile özelliklerin minimal alt kümelerinin elde edilmesi 25
3.6.1. Bit tabanlı ifadelerin genişletilmesi 25
3.6.2 Bit tabanlı ifadelerden oluşan indirgenmiş fark fonksiyonun disjunktif normal forma çevrilmesi esnasında gereksiz terimlerin oluşmasının engellenmesi 27
3.7. Boole fonksiyon yaklaşımlı özellik seçme yönteminin çok değerli veri kümelerine uyarlanması 34
3.8. Boole fonksiyon yaklaşımlı özellik seçme yöntemi ile yapılan deney sonuçları 49

4. İNDİRGENMİŞ FARK FONKSİYONUNU DİSJUNKTİF NORMAL FORMA ÇEVİRME KARMAŞIKLIĞININ DEĞERLENDİRİLMESİ **54**

4.1. İndirgenmiş fark fonksiyonunun disjunktif normal forma çevrilmesi sürecinde karşılaşılan en kötü hafıza karmaşıklığı 54
4.2. İndirgenmiş fark fonksiyonunun disjunktif normal forma çevrilmesi esnasında oluşan en kötü zaman karmaşıklığı 56

5. İNDİRGENMİŞ FARK FONKSİYONUNUN İTERATİF OLARAK BÖLÜNMESİYLE ÖZELLİKLERİN MİNİMAL ALT KÜMELERİNİN BULUNMASI **59**

5.1. İndirgenmiş fark fonksiyonun böl ve yönet (divide and conquer) stratejisi ile küçük parçalarla çözülecek hale getirilmesi 59
5.2. Fark fonksiyonu matrisi 65

5.3. Fark fonksiyonu matrisinin disjunktif normal forma çevrilmesi 67
5.4. Fark fonksiyonu matrisini parçalayarak özelliklerin minimal alt kümelerinin tamamını elde eden algoritma 68
5.5. ÇEVİR_DFM algoritmasının en kötü hafıza karmaşıklığının hesaplanması 76

6. DENEYSEL SONUÇLAR 78

7. SONUÇ 82

KAYNAKLAR 85

EKLER 90

ÖZGEÇMİŞ 94

SİMGELER VE KISALTMALAR

Simgeler

$\cup$: Birleşme işlemi
$\cap$: Lojik çarpma operatörü
& : Lojik bitsel AND operatörü
| : Lojik bitsel OR operatörü
$\oplus$: Lojik bitsel XOR operatörü
$\vee$: Önerisel OR operatörü
$\wedge$: Önerisel AND operatörü

Kısaltmalar

BT_FF : Bit tabanlı fark fonksiyonu
BT_FF_{min} : İndirgenmiş bit tabanlı fark fonksiyonu
BTİ : Bit tabanlı ifade (bit based clause)
CNF : Konjunktif normal form
DNF : Disjunktif normal form
EKHK : En kötü hafıza karmaşıklığı
EKZK : En kötü zaman karmaşıklığı
FM : Fark matrisi (discernibility matrix)
FF : Fark fonksiyonu
FF_{min} : İndirgenmiş fark fonksiyonu
FFM : Fark fonksiyonu matrisi
KTT : Kod tabanlı terim
ÖLT : Önerisel lojik toplam (propositional logic form)
ÖMAK : Özelliklerin minimal alt kümesi

1. GİRİŞ

Günümüzde bilgisayar ve iletişim teknolojilerindeki gelişmelere bağlı olarak veri depolama işlemlerinin hacmi ve yoğunluğu hızla artmaktadır. Finans sektörü, haberleşme sektörü, sağlık sektörü ve kamu uygulamaları başta olmak üzere birçok alanda veriler *veri tabanlarında* saklanmaktadır. Bir veri tabanı bir veya birden fazla *veri kümesinden* oluşabilir. Bir veri kümesi, $V = \{N, S \cup K\}$ şeklinde gösterilir. Burada, $N = \{O_i\}_{i=1}^{L}$ *objelerin* (nesnelerin) sonlu kümesi, $S = \{A_j\}_{j=1}^{M}$ *şart özelliklerinin* sonlu kümesi ve K *karar özelliğidir*. Her bir şart özelliği bir $\{A_j(O_i)\}_{i=1}^{L}$ sonlu değer kümesine, karar özelliği ise bir $\{K(O_i)\}_{i=1}^{L}$ sonlu değer kümesine sahiptir. Burada $\{A_j(O_i)\}$, A_j şart özelliğinin O_i objesi için değeri ve $K(O_i)$ ise K karar özeliğinin O_i objesi için değeridir. Genel olarak, bir bilgi sistemi, $i \in \{1,2,\ldots,L\}$ satırında $O_i = \{A_j(O_i)\}_{j=1}^{M}$ objesini ve $j \in \{1,2,\ldots,M\}$ sütununda A_j özelliğine ait $\{A_j(O_i)\}_{i=1}^{L}$ değer kümesini bulunduran bir veri kümesidir (Skowron, 1990; Swiniairski, 2001).

Veri kümesi içerisindeki faydalı verilerin bulunması, verilerin analiz edilmesi, verilerden gelecekle ilgili tahmin yapılması gibi işlemleri sağlayacak olan bağlantı ve kuralların aranması ve veri modelleri çıkartılması için *veri madenciliği* yöntemleri kullanılmaktadır.

Veri madenciliği *veri tabanı, istatistik* ve çoğunlukla *makine öğrenmesi* algoritmalarını kullanmaktadır. Bir veri kümesindeki özellik sayısının artması ile verilerin toplanma süreci, verilerin saklanması ve verilerin makine öğrenmesi algoritmaları ile işlenmesinde ihtiyaç duyulan zaman ve hafıza miktarları üstel olarak artar. Veri madenciliğinde karşılaşılabilecek bu gibi istenmeyen durumları önlemek için veri kümeleri *özellik seçme* denilen bir ön işleme tabi tutulmaktadır.

Özellik seçme işleminin amacı, bir V veri kümesini tanımlayan S orijinal özellikler kümesinde bulunan özelliklerden gerekli, önemli ve anlamlı olanları seçerek V veri kümesine ait *özelliklerin minimal alt kümesini* (ÖMAK) bulmaktır. Bir ÖMAK orijinal özelliklerin toplam sayısından daha az sayıda özelliğe sahip olmakla beraber orijinal V veri kümesini aynı başarıda sınıflandırabilmektedir (Liu ve Yu, 2005; Wang ve ark., 2007; Jensen ve Shen, 2007).

Genellikle bir veri kümesi bir veya daha fazla ÖMAK'a sahip olabilir ve bunlar içerisinde en az sayıda özelliğe sahip olana *Reduct* adı verilir (Komorowski ve ark., 1999). Özellik seçme işlemi, bir orijinal veri kümesinde bulunan gereksiz (fazla) özellikleri silip veri kümesini sadeleştirir. Özellik seçme işlemi sonucunda aşağıdaki avantajlar elde edilir:

- Orijinal veri kümesinde bulunan gereksiz özellikler silindiği için veri kümesinin boyutu azalır.
- Veri kümesi oluşturmak için gerekli olan veri toplama işlemi azalır.
- Boyutu küçülen veri kümesi daha basit şekilde tanımlanabilir, görüntülenebilir ve anlaşılabilir.
- Veri depolamak için gerekli olan hafıza miktarı azalır.
- Makine öğrenmesi ve veri madenciliği algoritmalarının çalışma süreleri ve çalışırken ihtiyaç duydukları hafıza miktarları azalır.
- Makine öğrenmesi ve veri madenciliği algoritmalarının sonuçları daha başarılı ve anlaşılabilir bir hale gelir.

Bu avantajlardan dolayı, özellik seçme, veri madenciliği veya makine öğrenmesini kapsayan birçok alanda kullanılabilir. Bunlar: *resim madenciliği, veri sınıflandırması, görüntü tanıma, karar destek sistemleri, müşteri ilişkileri yönetimi, saldırı tespit sistemleri, gen analizi, ekonomi tahmini, hava tahmini, bilgi edinimi ve keşfi, hata teşhisi, hastalık teşhisi* vs. dir.

N adet özelliğe sahip bir veri kümesinde ayrıntılı arama yaklaşımıyla (exhaustive search approach) elde edilebilecek özelliklerin minimal alt kümeleri sayısı $\sum_{i=1}^{N}\binom{N}{i} = 2^N - 1$ tanedir. Bu da özellik seçme işleminin N'e bağlı olarak üstel bir hesaplama karmaşıklığına (zaman ve hafıza karmaşıklığına) sahip olduğunu gösterir. Bu yüzden bir veri kümesindeki özellik sayısı arttıkça özellik seçme işleminin ihtiyaç duyduğu hesaplama karmaşıklığı da üstel olarak artacaktır. Eğer özellik seçme işlemi sonucu olarak sadece bir veya birkaç ÖMAK isteniyorsa, bu amaçla *sezgisel* (heuristic) algoritmalar kullanılabilir. Fakat bu bir riski de beraberinde taşımaktadır. Çünkü sezgisel olarak bulunan ÖMAK ait olduğu veri kümesini tanımlayan en etkili veya başarılı ÖMAK olmayabilir. Bu durumda bir veri kümesine ait bütün ÖMAK'ları

bulmak ve bunlar içerisinden en başarılı veya etkili olanı seçmek gerekir. Bir veri kümesine ait bütün ÖMAK'lar ancak *fark fonksiyonu* (discernibility function) tabanlı özellik seçme yöntemleri ile bulunabilir. Fark fonksiyonu (FF) tabanlı özellik seçme programları temel olarak iki faktörden dolayı bilgisayar hafızasını taşırır ve hata verir. Bunlardan birincisi çok büyük veri tabanları için FF'in çok büyük boyutlarda oluşması, bir diğeri ise FF'in *disjunktif normal forma* (DNF) çevrilirken gerekli hafıza miktarının özellik sayısı ile üstel olarak artmasıdır. Fakat genellikle birinci aşamada oluşturulan FF'nin birçok terimi gereksizdir ve bu gereksiz terimler FF'nin DNF'ye çevrilmesi işlemi sırasında geçici sonuçlar oluşturur ve kullanılan hafıza miktarını arttırır. Bu sebepten FF'deki gereksiz terimleri silerek elde edilen indirgenmiş FF ve bu FF'nin DNF'ye çevrilme işlemi orijinal FF'ye göre daha basit ve hızlı olacaktır. Bu sebeplerden dolayı, bu tez çalışmasında fark fonksiyonu tabanlı bir özellik seçme yöntemi geliştirilmiştir. Bu yöntemde öncelikle herhangi bir veri kümesinin doğruluk tablosu kullanılarak o veri kümesine ait FF'nin indirgenmiş hali elde edilir. Elde edilen indirgenmiş haldeki FF iteratif olarak bölünür. Her iterasyondaki bölme işlemi sonrasında elde edilen bölünmüş FF DNF'ye çevrilir ve çevrim esnasında geçici sonuçların oluşması engellenir. Bütün iterasyonlar sonunda bir veri kümesine ait bütün ÖMAK'lar elde edilmiş olur. Geliştirilen metot diğer FF tabanlı yöntemlere göre çok daha az hafızaya ihtiyaç duyarak özellik seçme işlemini çok daha kısa sürede gerçekleştirir. Dahası, diğer FF tabanlı özellik seçme programları tarafından işlenemeyen veri kümeleri de bu yöntemle işlenebilir.

Yapılan bu tez çalışması 7 bölümden oluşmaktadır. Birinci bölümde özellik seçmenin amacı, avantajları ve özellik seçme işleminde karşılaşılan sorunlardan bahsedilerek bu sorunları çözmek için geliştirilen yöntem açıklanmıştır. 2. bölümde özellik seçme ile ilgili literatürde yapılan çalışmalar hakkında bilgi verilmiştir. 3. bölümde bir bilgi sisteminin ikili doğruluk tablo görüntüsü yardımıyla indirgenmiş fark fonksiyonun (FF) elde edilmesi ve bu FF kullanılarak bir veri kümesine ait ÖMAK'ların oluşturulması anlatılmıştır. 4. bölümde indirgenmiş FF'nun disjunktif normal forma çevrilmesi esnasında oluşan hesap karmaşıklığı değerlendirilmiştir. 5. bölümde indirgenmiş FF'nin disjunktif normal forma çevrilmesi esnasında oluşan hesap karmaşıklığının azaltılabilmesi için fark fonksiyon matrisi (FFM) kullanarak geliştirilen yöntem anlatılmıştır. 6. bölümde yapılan deneylerin sonuçları gösterilmiştir. 7. bölümde ise yapılan tez çalışmasının sonuçları kısaca özetlenerek çalışmanın geliştirilmesi için çeşitli önerilerde bulunulmuştur.

2. ÖZELLİK SEÇMENİN LİTERATÜRDEKİ YERİ

Bir veri kümesi için özellik seçme, 1970'li yıllardan beri araştırılan ve geliştirilen makine öğrenmesi algoritmaları ve veri madenciliği içinde sıklıkla kullanılan bir veri önişlemedir. Özellik seçme işleminde herhangi bir veri kümesindeki orijinal özelliklerden önemli ve gerekli olan özellikleri içeren ÖMAK seçilir. Özellik seçme işlemi bitiminde seçilen ÖMAK içerisinde gereksiz ve önemsiz özellikler bulunmaz ve böylece makine öğrenmesi algoritmalarının öğrenme aşamasındaki verimliği artar ve öğrenilen sonuçların daha anlaşılabilir olması sağlanır (Blum ve Langley, 1997; Dash ve Liu, 1997; Kohavi ve John, 1997).

Son yıllarda gen analizi, görüntü işleme ve müşteri ilişkileri yönetimi gibi birçok alanda kullanılan veri tabanlarının hem örnek hem de özellik sayıları hızla artmıştır. Bu veri tabanlarındaki hızlı büyüme, birçok makine öğrenme algoritmasının ölçeklenebilirliğinde ve öğrenmesinde önemli problemler oluşturur. Mesela, büyük veri tabanlarında (yüzlerce ve binlerce özelliklerden oluşan veri kümeleri) yüksek oranda bulunan gereksiz ve önemsiz özellikler makine öğrenmesi algoritmalarının performansını düşürmektedir. Bu sebepten dolayı günümüzde özellik seçme yüksek boyutlu veri kümeleri ile çalışan makine öğrenmesi algoritmaları için ihtiyaç duyulan önemli bir konu haline gelmiştir. Bu amaçla büyük veri kümelerini indirgemek için araştırmacılar tarafından her hangi bir veri tabanındaki önemli ve gerekli verileri içeren bir veya az sayıda ÖMAK oluşturan sezgisel özellik seçme algoritmaları geliştirilmiştir.

Sezgisel özellik seçme algoritmalarını temel olarak üç grupta sınıflandırılabilir. Bunlar: *filtre metotlar* (filter method), *sarma metotlar* (wrapper method) ve *melez metotlardır* (hybrid methot) (Das,2001; Li ve ark 2005; Chouchoulas,2001; Kohavi ve John, 1997; Yu ve Liu,2003; Hall 1998; Swiniairski, 2001).

Filtre metotlar, veri kümesinin her bir özelliğine istatistiksel ölçütlere göre çalışan değerlendirme fonksiyonları yardımıyla puan verir. Değerlendirme fonksiyonları uzaklık ölçümleri, bilgi ölçümleri, bağımlılık ölçümleri ve tutarlılık ölçümleri gibi ölçümlerin bir veya birkaçını kullanır. Filtre metotlar eğitim veri kümesi üzerinde çalışarak her bir özelliğin puanını ayrı ayrı hesaplar. Sonuç olarak elde edilen puanlar arasından en yüksek puana sahip özellikler ÖMAK'ı oluşturur (Gheyas ve Smith, 2010). Sıklıkla kullanılan filtre metotları t-test (Hua ve ark., 2008), chi-square test (Jin ve ark., 2006), Wilcoxon Many-Whitney test (Liao ve ark., 2007), karşılıklı bilgi (mutual information) (Peng ve ark., 2005), Pearson korelasyon katsayıları (Pearson correlation

coefficients) (Biesiada ve Duch, 2008) ve temel bileşenler analizidir (principal component analysis) (Rocchi ve ark., 2004). Filtre metotlar sarma metotlara göre daha hızlı çalışmasına rağmen genellikle veri kümesinde bulunan gereksiz ve önemsiz özelliklerin belirlenmesinde ve birbirleriyle etkileşim içerisinde olan özellik gruplarının belirlenmesinde çok başarılı değildir. Buna ek olarak filtre metotlarda ÖMAK'ı oluşturacak özelliklerin seçiminde kullanan gerekli ve gereksiz olanları birbirinden ayıran eşik değerinin nasıl belirlendiği tam olarak kesinlik kazanmamıştır. Bu sebeplerden dolayı filtre metotlar tarafından seçilen ÖMAK genellikle sarma metotlar tarafından seçilen ÖMAK'a göre daha az etkili veya başarılıdır.

Sarma metotlarda ÖMAK belirlenirken bir sınıflandırıcıdan veya öğrenme algoritmasından faydalanılır. Sarma metotlarla seçilen ÖMAK'ın uygunluğu orijinal veri kümesi üzerinde çalıştırılması planlanan öğrenme algoritması veya sınıflandırıcı tarafından test edilerek belirlenir. Bu yüzden sarma metotlarda ki seçim kıstasına bağımsız kıstas adı verilir. Mesela sınıflandırma işlemine sokulacak veri kümesi için özellik seçme işleminde ki uygunluk kıstası sınıflandırma doğruluk oranı (classification accuracy) veya bir kümeleme işlemine sokulacak veri kümesi için özellik seçme işlemindeki uygunluk kıstası kümeleme iyiliği değeri (cluster goodness) olarak belirlenebilir (Somol ve ark., 2007). Genellikle sarma metotlar filtre metotlarına göre daha yavaş çalışmalarına rağmen daha iyi performans sergilerler. Sarma metotlar arama stratejilerine göre iki gruba ayrılırlar: *açgözlü (greedy) sarma metotlar* ve *rasgele/tahmini (randomized/stochastic) sarma metotlar*.

Açıkgözlü sarma metotlar diğer sarma metotlara göre daha az işlemci zamanı kullanırlar. *Sıralı geriye doğru seçim* (Cooter ve ark., 2001) ve *sıralı ileriye doğru seçim* (Colak ve Isik, 2003) en çok kullanılan açıkgözlü tepe tırmanma arama stratejisi (greedy hill-climbing search strategy) algoritmalarındandır. Sıralı geriye doğru seçim algoritmasının başlangıcında ÖMAK orijinal veri kümesindeki bütün özellikleri içerir ve algoritma ilerledikçe en az umut veren özellikler ÖMAK'tan çıkarılır. Eğer ÖMAK'ta kalan özelliklerden birinin daha çıkarılmasıyla değerlendirme kıstası olarak kullanılan öğrenme algoritmasının performansı daha önceden belirlenen eşik değerinin altına düşerse algoritma sonlanır. Sıralı geriye doğru seçim algoritması tekdüzelik varsayımına dayanır (Yang ve Honavar, 1998). Yani sıralı geriye doğru seçim algoritmasının çalışması sonunda elde edilen varsayılan performans bu veri taban için en az sayıda özellikle elde edilmiş olan en iyi performanstır. Elde edilen ÖMAK'a yeni bir özellik eklense bile elde edilecek olan performans elde edilmiş olan varsayılan

performanstan daha iyi olamaz (Swiniairski, 2001). Fakat bu varsayım şüphelidir çünkü arama uzayının çok geniş olması sıralı geriye doğru seçim algoritmasına çeşitli zorluklar getirir. Çünkü gerçekte her hangi bir veri kümesinin özellik uzayının artması ile sıralı geriye doğru öğrenme algoritmasının tahmini başarısı azalır. Böylece sıralı geriye doğru seçim algoritması yüksek boyutlu bir veri kümesiyle karşılaştığı zaman silinecek her hangi bir özelliğin sonucu tam olarak nasıl etkileyeceğini belirleyemeyebilir ve böylece gerekli ve önemli bir özelliği algoritmanın ilk döngülerinde silinebilir. Buna karşılık, sıralı ileriye doğru seçim algoritmaları boş ÖMAK ile başlar ve sınıflandırma başarısında hiçbir gelişme olmayana kadar en umut verici özellikleri yinelemeli olarak ÖMAK'a ekler. İlk adımda en umut verici özellik seçilirken, bundan sonraki adımlarda seçilmiş özellikler ile en iyi birlikteliği sağlayıp sınıflandırma başarısını arttıran özellik seçilerek ÖMAK'a eklenir. Sıralı geriye doğru seçim ve sıralı ileriye doğru seçim algoritmalarının en önemli problemi tek yönlü arama yapmalarıdır. Bu probleme çözüm bulmak için Pudil ve ark. (1994) özellik seçmede değişen sıralı arama algoritmasını sunmuşlardır. Bu algoritma her yinelemede ÖMAK'a bir eleman ekler veya bir elemanı ÖMAK'tan çıkartır. Fakat daha sonra yapılan deneysel çalışmalara göre değişen sıralı arama algoritması sıralı ileriye doğru seçim algoritmasından daha etkili değildir (Bensch ve ark., 2005) ve 100 özellikten daha fazla özelliği içeren veri kümeleri içinse uygun değildir (Ng ve ark., 1997). Sıralı arama algoritmalarında bir özelliğin ÖMAK'a eklenmesi veya çıkarılması sadece o özelliği etkilemez o an için ÖMAK'ta var olan tüm özellikleri etkiler. Bu sebepten dolayı bir özelliğin ÖMAK'a eklenmesinin veya çıkarılmasının ne kadar uygun olup olmayacağı kesin olarak açıklanamaz. Bu da sıralı arama algoritmalarının en temel problemidir.

Tahmini sarma metotlar geniş ölçekli kombinasyon problemlerinin çözümü için kullanılan karınca koloni optimizasyonu, genetik algoritmalar, parçacık sürüsü optimizasyonu ve tavlama benzetimi (simulated annealing) gibi algoritmalardır (Yang ve Honavar, 1997; Vieira ve ark., 2007; Wang ve ark., 2007; Ronen ve Jacob, 2006). Bu tip algoritmalar bir özelliğin gerekliliğini ve diğer özellikler ile etkileşimini etkili bir şekilde belirler ve buna göre ÖMAK'ı oluşturur. Fakat bu tip algoritmaların dezavantajı hesaplama karmaşıklığının çok fazla olmasıdır (Gheyas ve Smith, 2010).

Son senelerde birçok araştırmacı hem filtre ve hem de sarma metotların ortak avantajlarını kullanabilmek için melez (hybrid) sezgisel özellik seçme algoritmaları geliştirmişlerdir. Tan ve ark. (2006) t-statistics algoritmasını ve genetik algoritmayı içeren, Shazzad ve Park (2005) korelasyon tabanlı özellik seçme algoritmasını ve

genetik algoritmayı içeren, Yan ve Yuan (2004) temel bileşenler analizi ve karınca koloni optimizasyonu algoritmalarını içeren, Sivagaminathan (2007) yapay sinir ağları ve karınca koloni optimizasyonu algoritmalarını içeren ve Osei-Bryson ve ark. (2003) chi-square ve çok amaçlı optimizasyon algoritmalarını içeren Fatourechi ve ark. (2007) ve Huang ve ark. (2006) karşılıklı bilgi algoritması (mutual information) ve genetik algoritmayı içeren melez sezgisel özellik seçme yaklaşımları geliştirmişlerdir. Melez metotların temelinde yatan fikirde filtre metotlar ile orijinal özellik kümesinden en az öneme sahip olan özellikler silinerek bir özellik havuzu oluşturulur. Daha sonra sarma metotlar oluşturulan bu özellik havuzunu kullanarak ÖMAK'ı seçer.

Sezgisel özellik seçme algoritmaları ile seçilen ÖMAK'ların sayısı genellikle bir veya birkaç taneden fazla değildir. Fakat bir veri tabanına ait faklı ÖMAK'lar o veri tabanının kullanılacağı makine öğrenmesi veya veri madenciliği algoritmaları için farklı etkiler veya başarı oranları gösterebilir. Yani sezgisel algoritmalar tarafından seçilen ÖMAK en etkili veya başarılı olan ÖMAK olmayabilir (Hall ve Holmes, 2003). Bu problemi çözmek içinde bir veri tabanına ait bütün ÖMAK'ları bulmak ve bunlar içerisinden en uygun veya başarılı olanı seçmek gereklidir. Buda sadece FF tabanlı özellik seçme algoritmaları tarafından yapılabilir.

FF tabanlı özellik seçme algoritmaları FF'yi DNF'ye çevirerek bir veri kümesine ait bütün ÖMAK'ları bulur ve bu işlem bir NP-hard problemdir. Bu yüzden hesaplama karmaşıklığı da çok büyüktür (Jensen ve Shen, 2004; Skowron ve Rauszer, 1992; Chen ve ark., 2008). Günümüze kadar FF tabanlı özellik seçme konusunda birçok çalışma yapılmasına rağmen bu çalışmaların çok azında bu yöntemin hesaplama karmaşıklığının azaltılması üzerinde durulmuştur. Özellikle Ohrn ve ark. (1998) FF tabanlı Johnson azaltıcı (Johnson reducer) (Johnson, 1974) olarak da adlandırılan bir algoritma açıklamışlardır. Bu algoritmaya göre ayırt edici matriste en yüksek frekansa sahip özellik, ÖMAK'a eklenmekte ve FF içerisinde bu özelliği içeren diğer bütün satırlar FF'den silinmektedir. FF içerisindeki bütün satırlar silindiği zaman algoritma sonlanmakta ve elde edilen ÖMAK algoritmanın sonuç değerini vermektedir (Jensen ve Shen, 2007). Benzer bir yaklaşımla Wang ve Wang (2001) ÖMAK'ı iteratif olarak bulmaktadır. Bu yaklaşımın her iterasyonunda FF'de en yüksek frekansa sahip özellik seçilmekte ve bu özelliği içeren bütün elemanlar ayırt edici matristen silinmektedir. Algoritma ÖMAK'ı bulana kadar devam etmektedir. Her iki metotta FF tabanlı özellik seçme yöntemleri olmasına rağmen sezgiseldirler. Çünkü sonuç olarak elde edilen ÖMAK'ın en uygun ÖMAK olup olmadığını kesin olarak ispatlanamaz. Starzyk ve ark.

(2000) özellik seçme işlemini hızlandırabilmek için güçlü sıkıştırılabilirlik (strong compressibility) kavramını ortaya atmışlardır. Bu kavram yardımıyla indirgenmiş FF oluşturmuşlar ve genişleme algoritması (expansion algorithm) adını verdikleri algoritma ile herhangi bir veri kümesine ait bütün ÖMAK'ları daha hızlı bulduklarını söylemişlerdir. Fakat Wang ve ark. (2007) bu yaklaşımın sadece küçük veri kümelerinde etkili oluğunu savunmuştur. Tan ve ark. (2007) yeni bir reduct seçme algoritması sunmuşlardır. Bu algoritmanın temeli Boole uzayındaki kesikli boyut indirme problemlerinin gerçek uzaydaki sürekli optimizasyon problemlerine dönüştürülmesidir. Yapılan deneysel çalışmalar da bu yaklaşımın Dinamik Reduct (Bazan, 1998; Bazan, 1994) ve Genetik Reduct (Vinterbo, 2000) yaklaşımlarından daha hızlı çalıştığı ispatlamıştır.

3. VERİ KÜMELERİNDE ÖZELLİK SEÇME İŞLEMİNE BOOLE FONKSİYON YAKLAŞIMI

3.1. Veri kümelerine Boole fonksiyonu yaklaşımı

Girişleri sayısı n çıkışları sayısı ise m olan bir D Boole fonksiyonu $h: B^n \to Y^m$ şeklinde gösterilir. Burada $B = \{0,1\}$ ve $Y = \{0,1,*\}$. Hatırlatalım ki, burada $B^n = 2^n$ giriş kombinasyonları uzayı, $Y^m = 3^m$ ise çıkış kombinasyonları uzayıdır. B^n uzayındaki her hangi bir kombinasyon *minterm* olarak isimlendirilir.

$m = 1$ durumunda D fonksiyonu tek çıkışlı bir fonksiyon olur. Tek çıkışlı bir D fonksiyonun giriş kombinasyonları kümesi *doğrular kümesi* denilen $S_{ON}(D)$, *yanlışlar kümesi* denilen $S_{OFF}(D)$ ve *önemsizler kümesi* denilen $S_{DC}(D)$ kümelerine ayrılır. Bilindiği gibi, n değişkeni için 2^n kadar minterm mümkündür. Bu mintermlerden her birini bir x olarak gösterirsek $S_{ON}(D)$, $S_{OFF}(D)$ ve $S_{DC}(D)$ kümeleri formal olarak aşağıdaki gibi tanımlanabilir:

$$S_{ON}(D) = \{x : x \in B^n \; ve \; D(x) = 1\} \tag{3.1}$$

$$S_{OFF}(D) = \{x : x \in B^n \; ve \; D(x) = 0\} \tag{3.2}$$

$$S_{DC}(D) = B^n - (S_{ON}(D) \cup S_{OFF}(D)) \tag{3.3}$$

Eğer bir D fonksiyonu için $S_{DC}(D) = \varnothing$ durumunda o fonksiyon *tam belirlenmiş fonksiyon* olarak, $S_{DC}(D) \neq \varnothing$ durumunda ise söz konusu fonksiyonu *natamam belirlenmiş fonksiyon* olarak tanımlanır. İleride, $S_{ON}(D)$ kümesindeki mintermler *Doğru-minterm*, $S_{OFF}(D)$ kümesindeki mintermler *Yanlış-minterm* ve $S_{DC}(D)$ kümesindeki mintermler ise *Önemsiz-minterm* olarak adlandırılacaktır (Şirzat ve ark., 2011).

3.2. Bir veri kümesinin Boole fonksiyonu olarak yorumlanması

Bir özellik seçme problemini bir Boole fonksiyonu olarak ele alabilmek için, D karar özelliğine $A_1, A_2, \ldots A_n$ şart özelliklerinin fonksiyonu olarak bakılır, yani bir veri

kümesi bir $D = f(A_1, A_2, \ldots A_n)$ fonksiyonu şeklinde temsil edilir. Eğer bir veri kümesini karakterize eden bütün özellikler ikili değerli (binary valued) olursa bu veri kümesi n değişkenli bir Boole fonksiyonu ile temsil edilebilir. Bu şekilde tanımlanabilen veri kümeleri, bu tez çalışmasında, *ikili değerli veri kümeleri* olarak adlandırılacaktır. Fakat şu unutulmamalıdır ki genellikle bir veri kümesinde bulunan herhangi bir özellik ikiden fazla değere sahip olabilir. Bu tip özeliklere *çok değerli özellik* ve bu tip özellikleri içeren veri kümelerine ise *çok değerli veri kümeleri* adı verilir. Boole fonksiyonunu çok değerli özellikleri olan bir veri kümesine uygulayabilmek için söz konusu özelliklerin değerlerinin aşağıdaki gibi kodlanması gerekir.

1. A_j özelliğinin aldığı bütün farklı değerleri içeren bir V_j kümesi oluşturulur. Bu V_j kümesindeki değerleri ikili koda çevirebilmek için kaç bitin gerekli olduğu $n_j = \left\lfloor \log_2 \left| V_j \right| \right\rfloor$ formülü ile hesaplanır.
2. İkili olarak n_j adet bitle kodlanmış özellik değerlerini içeren yani V_j değer kümesinin ikili kodlu hali olan E_j kümesi oluşturulur.
3. Birinci ve ikinci adımlar bütün özellikler için tekrarlanır: $j = 1,2,3,\ldots n$.
4. Karar değişkeninin bütün farklı değerlerini içeren bir V_d kümesi oluşturulur. Bu V_d kümesindeki değerleri ikili koda çevirebilmek için karar değişkeninin kaç bitle ifade edileceği $d = \left\lfloor \log_2 \left| V_d \right| + 1 \right\rfloor$ formülü ile hesaplanır.
5. İkili olarak d adet bitle kodlanmış karar değişkenini içeren V_d kümesinin ikili kodlu hali olan E_d kümesi oluşturulur. Burada $\{0\}^d$ kodu V_d kod kümesinde yer almaz.

Örnek olarak Çizelge 3.1'de gösterilen çok değerli veri kümesinin $A_5(Feet)$ özelliğinin ve D karar özelliğinin ikili koda çevrilmesini gösterelim.

Çizelge 3.1. Örnek veri kümesi

Hair A_1	Teeth A_2	Eye A_3	Feather A_4	Feet A_5	Eat A_6	Milk A_7	Fly A_8	Swim A_9	Animal D
Yes	Pointed	Forward	No	Claw	Meat	Yes	No	Yes	Tiger/Cheetah
Yes	Blunt	Side	No	Hoff	Grass	Yes	No	No	Giraffe/Zebra
No	No	Side	Yes	Web	Fish	No	No	Yes	Penguin
No	No	Side	Yes	Claw	Grain	No	Yes	Yes	Albatross
No	No	Side	Yes	Claw	Grain	No	No	No	Ostrich
No	No	Forward	Yes	Claw	Meat	No	Yes	No	Eagle

A_5 özelliği için değer kümesi $V_5 = \{Claw, Hoff, Web\}$ şeklinde oluşturulur. Bu özellikte $|V_5| = 3$ değeri olduğundan dolayı bu özelliği ikili koda çevirmek için $n_5 = \lceil \log_2 |3| \rceil = 2$ adet bit gereklidir. Sonuç olarak $V_5 = \{Claw, Hoff, Web\}$ kümesi $E_5 = \{00,01,10\}$ olarak ikili kodlanır. Bu özelliğin değerleri 2 bit ile kodlandığı için A_5 özelliğinin A_{51} ve A_{52} olmak üzere iki adet alt özelliği bulunur. D karar özelliğinin değer kümesi $V_D = \{Tiger/Cheetah, Giraffe/Zebra, Penguin, Albatross, Ostrich, Eagle\}$ şeklinde oluşturulur. Karar özelliğinde $|V_D| = 6$ değeri olduğundan dolayı karar özelliğini ikili koda çevirmek için $d = \lceil \log_2 (6+1) \rceil = 3$ adet bit gereklidir. Sonuç olarak V_D kümesi $E_D = \{001,010,011,100,101,110\}$ şeklinde ikili kodlanır. E_D kümesindeki elemanların her biri 3 bit ile ifade edildiği için D karar özelliği D_1, D_2 ve D_3 alt fonksiyonlarından oluşur.

Yukarıdaki örnek bize çok-değerli özellikleri olan bir veri kümesinin özelliklerinin değerlerinin ikili koda çevrilmesiyle, ikili değerli bir veri kümesi oluşturulabileceğini göstermiştir. Bu oluşturulan veri kümesine Boole fonksiyonun doğruluk tablosu olarak bakılabilir. Sonuç olarak, bütün veri kümelerine Boole fonksiyonlar tarafından temsil edilebilecek ve üzerinde işlem yapılabilecek nesneler olarak bakılabilir.

3.3. Bir veri kümesinin doğruluk tablo görüntüsünün elde edilmesi

Boole fonksiyonların gösterimdeki temel veri yapısı doğruluk tablosudur. Her Boole fonksiyon bir doğruluk tablosunun iki sütunu ile temsil edilir. Birinci sütun mintermler sütunudur. Bu sütunun her satırında $S_{ON}(D) \cup S_{OFF}(D)$ kümesindeki mintermlerden bir tanesi bulunur. İkinci sütun ise *fonksiyon (karar)* sütunudur (Şirzat ve

ark., 2011). Bu sütunun her satırında fonksiyonun satırın birinci sütununda bulunan mintermden aldığı karar değeri yazmaktadır. Bir veri kümesi ilgili doğruluk tablosuna aşağıdaki gibi çevrilir:

1. Karar özellikleri ve şart özelliği değerleri ikili kodlanır.
2. Şart özelliklerinin değerlerinin ikili kodları birleştirilerek her objeye ait mintermler oluşturulur.
3. 2. adımın sonucu veri kümesinin doğruluk tablo görüntüsü olarak nitelendirilir.

Örnek 3.1: Çizelge 3.2'de tüm özellikleri ikili değerli olan örnek bir veri kümesi (Tavangavel ve ark., 2005) gösterilmiştir.

Çizelge 3.2. Örnek ikili değerli veri kümesi

Objeler	Şart özellikleri				Karar özelliği
	Weight A1	Door A2	Size A3	Cylinder A4	Mileage D
U1	Low	2	Com	4	High
U2	Low	4	Sub	6	Low
U3	Low	4	Com	4	High
U4	High	2	Com	6	Low
U5	High	4	Com	4	Low
U6	High	4	Sub	6	Low
U7	Low	2	Sub	6	Low

Bu tablodaki şart özelliklerinin ve karar özelliğinin değerleri aşağıdaki gibi ikili kodlara çevrilir:

$$V_1 = \{Low, High\} \rightarrow E_1 = \{0,1\}$$

$$V_2 = \{2,4\} \rightarrow E_2 = \{1,0\}$$

$$V_3 = \{Com, Sub\} \rightarrow E_3 = \{0,1\}$$

$$V_4 = \{4,6\} \rightarrow E_4 = \{0,1\}$$

$$V_d = \{Low, High\} \rightarrow E_d = \{0,1\}$$

Bu kodlamalara dayanarak, aynı satırda bulunan şart kodları birleştirilerek Çizelge 3.2'de verilen veri kümesinin doğruluk tablosu gösterimi Çizelge 3.3'deki gibi oluşturulur.

Çizelge 3.3. Çizelge 3.2'deki ikili değerli veri kümesinin doğruluk tablosu gösterimi

Mintermler					Fonksiyon
	A1	A2	A3	A4	D
T1	0	1	0	0	1
T2	0	0	1	1	0
T3	0	0	0	0	1
T4	1	1	0	1	0
T5	1	0	0	0	0
T6	1	0	1	1	0
T7	0	1	1	1	0

Bu tabloya dayanarak bölüm 3.1' de anlatılan $S_{ON}(D)$, $S_{OFF}(D)$ ve $S_{DC}(D)$ kümeleri aşağıdaki denklemlere göre oluşturulur:

$$S_{ON}(D) = \{T : D(T) = 1\} \tag{3.4}$$

$$S_{OFF}(D) = \{T : D(T) = 0\} \tag{3.5}$$

$$S_{DC}(D) = \{0,1\}^n - \{T_i\}_{i=1}^{M} \tag{3.6}$$

Burada n şart özelliklerinin sayısı $\left|\{0,1\}^n\right| = 2^n$ ve M ise veri kümesindeki objelerin sayısıdır. 3.4, 3.5 ve 3.6 denklemlerini Çizelge 3.3'e uygulayarak aşağıdaki kümeleri elde ederiz.

$$S_{ON}(D) = \{T_1, T_3\} = \{0100, 0000\}$$
$$S_{OFF}(D) = \{T_2, T_4, T_5, T_6, T_7\} = \{0011, 1101, 1000, 1011, 0111\}$$
$$S_{DC}(D) = \{0,1\}^4 - \{T_i\}_{i=1}^{7} = \{0001, 0010, 0101, 0110, 1001, 1010, 1100, 1110, 1111\}$$

Genelde büyük veri kümeleri içerebilecekleri objelerin sadece çok az bir kısmını bulundururlar. Yani bir veri kümesindeki şart özelliklerinin sayısı ve şart özelliklerin alabileceği değerlerin sayısı da artarsa $S_{DC}(D)$'nin $S_{ON}(D) + S_{OFF}(D)$'e olan oranı da artar. Bu ise büyük veri kümelerindeki özellik seçme işlemi esnasında, $S_{DC}(D)$ kümelerini işlemek çok fazla hesaplama karmaşıklığına sebep olacaktır anlamına gelmektedir. Bunun yerine $S_{OFF}(D)$ kümesini kullanmak özellik seçme işlemini kolaylaştıracak ve daha az hesaplama karmaşıklığı yaratacaktır. Bu sebepten bundan

sonra doğruluk tablosuna dayanan işlemlerde $S_{ON}(D)$ ve $S_{OFF}(D)$ kümeleri kullanılacaktır.

3.4. Bir veri kümesi için fark fonksiyonunun elde edilmesi

$S=\{U, C\cup D\}$ gibi bir veri kümesinin var olduğunu düşünelim. Burada $U=\{u_1, u_2 \ldots u_m\}$ objeler kümesi, $C=\{a_1, a_2 \ldots a_n\}$ özellikler kümesi, D ise karar özelliğidir. Her $a_j \in C$ özelliğinin alabileceği değerler kümesi $V_{a_j}=\{a_j(u_i)\}_{i=1}^{|m|}$' dir. Burada $a_j(u_i)$, a_j özelliğinin u_i objesinde aldığı değer ve m ise veri kümesindeki toplam obje sayısıdır. Bu S veri kümesinin fark matrisi (FM) $m\times m$ boyutunda bir matristir. Bu matrisin her bir giriş değerini oluşturan H_{ik}, u_i ve u_k objelerinin birbirinden farklı olduğu özelliklerin lojik toplamıdır ve aşağıda gösterildiği gibi elde edilir (Degang ve ark. 2007; Jensen ve Shen, 2007; Skowron,1990, Skowron ve Rauszer,1992; Komorowski ve ark., 1999).

$$H_{ik}=\left(\forall a_j : a_j(u_i)\neq a_j(u_k)\ \&\ d_i\neq d_k,\ j\in\{1,2,\ldots n\}\ i,k\in\{1,2,\ldots m\}\right) \tag{3.7}$$

Denklem 3.7'de tanımlanan H_{ik} önerisel lojik toplamı (ÖLT) (propositional logic clause) aşağıdaki gibi tanımlanır:

$$H_{ik}=h_{ikj}=h_{ik1}\vee h_{ik2}\vee\ldots\vee h_{ikj}\vee\ldots\vee h_{ikn} \tag{3.8}$$

Denklem 3.8'e göre eğer $a_j(u_i)\neq a_j(u_k)$ ise $h_{ikj}=a_j$ ve $a_j(u_i)=a_j(u_k)$ ise $h_{ikj}=0$'dır. Denklem 3.8'deki ÖLT'de bulunan ifadelerin birleştirilmesiyle bir veri kümesinin FF'si elde edilebilir. Bunu yapan algoritma şekli 3.1.'de gösterilmektedir (Skowron ve Rauszer, 1992, Komorowski ve ark., 1999, Jensen ve Shen, 2007; Swiniarski ve A. Skowron,2003).

```
ÖLT_FM_OLUŞTUR (U, m, n)
{
        FM = ∅
        For i = 1 to m-1
        {       For k = i+1 to m
                {       H_ik =∅
                        If d_i ≠ d_k
                        {       For j = 1 to n
                                {
                                        If a_j(u_i) ≠ a_j(u_k) then
                                        {       h_ikj = a_j
                                                H_ik = H_ik ∨ h_ikj
                                        }
                                }
                                DF= DF ∪ H_ik
                        }
                }
        }
    Return (DF)
}
```

Şekil 3.1. Önerisel lojik toplamda fark matrisi oluşturan algoritma

Şekil 3.1'de gösterilen algoritmanın hafıza karmaşıklık değeri $O(m^2)$ ve zaman karmaşık değeri $O(n \times m^2)$'dir (Wang ve ark., 2007; Brayton ve ark., 1984). Ancak $O(n \times m^2)$ olan zaman karmaşıklığı paralel işleme yöntemleri yardımıyla $O(m^2)$'ye düşürülebilir.

Örnek 3.2: Çizelge 3.2' deki veri kümesinin FF'sini elde edelim. ÖLT_FM_OLUSTUR algoritması kullanılarak elde edilen FF aşağıdaki ÖLT'deki elemanlardan oluşur.

$H_{12} = A_2 \vee A_3 \vee A_4$, $H_{14} = A_1 \vee A_4$, $H_{15} = A_1 \vee A_2$, $H_{16} = A_1 \vee A_2 \vee A_3 \vee A_4$, $H_{17} = A_3 \vee A_4$, $H_{23} = A_3 \vee A_4$, $H_{34} = A_1 \vee A_2 \vee A_4$, $H_{35} = A_1$, $H_{36} = A_1 \vee A_3 \vee A_4$, $H_{37} = A_2 \vee A_3 \vee A_4$

Bu elemanların birleştirilmesiyle oluşan FF aşağıdaki gibi gösterilir:

$$FF = \bigcup_{\forall i} \bigcup_{\forall k} H_{ik} = H_{12} \cup H_{14} \cup H_{15} \cup H_{16} \cup H_{17} \cup H_{23} \cup H_{34} \cup H_{35} \cup H_{36} \cup H_{37}$$

3.4.1. Fark fonksiyonunun indirgenmesi

Genellikle, elde edilen FF içerisinde gereksiz (başkaları tarafından yutulan) bileşenler bulunabilir. Bu bileşenler $a \wedge (a \vee b) = a$ lojik kuralını kullanarak sadeleştirilir. Bu kurala göre H_{23} bileşeni H_{12}, H_{17} ve H_{37} bileşenlerini, H_{35} bileşeni ise H_{14}, H_{15}, H_{16}, H_{34} ve H_{36} bileşenlerini yutar. Böyleye indirgenmiş FF'de $H_{23} = A_3 \vee A_4$ ve $H_{35} = A_1$ bileşenleri kalır. Burada önce FF'in tamamının oluşturulması ve daha sonra oluşturulan FF içerisindeki gereksiz terimlerin silinmesi, algoritmanın hesaplama karmaşıklığını arttırmaktadır. Bu olumsuzluğu önlemek için FF'nin oluşturulması esnasında gereksiz elemanlar FF'ye eklenmeden silinmelidir. Bitsel lojik işlemlere dayanan bu çözüm yöntemi aşağıda verilmiştir.

3.5. Bir veri kümesinin doğruluk tablo görüntüsünden indirgenmiş fark fonksiyonunun elde edilmesi

Buradaki amaç, bir veri kümesine ait indirgenmiş FF'yi bitsel işlemler kullanarak veri kümesinin doğruluk tablo görüntüsünden elde etmektir. Bunun için ÖLT'de yazılan 3.8 denkleminin aşağıdaki gibi bir *bit tabanlı ifadesi* (BTİ) yazabilir:

$$B_{ik} = b_{ik1} b_{ik2} \ldots b_{ikj} \ldots b_{ikn} \tag{3.9}$$

Bu durumda b_{ikj} aşağıdaki gibi elde edilir.

$$a_j(u_i) \neq a_j(u_k) \text{ ise } b_{ikj} = 1 \text{ ve } a_j(u_i) = a_j(u_k) \text{ ise } b_{ikj} = 0 \quad (3.10)$$

3.8 ve 3.9 denklemleri karşılaştırılırsa, denklem 3.9'da lojik OR (V) operatörünün bırakıldığı görülür. Her hangi bir BTİ bu şekilde daha sıkışık ve daha basit bir gösterimle ifade edilebilir. Ancak her hangi bir BTİ'yi bir lojik işleme sokmadan önce, BTİ'nin komşu komponetleri arasındaki işaretler dikkate alınmalıdır. Bir ÖLT'nin BTİ'ye çevrilmesinde veya BTİ'nin ÖLT'ye çevrilmesi aşamasında, 3.8 ve 3.9 denklemleri arasındaki ilişki aşağıdaki veri yapısı ile belirlenir:

Struct_ÖLT-BTi{Unsigned a_1:1; Unsigned a_2:1; … ;Unsigned a_n:1;} (3.11)

Bunun manası şudur ki, eğer bir a_j özelliği ÖLT içerisinde bulunuyorsa, BTİ'de j. bitin değeri 1'dir ve eğer a_j özelliği ÖLT içerisinde bulunmuyorsa, BTİ'de j. bitin değeri 0'dir. Yani, eğer BTİ'de j. bitin değeri 1 ise a_j özelliği ÖLT içerisinde bulunur ve eğer BTİ'de j. bitin değeri 0 ise a_j özelliği ÖLT içerisinde bulunmaz.

Yukarıda, alt bölüm 3.3'de her hangi bir ikili kodlu veri kümesinin bir doğruluk tablosu ile gösterilebileceği ve bu doğruluk tablosundaki her U_i objesinin bir T_i mintermi ifade edilebileceği açıklanmıştı. Buradan denklem 3.9'da ifade edilen U_i ve U_k objeleri arasındaki B_{ik} farkı basit bir şekilde aşağıdaki gibi elde edilebilir.

$$B_{ik} = T_i \oplus T_k \quad (3.12)$$

Mesela, Çizelge 3.2'deki U_1 ve U_2 objeleri Çizelge 3.3'de T_1 ve T_2 mintermleri olarak ifade edilmişlerdir. Denklem 3.12'ye göre bu objeler arasındaki farklar aşağıdaki gibi bulunabilir.

$$B_{12} = T_1 \oplus T_2 = 0100 \oplus 0011 = 0111$$

U_1 ve U_2 objeleri arasındaki fark olarak elde edilen bu BTİ'nin ÖLT karşılığı denklem 3.11'de tanımlanan yapıya göre aşağıdaki gibi olacaktır.

$$H_{12} = A_2 \vee A_3 \vee A_4$$

Yukarıda, alt bölüm 3.3'de ikili değerli bir veri kümesinin doğruluk tablosundaki mintermlerin D karar değişkenine göre $S_{ON}(D) = \{T^{ON} : D(T) = 1\}$ ve $S_{OFF}(D) = \{T^{OFF} : D(T) = 0\}$ kümelerine bölüneceği anlatılmıştı. Bu kümeler dikkate alınarak denklem 3.12 aşağıdaki gibi yeniden yazılır.

$$B_{ik} = T_i^{ON} \oplus T_k^{OFF} \text{ i} = 1,2,\ldots M_1 \text{ ve k} = 1,2,\ldots M_2 \tag{3.13}$$

Burada $M_1 = |S_{ON}(D)|$, S_{ON} kümesinde bulunan elemanların toplam sayısı, $M_2 = |S_{OFF}(D)|$ ise S_{OFF} kümesinde bulunan elemanların toplam sayısıdır. Denklem 3.13 ile elde edilen BTİ'ler aşağıdaki kümede birleştirilir. Bu kümeye *bit tabanlı fark fonksiyonu (BT_FF)* adı verilir.

$$BT_FF(D) = \bigcup_{i=1}^{M_1} \bigcup_{k=1}^{M_2} B_{ik} \tag{3.14}$$

Denklem 3.14'de ki işlem sonrasında $BT_FF(D)$ fonksiyonundaki toplam bileşen sayısı $|BT_FF(D)| = M_1 \times M_2$ kadar olur. $BT_FF(D)$ fonksiyonu aşağıdaki gibi FF'e çevrilir (Jensen ve Shen, 2007; Komorowski ve ark., 1999).

$$FF(D) = \bigcap_{i=1}^{M_1} \bigcap_{k=1}^{M_2} (\varphi_0 : B_{ik} \in BT_FF(D)) \tag{3.15}$$

Burada $\cap$ lojik çarpmanı, φ_0 ise BTİ'lerin ÖLT'ye dönüştürülmesi gerektiğini simgelemektedir. Denklem 3.14 ve 3.15'den anlaşılabileceği gibi $BT_FF(D)$ kümesi FF'in bit tabanlı temsilidir. Fakat genellikle $BT_FF(D)$ fonksiyonunda fazla BTİ'ler de bulunabilir. Bunların $BT_FF(D)$ fonksiyonundan silinmesiyle $BT_FF(D)$

fonksiyonunu $M_1 \times M_2$ olan eleman sayısı $Q \leq M_1 \times M_2$'ye düşer. Örnek olarak, Çizelge 3.4'de UCI makine öğrenmesi ambarındaki bir çok veri kümesine ait tüm BTİ'leri içeren orijinal $BT_FF(D)$ fonksiyonları ve gereksiz BTİ'lerin silinmesi ile indirgenmiş $BT_FF_{\min}(D)$ fonksiyonlarının karşılaştırılması gösterilmektedir.

Çizelge 3.4. $BT_FF(D)$ ile $BT_FF_{\min}(D)$'in karşılaştırılması

Veri kümesi	Özellik / Obje sayısı	BT_FF	BT_FF$_{min}$	BT_FF/BT_FF$_{min}$
Anneal	38/798	318003	55	5781,8
Australian	14/690	237705	23	10335
Iris	4/150	11175	6	1862,5
Monk1	6/124	7626	3	2542
Monk2	7/169	14196	6	2366
Monk3	6/122	7381	4	1845,3
Diabetes	8/168	14028	15	935,2
Hearth	13/270	36315	68	534
Zoo	17/101	5050	14	360,7
Lymn	18/148	10878	154	70,6
Statlog	24/1000	499500	309	1616,5
Chess	36/3196	5×10^6	29	1.76×10^5
Vote	16/435	94395	15	6293
Mushroom	22/8124	32×10^6	30	1.1×10^6
Ionesphere	33/351	61425	235	261.3
Shuttle	9/43500	946×10^6	8	118×10^6
Sonar	60/208	21528	2004	10.7
Spectf	44/187	17391	2096	8.29
Dna	57/106	5565	2759	2.01
		985×10^6	7833	119.5×10^6

Yukarıda, alt bölüm 3.4'te verilmiş olan ÖLT_FM_OLUŞTUR algoritması ile *m* objeli bir veri kümesi için elde edilecek bir FF'nin bileşenlerinin toplam sayısı $|FM| = 0.5 \times (m^2 - m)$'dir (Wang ve ark., 2007; Chen ve ark., 2008; Yao ve Zhao, 2009). Çizelge 3.4'te bu değer BT_FF sütununda gösterilmektedir. Çizelge 3.4'te bulunan $BT_FF_{\min}$ sütunu ise indirgenmiş $BT_FF(D)$'nin yani $BT_FF_{\min}(D)$'in bileşen sayısını göstermektedir. Burada Çizelge 3.4'e bakarak orta ve büyük ölçekli veri kümeleri için $BT_FF(D)$ ile $BT_FF_{\min}(D)$ arasındaki eleman sayısının oranının yüzlerden milyonlara kadar çıkabileceğini söyleyebiliriz.

Yukarıda denilenlerden anlaşılacağı gibi, $BT_FF(D)$'in elde edilip daha sonra indirgenmesi yerine $BT_FF_{\min}(D)$'in direk olarak doğruluk tablosundan elde edilmesi özellik seçme işleminin daha hızlı çalışması ve daha az hafızaya ihtiyaç duyması açısından elverişli olacaktır. Bu işlem ise denklem 3.13 tarafından üretilen her yeni

BTİ'nin $BT_FF(D)$ içerisinde var olan BTİ'ler ile karşılaştırılarak fazla BTİ'lerin tespit edilip silinmesiyle gerçekleştirilir. Bu yaklaşımı gerçekleştiren *BT_FF$_{min}$_OLUŞTUR(D)* algoritması Şekil 3.2'de gösterilmiştir.

```
BT_FFmin_OLUŞTUR(D) (SON(D), SOFF(D), M1,M2)
{
        BT_FF (D)={1}^N; | BT_FF (D) | =1
        For i=1 to M1
        {
                For k=1 to M2
                {
                        B=T_i^ON ⊕ T_k^OFF
                        BT_FF_İNDİRGE(D) (BT_FF (D), B)
                }
        }
        Return (BT_FFmin(D)= BT_FF (D))
}
```

Şekil 3.2. Bit tabanlı indirgenmiş fark fonksiyonunu oluşturan algoritma

BT_FF$_{min}$_OLUŞTUR(D) algoritmasındaki alt prosedür *BT_FF_İNDİRGE(D)* en son üretilen *B* BTİ'sini $BT_FF(D)$ kümesi içerisindeki diğer BTİ'lerle karşılaştırır. Bu karşılaştırma sonunda mümkün olabilecek üç durum vardır:

1. $BT_FF(D)$ kümesindeki BTİ'lerden $g_j \in BT_FF(D)$, $j=\in\{1, 2, \ldots, |BT_FF(D)|\}$ en az bir tanesi yeni oluşturulan B'yi yutar.
2. $BT_FF(D)$ kümesindeki BTİ'lerden $g_j \in BT_FF(D)$ hiç biri yeni oluşturan B'yi yutamaz.
3. $BT_FM(D)$ kümesindeki BTİ'lerden $g_j \in BT_FM(D)$ en az bir tanesi yeni oluşturan B tarafından yutulur.

Burada X ve Y BTİ'leri için eğer $X = X \& Y$ ise X Y'yi yutar ve eğer $Y = X \& Y$ ise Y X'i yutar. Bu yutma işlemiyle $BT_FF(D)$ fonksiyonunun indirgeniş halini bulan *BT_FF_İNDİRGE(D)* alt prosedürü Şekil 3.3'de gösterilmiştir.

```
BT_FF_İNDİRGE(D) (BT_FF(D), B)
{
    While j≤| BT_FF(D) |
    {
       (1) Select g_j ∈BT_FF(D)
       (2) G=B&g_j
       (3) If G=g_j
           Then
       (4) Return (BT_FF(D), | BT_FF(D) | )
           Else
       (5) If G=B
           Then
       (6) BT_FF(D) =BT_FF(D) – g_j ; | BT_FF(D)| = | BT_FF(D) | -1; j=j+1
    }
       (7) BT_FF(D) =BT_FF(D) ∪ B; | BT_FF(D) | = | BT_FF(D) | +1
Return (BT_FF(D), | BT_FF(D) | )
}
```

Şekil 3.3. Bit tabanlı fark fonksiyonunu indirgeyen alt algoritma

BT_FF_İNDİRGE(D) algoritmasında yeni üretilen BTİ *B*, $BT_FF(D)$ fonksiyonundaki tüm bileşenler ile karşılaştırılır. Eğer *B*, $BT_FM(D)$ kümesinde bulunan herhangi bir BTİ tarafından yutulursa $BT_FF(D)$ kümesi değişmez ve $BT_FF_{\min}(D)$ olarak geri döndürülür (BT_FF_İNDİRGE(D) algoritmasının 2., 3. ve 4. adımları). Fakat diğer durumda *B*'nin yuttuğu bütün bileşenler $BT_FF(D)$ fonksiyonundan silinir ve *B* bileşeni $BT_FF(D)$ fonksiyonuna eklenir ve yeni $BT_FF(D)$ fonksiyonu $BT_FF_{\min}(D)$ olarak geri döndürülür (BT_FF_İNDİRGE(D) algoritmasının 5. ve 6. adımları). Eğer bu iki durumdan her hangi biri gerçekleşmezse *B*,

$BT_FF(D)$ fonksiyonun yeni bileşene olarak fonksiyona eklenir ve yeni $BT_FF(D)$ kümesi $BT_FF_{\min}(D)$ olarak geri döndürülür (BT_FF_İNDİRGE(D) algoritmasının 7. adımı).

Her zaman $\left|BT_FF_{\min}(D)\right| \le \left|BT_FF(D)\right|$ olacağı için, $BT_FF_{\min}(D)$'in eleman sayısı $Q \le M_1 \times M_2$ olacaktır. Buna göre de denklemler 3.14 ve 3.15 aşağıdaki gibi baştan yazılabilir:

$$BT_FF(D) = \{B_q\}_{q=1}^{Q} \tag{3.16}$$

$$BT_FF_{\min}(D) = \bigcap_{q=1}^{Q} \left(\varphi_0 : B_q \in BT_FF(D)\right) \tag{3.17}$$

BT_FF$_{\min}$_OLUŞTUR(D) algoritmasında yapılan analizlere göre, algoritmanın polinomal en kötü durumlu (worst-case) hafıza ve zaman karmaşıklığı sırasıyla $O(M^2)$ ve $O(M^4)$ değerindedir.

Örnek 3.3: Çizelge 3.2' deki veri kümesinin $BT_FF_{\min}(D)$ fonksiyonunu elde edelim. Bunun için alt bölüm 3.3 gösterilen örnekte elde ettiğimiz $S_{ON}(D)$ ve $S_{OFF}(D)$ kümelerini kullanacağız. Aşağıdaki örnekte bir BTİ tarafından yutulan diğer BTİ'lerin üzeri çizilmiştir.

Giriş verileri:

$S_{ON}(D) = \{0100, 0000\}$

$S_{OFF}(D) = \{0011, 1101, 1000, 1011, 0111\}$

$BT_FF(D) = \{1111\}$

Bu değerlere göre $BT_FF_{\min}(D)$'in adım adım oluşturulması aşağıda anlatılmaktadır.

İterasyon 1: $T_{ON}^1 = 0100 \in S_{ON}(D)$, $BT_FF(D) = \{1111\}$

1.1. $T_{OFF}^1 = 0011$

$B = T_{ON}^1 \oplus T_{OFF}^1 = 0100 \oplus 0011 = 0111$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),0111)$

$BT_FF(D)$=$\{\{\sout{1111}\},0111\} \rightarrow \{0111\}$

1.2. $T_{OFF}^{2} = 1101$

$B = T_{ON}^{1} \oplus T_{OFF}^{2} = 0100 \oplus 1101 = 1001$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),1001)$

$BT_FF(D)$=$\{\{\sout{0111}\},1001\} \rightarrow \{0111\}$

1.3. $T_{OFF}^{3} = 1000$

$B = T_{ON}^{1} \oplus T_{OFF}^{3} = 0100 \oplus 1000 = 1100$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),1100)$

$BT_FF(D)$=$\{\{0111,1001\},0111\} \rightarrow \{0111,1001,1100\}$

1.4. $T_{OFF}^{4} = 1011$

$B = T_{ON}^{1} \oplus T_{OFF}^{4} = 0100 \oplus 1011 = 1111$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),1111)$

$BT_FF(D)$=$\{\{0111,1001,0111\},\sout{1111}\} \rightarrow \{0111,1001,1100\}$

1.5. $T_{OFF}^{5} = 0111$

$B = T_{ON}^{1} \oplus T_{OFF}^{5} = 0100 \oplus 0111 = 0111$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),0011)$

$BT_FF(D)$=$\{\{\sout{0111},1001,0111\},0011\} \rightarrow \{1001,1100,0011\}$

İterasyon 2: $T_{ON}^{2} = 0000 \in S_{ON}(D)$, $BT_FF(D) = \{1001,1100,0011\}$

1.1. $T_{OFF}^{1} = 0011$

$B = T_{ON}^{2} \oplus T_{OFF}^{1} = 0000 \oplus 0011 = 0011$

$BT_FF(D) = BT_FF_İNDİRGE(BT_FF(D),0011)$

$BT_FF(D)$=$\{\{1001,1100,0011\},\sout{0011}\} \rightarrow \{1001,1100,0011\}$

1.2. $T_{OFF}^{2} = 1101$

$B = T_{ON}^{2} \oplus T_{OFF}^{2} = 0000 \oplus 1101 = 1101$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),1101)$

$BT_FF(D)$=$\{\{1001,1100,0011\},\sout{1101}\} \rightarrow \{1001,1100,0011\}$

1.3. $T_{OFF}^{3} = 1000$

$B = T_{ON}^{2} \oplus T_{OFF}^{3} = 0000 \oplus 1000 = 1000$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),1000)$

$BT_FF(D)$={{~~1001~~,~~1100~~,0011},1000}$\rightarrow${0011,1000}

1.4. $T_{OFF}^{4} = 1011$

$B = T_{ON}^{2} \oplus T_{OFF}^{4} = 0000 \oplus 1011 = 1011$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),1011)$

$BT_FF(D)$={{0011,1000},~~1011~~}$\rightarrow${0011,1000}

1.5. $T_{OFF}^{5} = 0111$

$B = T_{ON}^{2} \oplus T_{OFF}^{5} = 0000 \oplus 0111 = 0111$

$BT_FF(D) = BT_FF_İNDİRGE(D)(BT_FF(D),0111)$

$BT_FF(D)$={{0011,1000},~~0111~~}$\rightarrow${0011,1000}

Çizelge 3.2.'de verilen tablodaki veri kümesinden elde edilen $BT_FF_{\min}(D)$ kümesi aşağıdaki gibidir.

$$BT_FF_{\min}(D) = \{0011,1000\}$$

BTİ formunda elde edilen $BT_FF_{\min}(D)$ denklem 3.11'deki veri yapısına göre ÖLT'ye çevrilerek elde edilen $FF_{\min}(D)$ aşağıdaki gibidir.

$$FF_{\min}(D) = \{0011,1000\} \xrightarrow{ÖLT/BTİ} \{A_3 \vee A_4\} \wedge A_1$$

Bu örnek FF tabanlı yaklaşımlar kullanılarak gerçekleştirilseydi yukarıda anlatılan alt bölüm 3.4'deki örnekte olduğu gibi, $|FF| = M_1 \times M_2 = 2 \times 5 = 10$ adet bileşenden meydana gelecekti. Fakat geliştirilen yöntemle sadece iki bileşenden oluşan bir $FF_{\min}(D)$ fonksiyonu elde edildi. Yani, geliştirilen yöntemle, bu veri kümesinin FF'inin oluşturulmasında gerekli olan hafıza miktarı 5 kat oranında azaltıldı.

3.6. Bit tabanlı ifade formunda bulunan indirgenmiş fark fonksiyonunun disjunktif normal forma çevrilmesi ile özelliklerin minimal alt kümelerinin elde edilmesi

3.6.1. Bit tabanlı ifadelerin genişletilmesi

Denklem 3.17'e göre konjunktif normal formda (CNF) bulunan $BT_FF_{\min}(D)$'in DNF'ye çevrilebilmesi için her BTİ'yi bileşenlerine ayırarak genişletmek gerekir. Yalnız şu unutulmamalıdır ki, her BTİ birden fazla 1 değerinde bit içerebilir ve bu 1 değerindeki her bit orijinal veri kümesinde bir özelliğe karşılık gelmektedir. Bu sebeplerden dolayı BTİ'ler gelişigüzel veya isteğe bağlı olarak bileşenlerine ayrılamazlar. Eğer böyle bir ayırma işlemi gerçekleşirse BTİ'lerde bulunan bilgilerde bozulma veya eksilme olabilir. Bu yüzden BTİ'ler öyle bir şekilde bileşenlerine ayrılmalıdırlar ki üzerlerinde taşıdıkları bilgiler bir kayba uğramamalıdır. Bu ayırma işlemi için BTİ'lerin bit ağırlık değerinden faydalanılacaktır. Bir BTİ'nin bit ağırlık değeri, BTİ'nin barındırdığı 1 değerinde olan bit sayısı kadardır ve bit ağırlık değeri $W(B_q)$ olarak gösterilir (Şirzat ve ark., 2011). Mesela, $B = \{0101010\}$ BTİ'sinin bit ağırlık değeri $W(B) = 3$'tür. Ayırma işlemi sonrasında $B_q \in BT_FF_{\min}(D)$ BTİ'lerin ayrılmış ifadeleri $E(B_q)$ şeklinde gösterilecektir. Ayrılmış $E(B_q)$'larının her birinin bit ağırlık değeri her zaman 1'dir.

$$E(B_q) = \{\Pr_i(B_q) : d_i = 1, i = 1,2,\ldots,N\} \tag{3.18}$$

Burada $\Pr_i(B_q)$, B_q'un i pozisyonuna göre projeksiyonu, d_i ise i. bitin değeridir. Bu şekilde bir genişleme sonrasında elde edilecek yeni BTİ'lerin hepsinin ağırlıkları $W(B) = 1$ değerinde olacaktır. Örnek olarak $B_1 = 001$ ve $B_2 = 0101$ BTİ'lerinin $E(B_1)$ ve $E(B_2)$ şeklinde genişletilmesi aşağıda gösterilmiştir.

$$E(B_1) = \{\Pr_i(001) \mid d_i = 1\} = \Pr_3(001) = \{001\}$$
$$E(B_2) = \{\Pr_i(0101) \mid d_i = 1\} = \{\Pr_2(0101), \Pr_4(0101)\} = \{0100, 0001\}$$

Denklem 3.18 için anlatılan BTİ'lerin genişlemesi dikkate alarak denklem 3.17 aşağıdaki gibi yeniden yazılabilir.

$$DNF_{BB}(D) = \bigvee_{q=1}^{Q} E(B_q) \tag{3.19}$$

Burada $B_q \in BT_FF_{\min}(D)$ ve | bitsel OR operatörüdür. Denklem 3.17 ile denklem 3.19 karşılaştırıldığı zaman, denklem 3.17'de kullanılan önerisel birleştirme operatörü yerine denklem 3.19'da bitsel OR operatörünün kullanıldığı görülmektedir. Çünkü iki önerisel ifadenin birleştirilmesi işlemi, genişletilmiş BTİ kümeleri için sadece bitsel OR operatörü kullanılarak gerçekleştirilebilir. Ayrıca, denklem 3.17'deki BTİ'lerin genişletilmesiyle $FF_{\min}(D)$'in DNF'i elde edilmiş olur. Bu yüzden denklem 3.19'da $BT_FF_{\min}(D)$ yerine $DNF_{BB}(D)$ yazılmıştır.

Örnek 3.4: Bu örnekte yukarıda örnek 3.3'de elde edilmiş olan $BT_FM_{\min}(D)$ kümesi $DNF(D)$'ye çevrilerek Çizelge 3.2.'de verilen veri kümesinin ÖMAK'larının tamamının bulunması adım adım anlatılmaktadır.

1. $BT_FF_{\min}(D) = \{B_1, B_2\} = \{0011{,}1000\}$ kümesi denklem 3.18'e göre aşağıdaki gibi genişletilir.
   $E(B_1) = \{\Pr_i(0011) \mid d_i = 1\} = \{\Pr_3(0011), \Pr_4(0011)\} = \{0010{,}0001\}$
   $E(B_2) = \{\Pr_i(1000) \mid d_i = 1\} = \Pr_3(1000) = \{1000\}$
2. Denklem 3.19 kullanılarak veri kümesinin $DNF_{BB}(D)$'si elde edilir.
   $DNF_{BB}(D) = \bigcup_{q=1}^{2} E(B_q) = E(B_1) \mid E(B_2) = \{0010{,}0001\} \mid \{1000\} = \{1010, 1001\}$
3. Elde edilmiş olan BTİ formundaki $DNF_{BB}(D)$ aşağıdaki veri yapısına uygun olacak şekilde ÖLT'ye çevrilerek Çizelge 3.2.'de gösterilen veri kümesinin ÖMAK'ları elde edilir. $DNF_{BB}(D)$ kümesindeki her bir elemanın 1 değerine sahip her biti Çizelge 3.2.'deki veri kümesinin bir özelliğini göstermektedir.

Struct_DNF_ÖMAK{Unsig A1:1; Unsig A2:1; Unsig A3:1; UnsigA4:1} (3.20)

Elde $DNF_{BB}(D)$ kümesindeki elemanlar denklem 3.20'deki veri yapısına göre aşağıdaki gibi ÖMAK'lara çevrilir.

$$\ddot{O}MAK = \{1010, 1001\} \xrightarrow{\text{DNF/ÖMAK}} \{\{\text{A1, A3}\}, \{\text{A1, A4}\}\}$$

Sonuç olarak $\{A_1, A_3\}$ ve $\{A_1, A_4\}$ kümeleri Çizelge 3.2.'de verilen veri kümesinin ÖMAK'larıdır.

3.6.2 Bit tabanlı ifadelerden oluşan indirgenmiş fark fonksiyonun disjunktif normal forma çevrilmesi esnasında gereksiz terimlerin oluşmasının engellenmesi

Yapılan deneylerde, CNF'den DNF'ye çevirme işlemi sırasında gerekli terimlerden çok lüzumsuz terimlerin oluştuğu görülmüştür. Mesela, 10 tane terimi ve her terimin 5 literali bulundurduğu bir CNF'nin DNF'ye çevirme işleminde, ilk iki terimin çarpılması sonucu oluşacak lüzumsuz terim sayısı 1 iken çevrilme işlemi sonunda bu sayı $Y^{Z-1} = 5^{10-1} = 5^9 = 1953125$ kadar yükselebilir. Bu örneğe bakarak gereksiz terimlerin oluşmasının engellenmesinin ne kadar önemli olduğunu söyleyebiliriz. $DNF_{BB}(D)$'nin oluşturulması aşamasında gereksiz terimlerin tespit edilmesi ve bu tespit edilen terimlerin $DNF_{BB}(D)$'den çıkartılması denklem 3.19'ın iteratif olarak gerçekleştirilmesiyle yapılır.

$$\left.\begin{aligned} F_0 &= \{\{0\}^N\} \\ F_q &= F_{q-1} \mid E(B_q)\ \forall \text{q} = 1,2,\ldots \text{Q} \\ DNF_{BB} &= F_q \end{aligned}\right\} \quad (3.21)$$

Gereksiz terimlerin engellenmesi için kullandığımız $F_q = F_{q-1} \mid E(B_q)$ formülünün hesaplanmasında F_{q-1} ve $E(B_q)$ arasında olan aşağıdaki ilişki kullanılır. $F_{q-1} = A \mid B$' yi $F_{q-1} = \{A, B\}$ ve $E(B_q) = C \mid D$'yi ise $E(B_q) = \{C, D\}$ olarak ele alalım. Burada her $A, B \in F_{q-1}$ teriminin bit ağırlık değeri $1 \le k \le n$ arasındadır ve her $C, D \in E(B_q)$ ise ağırlık değeri bir olan ayrılmış terimdir. $A, B \in F_{q-1}$ değerlerinin $C, D \in E(B_q)$ değerleri ile lojik birleşme işlemi yapılması sonrasında elde edilecek olan

sonuçların gerekli veya gereksiz terimler olup olmadığı aşağıdaki kurallara göre belirlenir. Bu kurala göre belirlenen terimlere erken tespit edilen lüzumsuz terimler adı verilir.

$$\left.\begin{array}{l}\text{Eğer } A|C=A \text{ veya } A|D=A \text{ ise}\\ F_{q-1}|E(B_q)=(A\vee B)|(C\vee D)\rightarrow A\vee BC\vee BD\\ \text{Eğer } B|C=B \text{ veya } B|D=B \text{ ise}\\ F_{q-1}|E(B_q)=(B\vee A)|(C\vee D)\rightarrow B\vee AC\vee AD\\ \text{Eğer } A|C=A \text{ veya } A|D=A \text{ ve } B|C=B \text{ veya } B|D=B \text{ ise}\\ F_{q-1}|E(B_q)=A\vee B\end{array}\right\} \quad (3.22)$$

Denklem 3.22'ye göre F_{q-1}'de bulunan en az bir tane bileşen $a_i \in F_{q-1}$, $a_i|E(B_q)=a_i$ eşitliğini sağlıyorsa $F_q=F_{q-1}|E(B_q)$ işlemi indirgenebilir. Bu durumu tespit eden denklem aşağıda gösterilmiştir.

$$\forall a_i \in F_{q-1}: a_i \,\&\, B_q \neq \{0\}^n \rightarrow a_i|E(B_q)=a_i \quad (3.23)$$

Yukarıdaki denkleme göre $a_i \in F_{q-1}$ ve $a_i \,\&\, B_q \neq 0$ ise $a_i|B_q$ işleminden bir tane bit dizisi oluşur ve bu bit dizisi bu yolla oluşan diğer bit dizilerini yutar. Bu yüzden $a_i \,\&\, B_q \neq \{0\}^N$ şartını sağlayan bütün $a_i \in F_{q-1}$ bileşenleri sonuç değerinin bir parçasıdır ve $E(B_q)$ ile işleme sokulmalarına da gerek yoktur. Bu şekilde oluşan parçaları toplamak için aşağıda tanımlanan V_{q1} kümesi kullanılır (Hacıbeyoğlu ve ark., 2011).

$$V_{q1}=\left\{a_i \in F_{q-1}\,|\,x \,\&\, B_q \neq \{0\}^n\right\} \quad (3.24)$$

F_{q-1} kümesinin kalan kısmı ile aşağıdaki denklemde gösterilen V_{q2} kümesi oluşturulur (Hacıbeyoğlu ve ark., 2011).

$$V_{q2}=F_{q-1}-V_{q1} \quad (3.25)$$

Denklem 3.25'den elde edilen $E(B_q)$ kümesi aşağıdaki denklemde gösterildiği gibi V_{q2} kümesi ile lojik OR işlemine sokulur (Hacıbeyoğlu ve ark., 2011).

$$T_q = V_{q2} \mid E(B_q) \tag{3.26}$$

Geç tespit edilen gereksiz terimler denklem 3.26'ün çalıştırılması ile belirlenebilir. Bu işlem sonrasında oluşan terimlerin gereksiz terim olup olmadıkları aşağıda gösterilen denkleme göre belirlenir. Denklem 3.27 kullanılarak tespit edilen gereksiz terimler $DNF_{BB}(D)$ kümesinden silinir (Hacıbeyoğlu ve ark., 2011).

$$T_q = T_q \cup w_{ji} \Leftrightarrow \bar{\exists} v \in V_{q1} : v \mid w_{ji} = v \tag{3.27}$$

Burada $w_{ij} = v_j \mid e_i$, $v_j \in V_{q2}$, $e_i \in E(B_q)$, $j \in 1,2,\ldots,\left|V_{q2}\right|$ ve $i \in 1,2,\ldots,\left|E(B_q)\right|$'dir.

Örnek 3.5: Bu örnekte Jensen ve Shen (2007)'den alınan indirgenmiş FF üzerinde erken ve geç tespit edilen gereksiz terimlerin oluşması engellenerek DNF'ye çevirme işlemi gerçekleştirilecektir.

$$FF_{\min}(D) = (\mathrm{a} \vee \mathrm{b} \vee \mathrm{c} \vee \mathrm{f}) \wedge (\mathrm{b} \vee \mathrm{d}) \wedge (\mathrm{a} \vee \mathrm{d} \vee \mathrm{e} \vee \mathrm{f}) \wedge (\mathrm{d} \vee \mathrm{e})$$

Bu fonksiyonu, BTİ'lerden oluşan bir kümeye çevirmek için aşağıda tanımlanan veri yapısı kullanılır.

STRUCT_BTİ{Unsig a:1;Unsig b:1;Unsig c:1;Unsig d:1;Unsig e:1;Unsig f:1} (3.28)

Denklem 3.28'deki yapı kullanılarak, $FF_{\min}$ fonksiyonu $BT_FF_{\min}(D)$ kümesine aşağıdaki gibi çevrilir.

$$\mathrm{B}_1 = (\mathrm{a} \vee \mathrm{b} \vee \mathrm{c} \vee \mathrm{f}) \overset{\text{STRUCT_BTİ}}{\rightarrow} 111001$$

$$\mathrm{B}_2 = (\mathrm{b} \vee \mathrm{d}) \overset{\text{STRUCT_BTİ}}{\rightarrow} 010100$$

$B_3 = (a \vee d \vee e \vee f) \overset{STRUCT_BTİ}{\rightarrow} 100111$

$B_4 = (d \vee e) \overset{STRUCT_BTİ}{\rightarrow} 000110$

$BT_FM_{\min} = \{B_1, B_2, B_3, B_4\} = \{111001, 010100, 100111, 000110\}$

$BT_FM_{\min}(D)$ fonksiyonun gereksiz bileşenlerinin oluşması engellenerek DNF'ye çevrilme işlemi aşağıda anlatılmaktadır. Bu işlem 4 iterasyon sonucunda gerçekleşmektedir.

İterasyon 1: q=1; B_1=111001; F_0={$\{0\}^6$}={000000}; V_{q1}=∅; V_{q2}={$\{0\}^N$}

1.1. F_0 kümesi içindeki bütün $a_i \in F_0$'lar B_1 ile lojik *AND* işlemine sokulur: a_1 & B_1 = 000000 & 111001 = $\{0\}^6$

1.2. F_0 kümesi denklemler 3.24 ve 3.25'ye göre aşağıdaki gibi iki kümeye ayrılır:

V_{11}={$a_1 \in F_0 : a_1$& $B_1 \neq \{0\}^n$ }

V_{12}=F_0–V_{11}

V_{11}=∅

V_{12}=F_0-V_{11}={a_1}= {000000}

1.3. Denklem 3.26'daki $T_1 = V_{12} \mid E(B_1)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{11} kümesinin elemanlarıyla karşılaştırılır. V_{11} kümesinin elemanları tarafından yutulan elemanlar T_1 kümesinden silinir.

E(B_1) ={100000, 010000, 001000, 000001}

$T_1 = V_{12} \mid E(B_1)$ = {000000}|{100000, 010000, 001000, 000001}={100000, 010000, 001000, 000001}.

T_1 kümesindeki hiçbir eleman V_{11} kümesindeki elemanlar tarafından yutulmamıştır.

1.4. $F_1 = V_{11} \cup T_1$ kümesi hesaplanır.

$F_1 = \varnothing \cup$ {100000, 010000, 001000, 000001} = {100000, 010000, 001000, 000001}

İterasyon 2: q=2; B_2=010100; F_1={100000, 010000, 001000, 000001}; V_{21}=∅; V_{22}={$\{0\}^N$}

2.1. F_1 kümesi içindeki bütün $a_i \in F_1$'lar B_2 ile lojik *kesişme* işlemine sokulur:

a_1 & B_2 =100000 & 010100 = $\{0\}^6$

a_2 & B_2 = 010000 & 010100 ≠ $\{0\}^6$

a_3 & B_2 = 001000 & 010100 = $\{0\}^6$

a_4 & B_2 = 000001 & 010100 = $\{0\}^6$

2.2. F_1 kümesi denklemler 3.24 ve 3.25'ye göre aşağıdaki gibi iki kümeye ayrılır:

V_{21}={$a_i \in F_1$: a_i& $B_2 \neq \{0\}^n$}

V_{22}= $F_1 - V_{21}$

V_{21}={a_2}={010000};

V_{22}=F_1-V_{11} = {a_1, a_3, a_4} = {100000, 001000, 000001}

2.3. Denklem 3.26'daki $T_2 = V_{22} \mid E(B_2)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{21} kümesinin elemanlarıyla karşılaştırılır. V_{21} kümesinin elemanları tarafından yutulan elemanlar T_2 kümesinden silinir.

E(B_2) = {010000, 000100}

$T_2 = V_{22} \mid E(B_2)$ = {100000, 001000, 000001} | {010000, 000100} = {~~110000~~, 100100, ~~011000~~, 001100, ~~010001~~, 000101}.

T_2 kümesindeki 110000, 011000 ve 010001 elemanları sırasıyla V_{21} kümesindeki 010000 elemanı tarafından yutulur ve T_2 kümesinden silinir.

2.4. $F_2 = V_{21} \cup T_2$ kümesi hesaplanır.

F_2 = 010000 ∪ {100100, 001100, 000101}={010000, 100100, 001100, 000101}

İterasyon 3: q=3, B_3=100111; F_2={010000, 100100, 001100, 000101}; V_{31}=∅; V_{32}={$\{0\}^N$}

3.1. F_2 kümesi içindeki bütün $a_i \in F_2$'lar B_3 ile lojik *kesişme* işlemine sokulur:

a_1&B_3 =010000&100111=$\{0\}^6$

a_2&B_3 =100100&100111≠{0}^{6}

a_3&B_3 =001100&100111≠{0}^{6}

a_4&B_3 =000011&100111≠{0}^{6}

3.2. F_2 kümesi denklemler 3.24 ve 3.25'ye göre aşağıdaki gibi iki kümeye ayrılır:

V_{31}={$a_i \in F_2$: a_i& B_3≠{0}^{n}

$V_{32}= F_2 - V_{31}$

V_{31}={a_2, a_3, a_4}={100100, 001100, 000101}

V_{32}=F_2-V_{31}={a_1 }={010000}

3.3. Denklem 3.26'daki $T_3 = V_{32} \mid E(B_3)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{31} kümesinin elemanlarıyla karşılaştırılır. V_{31} kümesinin elemanları tarafından yutulan elemanlar T_3 kümesinden silinir.

E(B_3)={100000, 000100, 000010, 000001}

$T_3 = V_{32} \mid E(B_3)$ = {010000} | {100000, 000100, 000010,000001}={110000, 010100, 010010, 010001}.

T_3 kümesindeki hiçbir eleman V_{31} kümesindeki elemanlar tarafından yutulmamıştır.

3.4. $F_3 = V_{31} \cup T_3$ kümesi hesaplanır.

$F_3 = V_{31} \cup T_3$= {100100, 001100, 000101} ∪ {110000, 010100, 010010, 010001} = {100100, 001100, 000101, 110000, 010100, 010010, 010001}

İterasyon 4: q=4, B_4=001100; F_3={100100, 001100, 000101, 110000, 010100, 010010, 010001}; V_{41}=∅; V_{42}={{0}^{N}}

4.1. F_2 kümesi içindeki bütün $a_i \in F_2$'lar B_3 ile lojik *kesişme* işlemine sokulur:

a_1&B_4 =100100&000110≠{0}^{6}

a_2&B_4 =001100&000110≠{0}^{6}

a_3&B_4 =000101&000110≠{0}^{6}

a_4&B_4 =110000&000110={0}^{6}

a_5&B_4 =010100&000110≠{0}^{6}

a_6&B_4 =010010&000110={0}^{6}

a_7&B_4 =010001&000110={0}^{6}

4.2. F_3 kümesi denklemler 3.24 ve 3.25'ye göre aşağıdaki gibi iki kümeye ayrılır:

$V_{41}=\{a_i \in F_3: a_i \& B_4 \neq \{0\}^n$

$V_{42}= F_3 - V_{41}$

$V_{41}=\{a_1, a_2, a_3, a_5, a_6\}=\{100100, 001100, 000101, 010100\}$

$V_{42}=F_3-V_{41}=\{a_4,a_7\}=\{110000, 010001, 010001\}$

4.3. Denklem 3.26'daki $T_3 = V_{32} \mid E(B_3)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{31} kümesinin elemanlarıyla karşılaştırılır. V_{31} kümesinin elemanları tarafından yutulan elemanlar T_3 kümesinden silinir.

$E(B_4)= \{001000, 000100\}$

$T_3 = V_{32} \mid E(B_3) =$ {110000, 010010, 010001} | {001000, 000100} = {111000, ~~110100~~, ~~011001~~, ~~010101~~, 011010, 010110}.

T_4 kümesindeki 110100, 011001, 010101 elemanları sırasıyla V_{41} kümesindeki 100100, 010001 elemanları tarafından yutulur ve T_4 kümesinden silinir.

4.4. $F_4 = V_{41} \cup T_4$ kümesi hesaplanır.

$F_4 = V_{41} \cup T_4$ = {100100, 001100, 000101,010100} $\cup$ {111000, 011010, 010110} = {100100, 001100, 000101,010100, 111000, 011010, *010110*}

Denklem 3.28'deki STURCT_BTİ veri yapısı dikkate alınarak elde edilen F_4 kümesi ÖMAK'lara aşağıdaki gibi çevrilir.

ÖMAK= {100100, 001100, 000101,010100, 111000, 011010, 010110} $\xrightarrow{\text{STRUCT_BTİ}}$ {{a,d}, {c,d}, {d,f}, {b,d}, {a,b,c}, {b,c,f}, {b,c,e}}

Aşağıda açıklanacak olan *ÇEVİR_BT_FM(D)* algoritması bu bölümde anlatılan örneğin daha da detaylandırılmış halini göstermektedir. Yukarıdaki örneğin 1. ve 2. adımları ÇEVİR_BT_FM(D) algoritmasının gövde kısmında, 3. ve 4. adımları ise ÇEVİR_BT_FM(D) algoritmasının içerisinde çağrılacak *GENERATE_PRODUCTS* adı verilen alt algoritmada anlatılacaktır. GENERATE_PRODUCTS alt algoritması $T_q = V_{q2} \mid E(B_q)$ ve $F_q = V_{q1} \cup T_q$ denklemlerindeki değerleri hesaplayacaktır. Fakat

gereksiz terimlerin yakalanabilmesi için $T_q = V_{q2} \mid E(B_q)$ işlemi sonrası oluşturulan her yeni terimin V_{q1} kümesindeki elemanlar ile karşılaştırılması gerekmektedir. Eğer V_{q1} bu terimlerden herhangi birisini yutarsa o terim T_q kümesinden silinir. Sonuç olarak sadece gerekli olan terimler T_q kümesi içerisinde kalır ve bu T_q kümeleri de iteratif olarak işlenerek veri kümesinin ÖMAK'ları oluşturulur.

3.7. Boole fonksiyon yaklaşımlı özellik seçme yönteminin çok değerli veri kümelerine uyarlanması

Yukarıda alt bölüm 3.2'de çokdeğerli bir veri kümesinin özelliklerinin ikili değerlerle kodlanması anlatılmıştı. Bu bölümde de Çizelge 3.5'de gösterilen veri kümesinin (Komorowski ve ark., 1999) ÖMAK'ları yukarıda anlatılan yöntem kullanılarak elde edilecektir.

Çizelge 3.5. Çok değerli özelliklere sahip örnek veri kümesi

Objeler	A_1	A_2	A_3	A_4	D
U_1	MBA	Medium	Yes	Excellent	Accept
U_2	MBA	Low	Yes	Neutral	Reject
U_3	MCE	Low	Yes	Good	Reserve 2
U_4	MSc	High	Yes	Neutral	Accept
U_5	MSc	Medium	Yes	Neutral	Reserve 1
U_6	MSc	High	Yes	Excellent	Accept
U_7	MBA	High	No	Good	Accept
U_8	MCE	Low	No	Excellent	Reject

Çizelge 3.5'de gösterilen örnek veri kümesi 8 adet objeden, 4 adet şart özelliklerinden ve D karar özelliğinden oluşmaktadır. Burada A_1, A_2 ve A_4 şart özelliklerinin her biri 3 farklı değer, A_3 şart özelliği 2 farklı değer ve D karar özelliği 4 farklı değer almaktadır. Bu sebepten A_1, A_2 ve A_4 şart özelliklerinin her birinin değerleri $\lceil \log_2 3 \rceil = 2$ bit ile, A_3 şart özelliğinin değerleri $\lceil \log_2 2 \rceil = 1$ bit ile ve D karar özelliğinin değerleri $\lceil \log_2 (4+1) \rceil = 3$ bit ifade edilir. D karar özelliği için 000 kodlu ifade Çizelge 3.5.'de gösterilmeyen objeler için ayrılmıştır. Böylece, şart özellikleri ve karar özelliği aşağıdaki gibi kodlanır.

$A_1 \rightarrow A_{11}A_{12}$: İki bitten oluşur.

MBA → 00

MCE → 01

MCS → 10

$A_2 \rightarrow A_{21}A_{22}$: İki bitten oluşur.

Medium → 00

Low → 01

High → 10

$A_3 \rightarrow A_{31}$: Bir bitten oluşur.

Yes → 0

No → 1

$A_4 \rightarrow A_{41}A_{42}$: İki bitten oluşur.

Excellent → 00

Neutral → 01

Good → 10

$D \rightarrow D_1D_2D_3$: Üç bitten oluşur.

Accept → 001

Reject → 010

Reserve1 → 011

Reserve 2 → 100

Çizelge 3.5'deki veri kümesinin özellikleri yukarıdaki gibi kodlandıktan sonra kodlar birleştirilerek mintermler elde edilir. Sonuç olarak, elde edilen mintermler kullanılarak Çizelge 3.5'deki veri kümesinin doğruluk tablosu aşağıdaki gibi oluşturulur.

Çizelge 3.6. Çizelge 3.5.'de gösterilen çok değerli özelliklere sahip veri kümesinin doğruluk tablosu

Mintermler	$A_{11}A_{12}$	$A_{21}A_{22}$	A_3	$A_{41}A_{42}$	$D_1D_2D_3$
T_1	0 0	0 0	1	0 0	0 0 1
T_2	0 0	0 1	1	0 1	0 1 0
T_3	0 1	0 1	1	1 0	1 0 0
T_4	1 0	1 0	1	0 1	0 0 1
T_5	1 0	0 0	1	0 1	0 1 1
T_6	1 0	1 0	1	0 0	0 0 1
T_7	0 0	1 0	0	1 0	0 0 1
T_8	0 1	0 1	0	0 0	0 1 0

Çizelge 3.6'daki gibi kodlanmış karar özelliğinin her bir biti için $S_{ON}(D)$ ve $S_{OFF}(D)$ kümeleri yukarıda alt bölüm 3.3'de anlatılan örnekteki gibi elde edilir. Elde edilen $S_{ON}(D)$ ve $S_{OFF}(D)$ kümeleri aşağıdaki gibidir:

$$S_{ON}(D_1) = \{T_3\} = \{0101110\}$$

$$S_{OFF}(D_1) = \{T_1, T_2, T_4, T_5, T_6, T_7, T_8\} = \begin{Bmatrix} 0000100, 0001101, 1010101, 1000101, 1010100, \\ 0010010, 0101000 \end{Bmatrix}$$

$$S_{ON}(D_2) = \{T_2, T_5, T_8\} = \{0001101, 1000101, 0101000\}$$

$$S_{OFF}(D_2) = \{T_1, T_3, T_4, T_6, T_7\} = \{0000100, 0101110, 1010101, 1010100, 0010010\}$$

$$S_{ON}(D_3) = \{T_1, T_4, T_5, T_6, T_7\} = \{0000100, 1010101, 1000101, 1010100, 0010010\}$$

$$S_{OFF}(D_3) = \{T_2, T_3, T_8\} = \{0001101, 0101110, 0101000\}$$

Elde edilen $S_{ON}(D)$ ve $S_{OFF}(D)$ kümeleri, $\{A_1, A_2, A_3, A_4\}$ özellikler kümesinin ÖMAK'larını elde etmek için kullanılacaktır. Yalnız ikideğerli veri kümelerinden farklı olarak çokdeğerli veri kümelerini işleyebilmek için aşağıdaki özelliklerin kullanılması gerekmektedir.

Çizelge 3.6'da da görüldüğü gibi A_1 özelliği $A_{11}A_{12}$ şeklinde 2 bitle, A_2 özelliği $A_{21}A_{22}$ şeklinde 2 bitle, A_4 özelliği $A_{41}A_{42}$ şeklinde 2 bitle ve A_3 özelliği bir bit ile ifade edilir. Yani A_1, A_2 ve A_4 özelliklerinin iki adet alt özellikleri bulunmaktadır. Böyle bir kodlama sonrasında Çizelge 3.6.'da bulanan her bir şart özelliğinin alt özellik sütunları aynı özelliğin diğer alt özellik sütunları ile ilişkilendirilmek yani beraber kullanılmak zorundadır. Böylece denklem 3.16'yı kullanarak N adet özellikten oluşan bir veri kümesinin $BT_FF(D)$'sini bulunmaya çalışılırsa lojik *AND* işlemine sokulacak olan bit dizilerindeki toplam bit sayısının N'den fazla olacağı görülür. Yani alt bölüm 3.3'deki örnekte gösterildiği gibi, her zaman bir bit bir özelliğe karşılık gelmemektedir. Her özelliğin birden fazla bit ile ifade edildiği çokdeğerli veri kümelerinden elde edilen bu bit dizilerine *kod tabanlı terim* (KTT) adı verilir (Şirzat ve ark., 2011). Çokdeğerli veri kümelerinin ÖMAK'larının KTT'ler kullanılarak bulunabilmesi için bu KTT'lerin uygun BTİ'lere çevrilmesi gereklidir.

Çizelge 3.6'daki doğruluk tablosunun her satırı $A_{11}A_{12}A_{21}A_{22}A_3A_{41}A_{42}$ formatındaki mintermlerden oluşmaktadır. Diğer taraftan Çizelge 3.5'de ki veri

kümesinin her satırında $A_1A_2A_3A_4$ formatında özellik değerleri bulunmaktadır. Bu formatlar arasındaki ilişkiyi belirleyebilmek için aşağıda gösterilen veri yapıları kullanılacaktır.

Struct_KTT_CIZELGE_3.6 {Unsig SF_1:2; Unsig SF_2:2; Unsig SF_3:1; Unsig SF_4:2;} (3.29)

Struct_BTİ_CIZELGE_3.5{Unsig A_1:1; Unsig A_2:1; Unsig A_3:1; Unsig A_4:1} (3.30)

Struct_KTT_CIZELGE_3.6 ve *Struct_BTİ_CIZELGE_3.5* veri yapıları arasındaki ilişkiye göre $A_{11}A_{12}$, $A_{21}A_{22}$, A_3 ve $A_{41}A_{42}$ alt özellikleri sırasıyla SF_1, SF_2, SF_3 ve SF_4 olarak ifade edilir. Böylece $A_{11}A_{12}A_{21}A_{22}A_3A_{41}A_{42}$ formatındaki bir ikili kodlanmış ifade $A_1A_2A_3A_4$ formatına aşağıdaki gibi çevrilebilir.

Eğer $KTT.SF_j = 0$ ise $BTİ.A_j = 0$
Eğer $KTT.SF_j \neq 0$ ise $BTİ.A_j = 1$ (3.31)

Örnek olarak $A_{11}A_{12}A_{21}A_{22}A_3A_{41}A_{42}$ formatındaki $KTT = 0001001$ ifadesini $A_1A_2A_3A_4$ formatına çevirelim. *Struct_KTT_CIZELGE_3.6* veri yapısına göre A_1, A_2 ve A_4 özellikleri iki bitten ve A_3 özelliği ise 1 bitten oluşur. Buna göre verilen $KTT = 0001001$ ifadesi dörde bölünür ve $T.SF_1 = 00$, $T.SF_2 = 01$, $T.SF_3 = 0$ ve $T.SF_4 = 01$ alt ifadeleri elde edilir. Buradan denklem 3.30 kabul edilerek elde edilen alt ifadelerin karşılıkları $A_1 = 0$, $A_2 = 1$, $A_3 = 0$ ve $A_4 = 1$ olarak bulunur. Sonuç olarak elde edilen özelliklerin ikili değerleri birleştirilerek $A_1A_2A_3A_4$ formatında 0101 değeri elde edilir.

Sonuç olarak çok değerli şart ve karar özelliklerine sahip olan bir veri kümesinin işlenmesi ikideğerli şart ve karar özelliklerine sahip olan bir veri kümesinin işlenmesinden farklıdır. $m = \lfloor \log_2 V_d + 1 \rfloor$ bit ile ifade edilen bir D karar özelliğinin her bir $D_1, D_2, \ldots D_m$ biti için elde edilen $S_{ON}(D_j)$ ve $S_{OFF}(D_j)$ $j = \in \{1,2,\ldots m\}$ ayrı ayrı işlenir. Her bir işlemden sonra elde edilen KTT'ler denklem 3.31 kullanılarak BTİ'lere

çevrilir. Elde edilen BTİ'ler kullanılarak veri kümesinin $FM(D_1, D_2, \ldots D_m)$'si aşağıdaki denklemde gösterildiği gibi elde edilir:

$$FM(D_1, D_2, \ldots D_{m)} = \bigcup_{h=1}^{m} FM(D_h) \tag{3.32}$$

Yukarıda Şekil 3.2'de gösterilen *BT_FM$_{min}$_OLUSTUR* algoritması geliştirilerek çok değerli bir veri kümesinin indirgenmiş bit tabanlı FM'sini bulan *CD_BT_FM$_{min}$_OLUSTUR* algoritması aşağıda Şekil 3.4'te gösterilmektedir:

```
CD_BT_FMmin_OLUSTUR (D)({SON(Dh), SOFF(Dh), Mh1, Mh2}, h=1..m )
{
    FM(D)={1}^N; |FM(D) |=1
        (1). For h=1 to m
        {
            SON(D) =SON(Dh);
            SOFF(D) =SOFF(Dh);
            M1=Mh1; M2=Mh2;
            (2). For i=1 to M1-1
            {
                (3).  For k=1 to M2
                {
                    B= B=T_i^ON ⊕ T_k^OFF
                    B'yi KKT'den BTİ'ye çevir
                    Reduce_BT_FM(D) (BT_FM(D), B)
                }
            }
        }
    Return (CSmin(D)=CS(D))
}
```

Şekil 3.4. $D_1, D_2, \ldots, D_m$ karar fonksiyonları için bit tabanlı indirgenmiş fark fonksiyonunu bulan algoritma

Şekil 3.4'de gösterilen *CD_BT_FM$_{min}$_OLUSTUR* algoritmasında $S_{ON}(D_h)$ ve $S_{OFF}(D_h)$ kümelerinin eleman sayıları M_{h1} ve M_{h2} olarak gösterilmiştir. Burada $h \in \{1,2,\ldots m\}$.

Genel olarak söylemek gerekirse, *CD_BT_FM$_{min}$_OLUSTUR* algoritması *BT_FM$_{min}$_OLUSTUR* algoritmasının m kere çalıştırılmasıdır. Burada m değeri N değerine bağlı olmadığı için *CD_BT_FM$_{min}$_OLUSTUR* algoritmasının en kötü hafıza ve zaman karmaşıklığı *BT_FM$_{min}$_OLUSTUR* algoritması kadar olacaktır. *CD_BT_FM$_{min}$_OLUSTUR* algoritması adım adım aşağıdaki örnekte gösterildiği gibi çalışmaktadır.

Örnek 3.6. : Bu örnekte Çizelge 3.5'deki veri kümesinin Çizelge 3.6'daki doğruluk tablosu kullanılarak ÖMAK'ları elde edilecektir. Örneğin, çözüm aşamasında D_1, D_2 ve D_3 karar fonksiyonlarının işlenmesinde örnek 3.3'deki D karar fonksiyonun işlenme yolu takip edilecektir. Aşağıdaki örnekte bir BTİ tarafından yutulan BTİ'lerin üzeri çizilmiştir.

D1 karar fonksiyonun işlenmesi: (1). For döngüsünün 1. iterasyonu.

Giriş verileri:

$S_{ON}(D_1) = \{0101110\}$

$S_{OFF}(D_1) = \{0000100, 0001101, 1010101, 1000101, 1010100, 0010010, 0101000\}$

$BT_FM = \{\{1\}^4\} = 1111$

1. T_1^{ON} = 0101110 ∈ $S_{ON}(D_1)$ 'nin $S_{OFF}(D_1)$ kümesinin elemanları ile işleme sokulması. (2). numalaralı For döngüsünün 1. iterasyonu ve (3). numaralı For döngüsünün 1., 2., 3., 4., 5., 6. ve 7. iterasyonları.

   1.1. $B = T_1^{ON} \oplus T_1^{OFF} = 0101110 \oplus 0000100 = 0101010$

   B = φ4 : B = φ4 : (01 01 0 10) = 1101

   Burada φ4 Struct_KTT_CIZELGE_3.6, Struct_BTİ_CIZLEGE_3.5 ve denklem 3.31'i kullanarak B'nin KKT'den BTİ'ye çevrilmesini sembolize etmektedir. BT_FM(D)= Reduce_BT_FM(D) (BT_FM(D), B) = {1101, ~~1111~~} = {1101}

1.2. $B = T_1^{ON} \oplus T_2^{OFF} = 0101110 \oplus 0001101 = 0100011$

B = φ4 : B = φ4 : (01 00 0 11) = 1001

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {1001, ~~1101~~} = {1001}

1.3. $B = T_1^{ON} \oplus T_3^{OFF} = 0101110 \oplus 1010101 = 1111011$

B = φ4 : B = φ4 : (11 11 0 11) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {1001}} = {1001}

1.4. $B = T_1^{ON} \oplus T_4^{OFF} = 0101110 \oplus 1000101 = 1101011$

B = φ4 : B = φ4 : (11 01 0 11) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {1001}} = {1001}

1.5. $B = T_1^{ON} \oplus T_5^{OFF} = 0101110 \oplus 1010100 = 1111010$

B = φ4 : B = φ4 : (11 11 0 10) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {1001}} = {1001}

1.6. $B = T_1^{ON} \oplus T_6^{OFF} = 0101110 \oplus 0010010 = 0111100$

B = φ4 : B = φ4 : (01 11 1 00) = 1110

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {1110, {1001}} = {1001, 1110}

1.7. $B = T_1^{ON} \oplus T_7^{OFF} = 0101110 \oplus 0101000 = 0000110$

B = φ4 : B = φ4 : (00 00 1 10) = 0011;

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {0011, {1001, 1110}}= {1001, 1110, 0011}

D2 karar fonksiyonun işlenmesi: (1). For döngüsünün 2. iterasyonu.

Giriş verileri:

$S_{ON}(D_2) = \{0001101, 1000101, 0101000\}$

$S_{OFF}(D_2) = \{0000100, 0101110, 1010101, 1010100, \ 0010010\}$

$BT_FM = \{1001, 1110, 0011\}$

1. T_1^{ON} = 0001101 ∈ $S_{ON}(D_2)$'nin $S_{OFF}(D_2)$ kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 1. iterasyonu ve (3). numaralı For döngüsünün 1., 2., 3., 4. ve 5. iterasyonları.

   1.1. $B = T_1^{ON} \oplus T_1^{OFF} = 0001101 \oplus 0000100 = 0001001$

   B = φ4 : B = φ4 : (00 01 0 01) = 0101

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {0101, {1001, ~~1110~~, 0011}}={0101, 1001, 0011}

   1.2. $B = T_1^{ON} \oplus T_2^{OFF} = 0001101 \oplus 0101110 = 0100011$

   B = φ4 : B = φ4 : (01 00 0 11) = 1001

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1001~~, {0101, 1001, 0011}}={0101, 1001, 0011}

   1.3. $B = T_1^{ON} \oplus T_3^{OFF} = 0001101 \oplus 1010101 = 1011000$

   B = φ4 : B = φ4: (10 11 0 00) = 1100

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {1100, {0101, 1001, 0011}} = {1100, 0101, 1001, 0011}

   1.4. $B = T_1^{ON} \oplus T_4^{OFF} = 0001101 \oplus 1010100 = 1011001$

   B = φ4 : B = φ4 : (10 11 0 0 1) = 1101

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {1100, 0101, 1001, 0011}}={1100, 0101, 1001, 0011}

   1.5. $B = T_1^{ON} \oplus T_5^{OFF} = 0001101 \oplus 0010010 = 0011111$

   B = φ4 : B = φ4 : (00 11 1 11) = 0111

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~0111~~, {1100, 0101, 1001, 0011}}={1100, 0101, 1001, 0011}

2. T_2^{ON} = 1000101 ∈ $S_{ON}(D_2)$'nin $S_{OFF}(D_2)$ kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 2. iterasyonu ve (3). numaralı For döngüsünün 1., 2., 3., 4. ve 5. iterasyonları.

   2.1. $B = T_2^{ON} \oplus T_1^{OFF} = 1000101 \oplus 0000100 = 1000001$

   B = φ4 : B = φ4 : (10 00 0 01) = 1001

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1001~~, {1100, 0101, 1001, 0011}}={1100, 0101, 1001, 0011}

2.2. $B = T_2^{ON} \oplus T_2^{OFF} = 1000101 \oplus 0101110 = 1101011$

B = φ4 : B = φ4 : (11 01 0 11) = 1001

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {1100, 0101, 1001, 0011}}={1100, 0101, 1001, 0011}

2.3. $B = T_2^{ON} \oplus T_3^{OFF} = 1000101 \oplus 1010101 = 0010000$

B = φ4 : B = φ4: (00 10 0 00) = 0100

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {0100, {~~1100~~, ~~0101~~, 1001, 0011}} = {0100, 1001, 0011}

2.4. $B = T_2^{ON} \oplus T_4^{OFF} = 1000101 \oplus 1010100 = 0010001$

B = φ4 : B = φ4 : (00 10 0 01) = 0101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~0101~~, {10100, 1001, 0011}}={0100, 1001, 0011}

2.5. $B = T_2^{ON} \oplus T_5^{OFF} = 1000101 \oplus 1010111 = 1010111$

B = φ4 : B = φ4 : (10 10 1 11) = 1111

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1111~~, {0100, 1001, 0011}}={0100, 1001, 0011}

3. T_3^{ON} = 0101000 ∈ $S_{ON}(D_2)$ 'nin $S_{OFF}(D_2)$ kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 3. iterasyonu ve (3). numaralı For döngüsünün 1., 2., 3., 4. ve 5. iterasyonları.

3.1. $B = T_3^{ON} \oplus T_1^{OFF} = 0101000 \oplus 0000100 = 0101100$

B = φ4 : B = φ4 : (01 01 1 00) = 1110

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1110~~, {0100, 1001, 0011}}={0100, 1001, 0011}

3.2. $B = T_3^{ON} \oplus T_2^{OFF} = 0101000 \oplus 0101110 = 0000110$

B = φ4 : B = φ4 : (00 00 1 10) = 0011

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~0011~~, {0100, 1001, 0011}}={0100, 1001, 0011}

3.3. $B = T_3^{ON} \oplus T_3^{OFF} = 0101000 \oplus 1010101 = 1111101$

B = φ4 : B = φ4: (11 11 1 01) = 1111

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1111~~, {0100, 1001, 0011}} = {0100, 1001, 0011}

3.4. $B = T_3^{ON} \oplus T_4^{OFF} = 0101000 \oplus 1010100 = 1111100$

B = φ4 : B = φ4 : (11 11 1 00) = 1110

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1110~~, {0100, 1001, 0011}}={0100, 1001, 0011}

3.5. $B = T_3^{ON} \oplus T_5^{OFF} = 0101000 \oplus 1010111 = 0111010$

B = φ4 : B = φ4 : (01 11 0 10) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}}={0100, 1001, 0011}

D_3 karar fonksiyonun işlenmesi: (1). For döngüsünün 3. iterasyonu.

Giriş verileri:

$S_{ON}(D_3) = \{0000100, 1010101, 1000101, 1010100, \ 0010010\}$

$S_{OFF}(D_3) = \{0001101, 0101110, 0101000\}$

$BT_FM = \{0100, 1001, 0011\}$

1. T_1^{ON} = 0101000 $\in$ $S_{ON}(D_3)$ 'nin $S_{OFF}(D_3)$ kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 1. iterasyonu ve (3). numaralı For döngüsünün 1., 2. ve 3 iterasyonları.

1.1. $B = T_1^{ON} \oplus T_1^{OFF} = 0101000 \oplus 0001101 = 0001001$

B = φ4 : B = φ4 : (00 01 0 01) = 0101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~0101~~, {0100, 1001, 0011}}={0100, 1001, 0011}

1.2. $B = T_1^{ON} \oplus T_2^{OFF} = 0101000 \oplus 0101110 = 0101010$

B = φ4 : B = φ4 : (01 01 0 10) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}}={0101, 1001, 0011}

1.3. $B = T_1^{ON} \oplus T_3^{OFF} = 0101000 \oplus 0101000 = 0101100$

B = φ4 : B = φ4: (01 01 1 00) = 1110

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1110~~, {0100, 1001, 0011}} = {0100, 1001, 0011}

2. T_2^{ON} = 1010101 ∈ S_{ON}(D3)'nin S_{OFF}(D3) kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 2. iterasyonu ve (3). numaları For döngüsünün 1., 2. ve 3. iterasyonları.

2.1. $B = T_2^{ON} \oplus T_1^{OFF} = 1010101 \oplus 0001101 = 1011000$

B = φ4 : B = φ4 : (10 11 0 00) = 1100

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1100~~, {0100, 1001, 0011}}={0100, 1001, 0011}

2.2. $B = T_2^{ON} \oplus T_2^{OFF} = 1010101 \oplus 0101110 = 1111011$

B = φ4 : B = φ4 : (11 11 0 11) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}}={0100, 1001, 0011}

2.3. $B = T_2^{ON} \oplus T_3^{OFF} = 1010101 \oplus 0101000 = 1111101$

B = φ4 : B = φ4: (11 11 1 01) = 1111

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1111~~, {0100, 1001, 0011}} = {0100, 1001, 0011}

3. T_3^{ON} = 1000101 ∈ S_{ON}(D_3)'nin S_{OFF}(D_3) kümesinin elemanları ile işlenme sokulması. (2). numaları For döngüsünün 3. iterasyonu ve (3). numaralı For döngüsünün 1., 2. ve 3. iterasyonları.

3.1. $B = T_3^{ON} \oplus T_1^{OFF} = 1000101 \oplus 0001101 = 1001000$

B = φ4 : B = φ4 : (10 01 0 00) = 1100

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1100~~, {0100, 1001, 0011}}={0100, 1001, 0011}

3.2. $B = T_3^{ON} \oplus T_2^{OFF} = 1000101 \oplus 0101110 = 1101011$

B = φ4 : B = φ4 : (11 01 0 11) = 1101

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}}={0100, 1001, 0011}

3.3. $B = T_3^{ON} \oplus T_3^{OFF} = 1000101 \oplus 0101000 = 1101101$

B = φ4 : B = φ4: (11 01 1 01) = 1111

BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1111~~, {0100, 1001, 0011}} = {0100, 1001, 0011}

4. T_4^{ON} = 1010100 ∈ $S_{ON}(D_3)$'nin $S_{OFF}(D_3)$ kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 4. iterasyonu ve (3). numaralı For döngüsünün 1., 2. ve 3. iterasyonları.

   4.1. $B = T_4^{ON} \oplus T_1^{OFF} = 1010100 \oplus 0001101 = 1011001$

   B = φ4 : B = φ4 : (10 11 0 01) = 1101

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}}={0100, 1001, 0011}

   4.2. $B = T_4^{ON} \oplus T_2^{OFF} = 1010100 \oplus 0101110 = 1111010$

   B = φ4 : B = φ4 : (11 11 0 10) = 1101

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}}={0100, 1001, 0011}

   4.3. $B = T_4^{ON} \oplus T_3^{OFF} = 1010100 \oplus 0101000 = 1111100$

   B = φ4 : B = φ4: (11 11 1 00) = 1110

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1110~~, {0100, 1001, 0011}} = {0100, 1001, 0011}

5. T_5^{ON} = 1010100 ∈ $S_{ON}(D_3)$'nin $S_{OFF}(D_3)$ kümesinin elemanları ile işlenme sokulması. (2). numaralı For döngüsünün 5. iterasyonu ve (3). numaları For döngüsünün 1., 2. ve 3. iterasyonları.

   5.1. $B = T_5^{ON} \oplus T_1^{OFF} = 1010100 \oplus 0001101 = 0011111$

   B = φ4 : B = φ4 : (00 11 1 11) = 0111

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~0111~~, {0100, 1001, 0011}}={0100, 1001, 0011}

   5.2. $B = T_5^{ON} \oplus T_2^{OFF} = 1010100 \oplus 0101110 = 0111100$

   B = φ4 : B = φ4 : (01 11 1 00) = 1110

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1110~~, {0100, 1001, 0011}}={0100, 1001, 0011}

   5.3. $B = T_5^{ON} \oplus T_3^{OFF} = 1010100 \oplus 0101000 = 0111010$

   B = φ4 : B = φ4: (01 11 0 10) = 1101

   BT_FM(D) = Reduce_BT_FM(D) (BT_FM(D), B) = {~~1101~~, {0100, 1001, 0011}} = {0100, 1001, 0011}

$S_{ON}(D_1)$, $S_{ON}(D_2)$ ve $S_{ON}(D_3)$ kümelerindeki bütün elemanlar, uygun olarak $S_{OFF}(D_1)$, $S_{OFF}(D_2)$ ve $S_{OFF}(D_3)$ kümesindeki elemanlar ile işleme sokulması sonucunda Çizelge 3.5'de gösterilen veri kümesinin indirgenmiş bit tabanlı FM'si elde edilir. Bundan sonra Şekiller 3.5 ve 3.6'da gösterilen algoritmalar kullanılarak CNF'deki bit tabanlı FM DNF'ye çevrilerek Çizelge 3.5'de gösterilen veri kümesinin ÖMAK'ları elde edilir.

```
Çevir_BT_FF(D) (BT_FF_min(D))
{
        F0={0}^N;  |F0|=1
        For q=1 to Q
        {
                Vq1 = ∅; Vq2 = {0}^N
                For i=1 to |Fq-1|
                {
                        Select ai∈Fq-1; g=ai&Bq
                        If g≠{0}^N
                        Then
                                Vq1 = Vq1∪ ai
                        Else
                                Vq2 = Vq2∪ ai
                }
        Generate_Products (Bq, Vq1, Vq2)
        }
Return (Fq = V1∪V3)
}
```

Şekil 3.5. Bit tabanlı fark fonksiyonu disjunktif normal forma çevirerek veri kümesinin özelliklerin minimal alt kümelerini bulan algoritma

Şekil 3.5'te gösterilen *Çevir_BT_FF(D)* algoritması örnek 3.6'nın formalize edilmiş halidir. Bu algoritmada lüzumsuz terimlerin oluşmasını engelleyerek veri kümesinin ÖMAK'larını oluşturan *Generate_Products* (B_q, V_{q1}, V_{q2}) alt algoritması da aşağıda Şekil 3.6'da gösterilmektedir.

```
Generate_Products (Bq, Vq1, Vq2)
{
    E(Bq)={Prj(Bq): dj =1, j=1,2, …, N}
    Vq3 =Ø
    For j=1 to |Vq2|
    {
        Select bj∈ Vq2
        For i =1 to | E(Bq)|
        {
            Select ci ∈ E(Bq); d= bj | ci
            For k=1 to |Vq1|
            {
                Select ek∈Vq1; f =ek&d
                If f ≠ek Then
                Vq3 =Vq3 ∪ d
            }
        }
    }
Return (Vq3)
}
```

Şekil 3.6. Gereksiz terimlerin oluşmasını engelleyerek özelliklerin minimal alt kümelerini oluşturan alt algoritma

BT_FM(D)={0100, 1001, 0011} için Şekiller 3.5. ve 3.6'da gösterilen algoritmalar uygulanarak Çizelge 3.5'de gösterilen veri kümesinin ÖMAK'ları aşağıdaki gibi elde edilir.

İterasyon 1: q=1; B_1=0100; F_0={{0}^{4}}={0000}; V_{q1}=∅; V_{q2}={{0}^{N}}

1.1. F_0 kümesi içindeki bütün $a_i \in F_0$'lar B_1 ile lojik *kesişme* işlemine sokulur: a_1 & B_1 = 0000 & 0100 = {0}^{6}

1.2. F_0 kümesi V_{11} ve V_{12} kümelerine ayrılır:

V_{11}={$a_1 \in F_0$: a_1& $B_1 \neq$ {0}^{n} }

$V_{12}=F_0-V_{11}$

$V_{11}=\varnothing$

$V_{12}=F_0-V_{11}=\{a_1\}=\{0000\}$

1.3. $T_1 = V_{12} \mid E(B_1)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{11} kümesinin elemanlarıyla karşılaştırılır. V_{11} kümesinin elemanları tarafından yutulan elemanlar T_1 kümesinden silinir. $E(B_1)=\{0100\}$

$T_1 = V_{12} \mid E(B_1) = \{0000\} \mid \{0100\} = \{0100\}$

1.4. $F_1 = V_{11} \cup T_1$ kümesi hesaplanır.

$F_1 = \varnothing \cup \{0100\}$

İterasyon 2: $q=2$; $B_2=1001$; $F_1=\{0100\}$; $V_{21}=\varnothing$; $V_{22}=\{\{0\}^N\}$

2.1. F_1 kümesi içindeki bütün $a_i \in F_1$'lar B_2 ile lojik *kesişme* işlemine sokulur:

$a_1 \& B_2 = 0100 \,\&\, 1001 = \{0\}^6$

2.2. F_1 kümesi denklemler V_{11} ve V_{12} kümelerine ayrılır:

$V_{21}=\{a_i \in F_1 : a_i \& B_2 \neq \{0\}^n\}$

$V_{22}= F_1 - V_{21}$

$V_{21}=\varnothing$

$V_{22}=F_1-V_{21} = \{0100\}$

2.3. $T_2 = V_{22} \mid E(B_2)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{21} kümesinin elemanlarıyla karşılaştırılır. V_{21} kümesinin elemanları tarafından yutulan elemanlar T_2 kümesinden silinir.

$E(B_2) = \{1000, 0001\}$

$T_2 = V_{22} \mid E(B_2) = \{0100\} \mid \{1000, 0001\} = \{1100, 0101\}.$

$F_2 = V_{21} \cup T_2$ kümesi hesaplanır.

$F_2 = \varnothing \cup \{1100, 0101\} = \{1100, 0101\}$

İterasyon 3: $q=3$, $B_3=0011$; $F_2=\{1100, 0101\}$; $V_{31}=\varnothing$; $V_{32}=\{\{0\}^N\}$

3.1. F_2 kümesi içindeki bütün $a_i \in F_2$'lar B_3 ile lojik *kesişme* işlemine sokulur:

$a_1 \& B_3 = 1100 \& 0011 = \{0\}^6$

$a_2 \& B_3 = 0101 \& 0011 \neq \{0\}^6$

3.2. F_2 kümesi V_{11} ve V_{12} kümelerine ayrılır:

$V_{31} = \{a_i \in F_2 : a_i \& B_3 \neq \{0\}^n$

$V_{32} = F_2 - V_{31}$

$V_{31} = \{a_2\} = \{0101\}$

$V_{32} = F_2 - V_{31} = \{a_1\} = \{1100\}$

3.3. $T_3 = V_{32} \mid E(B_3)$ kullanılarak elemanlar birer birer oluşturulur. Her yeni oluşturulan eleman V_{31} kümesinin elemanlarıyla karşılaştırılır. V_{31} kümesinin elemanları tarafından yutulan elemanlar T_3 kümesinden silinir.

$E(B_3) = \{0010, 0001\}$

$T_3 = V_{32} \mid E(B_3) = \{1100\} \mid \{0010, 0001\} = \{1110, \sout{1101}\}$.

3.4. $F_3 = V_{31} \cup T_3$ kümesi hesaplanır.

$F_3 = V_{31} \cup T_3 = \{0101\} \cup \{0101\} = \{0101, 0101\}$

Elde edilen bit tabanlı ÖMAK'lar {0101, 0101} Struct_BTİ_CIZELGE_3.5 veri yapısı kullanılarak ÖLT' ye çevrilir. Sonuç olarak Çizelge 3.5'te verilen veri kümesinin ÖMAK'ları $\{\{A_1, A_2, A_3\}, \{A_2, A_4\}\}$ şeklinde elde edilir.

3.8. Boole fonksiyon yaklaşımlı özellik seçme yöntemi ile yapılan deney sonuçları

Boole fonksiyon yaklaşımlı özellik seçme yöntemi ile bir veri kümesinin doğruluk tablo görüntüsünden indirgenmiş FF'nun lojik operatörler kullanılarak elde edilebileceği ve elde edilen bit tabanlı ifadelerden oluşan indirgenmiş FF'nin DNF'ye çevrilerek veri kümesine ait ÖMAK'ların tamamının elde edilebileceği yukarıda anlatılmıştır. Boole fonksiyon yaklaşımı kullanılarak geliştirilen bu yöntemin daha hızlı ve daha az hafızaya ihtiyaç duyarak çalıştığını göstermke için UCI makine öğrenmesi veri ambarından alınan 19 veri kümesi kullanılarak deneyler yapılmıştır. Deneylerde kullanılan bu veri kümelerinin şart özellikleri sayısı, şart özelliklerin farklı değerleri, şart özelliklerinin değerlerinin tipleri (sayısal, alfa-numerik ve karışık), obje sayıları ve karar özelliğinin farklı değerleri (sınıf sayısı) gibi değişik karakteristiktik özellikleri vardır. Kullanılan veri kümeleri aşağıda Çizelge 3.7'de gösterilmektedir.

Çizelge 3.7. Boole fonksiyon yaklaşımlı özellik seçme yönteminin deneylerinde kullanılan veri kümelerinin özellikleri

Veri kümesi	Şart özelliklerinin sayısı	Obje sayısı	Sınıf sayısı
Iris	4	150	3
Monk1	6	124	2
Monk2	7	169	2
Monk3	6	122	2
Diabetes	8	168	2
Shuttle	9	43500	5
Heart	13	270	2
Australian	14	690	2
Zoo	17	101	7
Lymn	18	148	4
Statlog	24	1000	2
Chess	36	3196	2
Vote	16	435	2
Mushroom	22	8124	2
Sonar	60	208	2
Ionosphere	33	351	2
Annealing	38	798	6
Spectf Heart	44	187	56
Dna	57	106	2

Bu yöntemin performansının değerlendirilebilmesi için elde edilen sonuçlar özellik seçme amacı ile en yoğun olarak kullanılmakta olan RSES (Rough Set Exploration System) ve RSAR (Rough Set-based Attribute Reduction) programlarının sonuçlarıyla karşılaştırılmıştır. RSES programı ile alınan sonuçların RSAR programına göre daha az hafıza ve zaman kullandığı ve ayrıca ÖMAK'ların tamamını elde ettiğinden dolayı, geliştirilen yöntem sadece RSES programının sonuçlarıyla karşılaştırılmıştır. RSES programı bir veri kümesinin ÖMAK'larını bulma, bu ÖMAK'ları kullanarak karar kuralları çıkartma, sürekli sayısal özellikler ayrıklaştırılması, veriler içerisinde desen arama ve veri işleme gibi birçok özelliği vardır (Bazan ve ark., 1999; Bazan ve ark., 2002; Bazan ve ark., 2000). Biz deneylerde RSES programın fark fonksiyonunu kullanarak bir veri kümesinin bütün ÖMAK'larını elde etme özelliğini kullandık. Yapılan deneylerde kullanılan bilgisayar Intel Core2Duo@2.40 GHz işlemcili, 2GB RAM'lı olup Microsoft Vista Home sürümü İşletim Sistemi ile desteklenmektedir.

Deneylerde kullanılan ve Çizelge 3.7'de gösterilen veri kümeleri hiçbir ön işleme tabi tutulmadan özellik seçme işlemine alınmaktadır. Aşağıdaki çizelgelerde geliştirilen yöntem ve RSES kullandıkları hafıza miktarı, işlemci süreleri ve elde ettikleri ÖMAK sayıları yönünden birbirleriyle karşılaştırılmaktadırlar. Programların

kullandığı hafıza miktarları ve işlemci süreleri *Microsoft process explorer* programıyla ölçülmüştür.

Çizelge 3.8. Elde edilen özelliklerin minimal alt kümelerine göre Boole fonksiyon yaklaşımlı özellik seçme yöntemi ve RSES'in karşılaştırılması

Veri kümesi	Elde edilen ÖMAK sayısı		Elde edilen reduct sayısı		Elde edilen reductların büyüklüğü	
	RSES	Boole fonksiyon yaklaşımlı özellik seçme yöntemi	RSES	Boole fonksiyon yaklaşımlı özellik seçme yöntemi	RSES	Boole fonksiyon yaklaşımlı özellik seçme yöntemi
Iris	4	4	4	4	3	3
Monk1	1	1	1	1	3	3
Monk2	1	1	1	1	6	6
Monk3	1	1	1	1	4	4
Diabetes	28	28	19	19	3	3
Shuttle	19	19	19	19	4	4
Heart	109	109	18	18	3	3
Australian	44	44	5	5	3	3
Zoo	34	34	1	1	1	1
Lymn	424	424	1	1	6	6
Statlog	2424	2424	1	1	5	5
Chess	4	4	4	4	29	29
Vote	3	3	1	1	9	9
Mushroom	292	292	13	13	4	4
Sonar	**Başarısız**	**31844**	**Başarısız**	**168**	**Başarısız**	**2**
Ionosphere	**Başarısız**	**4257**	**Başarısız**	**6**	**Başarısız**	**2**
Annealing	**Başarısız**	**275**	**Başarısız**	**6**	**Başarısız**	**6**
Spectf Heart	**Başarısız**	**26454**	**Başarısız**	**1**	**Başarısız**	**2**
Dna	**Başarısız**	**6259767**	**Başarısız**	**1**	**Başarısız**	**3**

Boole fonksiyon yaklaşımlı özellik seçme yöntemi ve RSES ile ayrı ayrı özellik seçme işlemine tabi tutulan 19 tane veri kümesinin 14 tanesinde aynı ÖMAK'lar elde edilmiştir. Kalan 5 veri kümesinde ise RSES hafıza yetersizliğinden dolayı başarısız olmasına rağmen Boole fonksiyon yaklaşımlı özellik seçme yöntemi bu veri kümelerinin ÖMAK'ları bulmuştur. Genellikle 24 ve daha fazla özellik içeren veri kümelerinde geliştirilen yöntem RSES'e göre daha iyi sonuçlar elde etmiştir.

Boole fonksiyon yaklaşımlı özellik seçme yöntemi ile RSES'in ihtiyaç duyduğu hafıza miktarı ve kullanılan işlemci zamanları sırasıyla aşağıdaki Çizelgeler 3.9 ve 3.10'da gösterilmektedir. Çizelge 3.9'da dikkati daha büyük bellek miktarlarına yönlendirmek için 1MB'den küçük olan hafıza miktarları 1MB'a eşitlenmiştir. Genelde, küçük veri kümeleri için RSES ve bu yöntem benzer miktarlarda hafıza kullandıkları görülmüştür. Fakat büyük veri kümeleri Sonar, Ionesphere, Annealing, Spectf Heart ve

DNA için RSES 1800MB hafıza miktarına ulaşmış ve bu durumda hafıza yetersizliğinden dolayı başarısız olmuştur. Buna rağmen, bu yöntem en fazla 895 MB hafıza miktarı kullanılarak bütün veri kümelerini başarı ile işlemiştir.

Çizelge 3.9. Kullanılan hafıza miktarına göre Boole fonksiyon yaklaşımlı özellik seçme yöntemi ve RSES'in karşılaştırılması

Veri kümesi	**Kullanılan Hafıza (MB)**		**Oran W_1 / W_2**
	RSES (W_1)	**Boole fonksiyon yaklaşımlı özellik seçme yöntemi (W_2)**	
Iris	1	1	1
Monk1	1	1	1
Monk2	1	1	1
Monk3	1	1	1
Diabetes	1	1	1
Shuttle	8,3	1	8,3
Heart	1	1	1
Australian	1	1	1/
Zoo	1	1	1
Lymn	1	1	1
Statlog	1	1	1
Chess	664	1	664
Vote	1	1	1
Mushroom	2,3	1	2,3
Sonar	**>1800**	**5,7**	**315**
Ionosphere	**>1800**	**1**	**1800**
Annealing	**>1800**	**1**	**1800**
Spectf Heart	**>1800**	**2,1**	**857**
Dna	**>1800**	**895**	**2,01**

Çizelge 3.10. Kullanılan işlemci zamanına göre geliştirilen Boole fonksiyon yaklaşımlı özellik seçme yöntemi ve RSES'in karşılaştırılması

Veri kümesi	**Kullanılan işlemci zamanı (saniye)**		**Oran T_1 / T_2**
	RSES (T_1)	**Boole fonksiyon yaklaşımlı özellik seçme yöntemi (T_1)**	
Iris	0,219	0,047	4,65
Monk1	0,172	0,031	5,55
Monk2	0,156	0,031	5,00
Monk3	0,093	0,016	5,80
Diabetes	0,717	0,749	0,96
Shuttle	1793,122	3470,5	0,52
Heart	0,436	0,140	3,11
Australian	0,886	0,858	1,03
Zoo	0,140	0,047	2,98
Lymn	20,577	0,078	263
Statlog	266,527	2,277	117
Chess	63,554	29,234	2,17
Vote	0,202	0,280	0,72
Mushroom	144,488	143,942	1,00
Sonar	**>216000**	**88,905**	**2429**
Ionosphere	**>14400**	**1,544**	**9326**
Annealing	**>1400**	**2,839**	**493**
Spectf Heart	**>324000**	**28,595**	**11330**
Dna	**>4920**	**20160**	**0,244**

Çizelge 3.10'da RSES ve Boole fonksiyon yaklaşımlı özellik seçme yöntemini tarafından kullanılan işlemci zamanları gösterilmiştir. Bu zamanlara bakıldığıda 3 veri kümesinde RSES'in Boole fonksiyon yaklaşımlı özellik seçme yöntemine göre daha kısa sürede ÖMAK'ları bulduğu görülmektedir. Kalan 16 veri kümesi için ise Boole fonksiyon yaklaşımlı özellik seçme yöntemi RSES'e göre daha iyidir. RSES tarafından çözülemeyen 5 veri kümesi için ise Boole fonksiyon yaklaşımlı özellik seçme yönteminin RSES'in hafıza yetersizliği hatasını verene kadar kullandığı işlemci zamanından çok daha az zamanda ÖMAK'ları bulduğu görülür.

Sonuç olarak Boole fonksiyon yaklaşımlı özellik seçme yöntemi yöntemin orta ve büyük veri kümeleri için RSES'e göre çok daha az hafıza miktarı ve işlemci zamanı kullanarak ÖMAK'ları bulduğu söylenebilir.

4. İNDİRGENMİŞ FARK FONKSİYONUNU DİSJUNKTİF NORMAL FORMA ÇEVİRME KARMAŞIKLIĞININ DEĞERLENDİRİLMESİ

4.1. İndirgenmiş fark fonksiyonunun disjunktif normal forma çevrilmesi sürecinde karşılaşılan en kötü hafıza karmaşıklığı

Yukarıda, alt bölüm 3.4'de, bir veri kümesinin U_i ve U_k objeleri arasındaki farklı özelliklerin belirlenmesiyle ÖLT formundaki H_{ik} ifadesinin oluşturulması denklem 3.8'le verilmiştir. Bu denkleme göre elde edilen herhangi bir H_{ik} ifadesinin içerisindeki özelliklerin sayısı o ifadenin ağırlığı olarak tanımlanır ve $W(H_{ik})$ olarak gösterilir. Örnek olarak, $H_{12} = (A_5 \vee A_6 \vee A_8)$ ifadesinin ağırlığı $W(H_{12}) = 3$'dür. Bir veri kümesinden elde edilen bütün ÖLT formatındaki terimlerin lojik birleşme işlemine tabi tutulmasıyla da o veri kümesinin fark matrisi elde edilir (Swiniarski ve Skowron, 2003; Komorowski ve ark., 1999; Jensen ve Shen, 2007; Skowron ve Rauszer,1992). Bu tanıma göre M tane özellikten oluşan bir veri kümesinin FM'si aşağıdaki gibi elde edilir.

$$\mathrm{DF} = \bigwedge_{i=1}^{M-1} \bigwedge_{k=i+1}^{M} \mathrm{H_{ik}} \tag{4.1}$$

Bir FM'yi oluşturan ifadelerin sayısına o FF'nin büyüklüğü denir. Bir Y FF'sinin büyüklüğü $Q(Y)$ olarak tanımlanır. Mesela, $D = (e \vee r)(d \vee r)(d \vee e \vee f)$ FF'sinin büyüklüğü $Q(D) = 3$ değerindedir. Bir A_j özelliğinin bir fonksiyon içerisinde bulunma sayısına o özelliğin frekansı denir ve $\varphi(A_j)$ şeklinde tanımlanır (Wang ve Wang, 2001).

Yukarıda bölüm 3.5'de de anlatıldığı gibi, M adet objeden oluşan bir veri kümesinden elde edilecek bir FF'in büyüklüğü $G = 0.5(M^2 - M)$ kadardır. Başka bir deyişle, M adet nesneden oluşan bir veri kümesinden elde edilen FF G tane bileşenden oluşur. Fakat yukarıda da anlatıldığı gibi bu elemanlardan bir kısmı gereksizdir. FF'den bu gereksiz elemanlar silindikten sonra elde edilen indirgenmiş $\mathrm{FF_{min}}$'in eleman sayısı $Q \le G$ olacaktır. Bu yüzden elde edilen indirgenmiş $\mathrm{FF_{min}}$ için denklem 3.8 ve 4.1 $q \in \{1,2,\ldots,Q\}$ için aşağıdaki gibi yeniden yazılabilir.

$$H_q = h_{q1} \vee h_{q2} \vee \ldots \vee h_{qj} \ldots \vee h_{qN} \tag{4.2}$$

$$FF_{min} = \bigwedge_{q=1}^{Q} H_q \tag{4.3}$$

Denklem 4.3'te gösterilen FF_{min}'in DNF'ye çevrilebilmesi için denklemin sağ tarafındaki H_q ifadelerinin, alt bölüm 3.6.1'de anlatıldığı gibi, genişletilmesi gerekir. Bu işlem sonucunda elde edilen FF_{min}'i oluşturan değişik implicantların sayısı genişletme işleminin hafıza karmaşıklığı S'yi belirler. Bu genişletme işlemi sırasında yapılan işlemlerin sayısı ise bu işlemin zaman karmaşıklığı T'yi belirler. Johnsonbaugh ve Schaefer (2004)'e göre bir algoritma tarafından kullanılan en fazla hafıza miktarı ve en fazla işlem sayısı sırasıyla *en kötü hafıza karmaşıklığı* (EKHK) ve *en kötü zaman karmaşıklığı* (EKZK) olarak adlandırılır. Denklem 4.3'ün EKHK ve EKZK'sını belirlemek için Çizelge 3.1'de gösterilen örnek veri kümesini kullanalım.

Denklemler 3.8 ve 4.1'e göre elde edilen FF aşağıdaki gibidir:

DF = $(A_2 \vee A_3 \vee A_5 \vee A_6 \vee A_9) \wedge (A_1 \vee A_2 \vee A_3 \vee A_4 \vee A_5 \vee A_6 \vee A_7) \wedge (A_1 \vee A_2 \vee A_3 \vee A_6 \vee A_7 \vee A_8) \wedge$
$(A_1 \vee A_2 \vee A_3 \vee A_4 \vee A_6 \vee A_7 \vee A_9) \wedge (A_1 \vee A_2 \vee A_4 \vee A_7 \vee A_8 \vee A_9) \wedge (A_1 \vee A_2 \vee A_4 \vee A_5 \vee A_6 \vee A_7 \vee A_9) \wedge$
$(A_1 \vee A_2 \vee A_4 \vee A_5 \vee A_6 \vee A_7 \vee A_8 \vee A_9) \wedge (\mathbf{A_1} \vee \mathbf{A_2} \vee \mathbf{A_4} \vee \mathbf{A_5} \vee \mathbf{A_6} \vee \mathbf{A_7}) \wedge (A_1 \vee A_2 \vee A_3 \vee A_4 \vee A_5 \vee A_6 \vee$
$A_7 \vee A_8) \wedge (\mathbf{A_5} \vee \mathbf{A_6} \vee \mathbf{A_8}) \wedge (\mathbf{A_5} \vee \mathbf{A_6} \vee \mathbf{A_9}) \wedge (A_3 \vee A_5 \vee A_6 \vee A_8 \vee A_9) \wedge (\mathbf{A_8} \vee \mathbf{A_9}) \wedge (\mathbf{A_3} \vee \mathbf{A_6} \vee \mathbf{A_9}) \wedge (\mathbf{A_3}$
$\vee \mathbf{A_6} \vee \mathbf{A_8})$

Bu FF'de koyu renk ile gösterilen bileşenler gerekli olanlardır. Lüzumsuz olan bileşenlerin FF'den silinmesiyle $Q = 6$ tane bileşenden oluşan bir FF_{min} elde edilir. Denklemler 4.2 ve 4.3'e göre elde edilen FM_{min} aşağıdaki gibidir:

$$FM_{min} = \bigwedge_{q=1}^{6} H_q = (A_1 \vee A_2 \vee A_4 \vee A_5 \vee A_6 \vee A_7) \wedge (A_5 \vee A_6 \vee A_8) \wedge (A_5 \vee A_6 \vee A_9) \wedge (A_8 \vee A_9) \wedge (A_3 \vee A_6 \vee A_9) \wedge (A_3 \vee A_6 \vee A_8) \tag{4.4}$$

Denklem 4.4'deki her bir lojik çarpım (AND) işlemi sonrasında yeni bir implikantın oluşacağı açıktır. Bu yüzden 4.4 ifadesindeki bütün çarpım işlemleri sonucunda oluşabilecek implikantların sayısının mümkün olan en yüksek değeri aşağıda gösterilen denklemden elde edilebilir.

$$S = \prod_{q=1}^{Q} W(H_q) \tag{4.5}$$

Denklem 4.5'teki $W(H_q)$ için ortalama ağırlık değeri aşağıdaki gibi ifade edilir.

$$n = \sum_{i=1}^{Q} W(H_i)/Q \tag{4.6}$$

Denklem 4.6'dan elde edilen n değeri kullanılarak denklem 4.5 ile ifade edilen hafıza karmaşıklığı aşağıdaki gibi bulunur.

$$S = O(n^Q) \tag{4.7}$$

Denklem 4.4'deki çarpım işlemleri sonrasında elde edilecek olan implikantların bir kısmı özellik seçme işlemi için lüzumsuz olabilir. Fakat bazı durumlarda bu gereksiz implikantların sayısı oldukça küçük olabildiğinden FM_{min}'den DNF'ye çevirme işleminin EKHK $S = O(n^Q)$ olarak kabul edilecektir. Yani bu dönüştürme işlemi Q'ye bağlı olan üstel bir hafıza karmaşıklığına sahiptir. Mesela, denklem 4.4'deki FM_{min}'in EKHK'sı $S = n^Q = 3.3333^6 = 1370$ değerindedir. FF tabanlı özellik seçme tekniklerinde Boole manipülasyon teknikleri kullanarak hafıza karmaşıklığı değerleri azaltılmasına rağmen (Starzyk ve ark., 2000), FM_{min}'den DNF'ye çevirme işlemi sırasında oluşan implikant sayısı çok büyük değerlere ulaşmakta ve hafıza yetersizliğinden dolayı özellik seçme işlemi hata verip sonlanmaktadır (Jensen ve Shen, 2007; Wang ve ark., 2007).

4.2. İndirgenmiş fark fonksiyonunun disjunktif normal forma çevrilmesi esnasında oluşan en kötü zaman karmaşıklığı

Denklem 4.3'te gösterilen FM_{min}'in DNF'ye çevrilmesi işleminin yarattığı hafıza karmaşıklığının yanında, bu çevirme işleminin bitmesi belli bir zamanda alacaktır. Çünkü bu çevirme işleminin bitmesi için birçok lojik çarpma işlemi gerçekleştirilecektir. Bu çevrim işleminde oluşabilecek maksimum çarpma işlemi denklem 4.8'de gösterildiği gibi elde edilir.

$$T_1 = S + (\sum_q W(H_q)) \tag{4.8}$$

Çok küçük veri kümelerinin işlenmesinde denklem 4.8'in sağ tarafındaki değer denklemin diğer tarafındaki değere göre çok küçüktür. Bu yüzden denklemin sağ tarafı göz ardı edilebilir. Böylece yeni elde edilen denklem $T_1 = S$ şeklinde olacaktır. FF_{min} 'in DNF'ye çevrilmesi işlemde oluşan gereksiz implikantların tespit edilebilmesi için DNF içerisindeki implikantlar ikişerli olarak karşılaştırması gerekir. Tüm gereksiz implikantları tespit edebilmek için toplamda $T_2 = 0.5(S^2 - S)$ kadar karşılaştırma yapılmalıdır. Bütün bu işlemler sonunda FF_{min} 'in DNF'ye çevrilme işleminin EKZK aşağıdaki gibi elde edilir.

$$T = T_1 + T_2 = 0.5 \text{ x } (S^2 + S) \tag{4.9}$$

Denklem 4.9'daki S yerine denklem 4.6'dan $S = O(n^Q)$ ifadesini koyarak aşağıdaki denklemi elde ederiz.

$$T = 0.5(n^{2Q} + n^Q) \rightarrow O(n^{2Q}) \tag{4.10}$$

Denklem 4.6 ile denklem 4.10'un karşılaştırılmasından aşağıdaki denklem elde edilir.

$$T = O(n^{2Q}) = O(n^Q)^2 = S^2 \tag{4.11}$$

Denklem 4.11'e bakarak FF_{min} 'in DNF'ye çevrilme işleminin EKZK değeri EKHK değerinin karesidir diyebiliriz (Şirzat ve ark., 2011).

Denklemler 4.6 ve 4.11'den S 'nin Q 'ye bağlı olarak üstel olarak büyüdüğü ve T 'ninde S 'nin karesine göre büyüdüğü görülür. Bu yüzden N ve M değerleri onlar ve yüzlere ulaştığı zaman sırasıyla S ve T değerleri çok büyük değerlere ulaşacak ve FF tabanlı özellik seçme işlemleri çok uzun zaman alan hesaplamalar ve yetersiz hafıza miktarları yüzünden imkânsız hale gelecektir. Bölüm 5'te, FF tabanlı özellik seçme

tekniklerinde oluşan bu hesaplama karmaşıklığını azaltmak için FF_{min} 'ni iteratif olarak bölüp işleyerek DNF'ye çeviren yöntem anlatılacaktır.

5. İNDİRGENMİŞ FARK FONKSİYONUNUN İTERATİF OLARAK BÖLÜNMESİYLE ÖZELLİKLERİN MİNİMAL ALT KÜMELERİNİN BULUNMASI

5.1. İndirgenmiş fark fonksiyonun böl ve yönet (divide and conquer) stratejisi ile küçük parçalarla çözülecek hale getirilmesi

Yukarıdaki bölümlerde de belirtildiği gibi, bir veri kümesine ait özelliklerin minimal alt kümelerinin tümünü elde etmek için veri kümesinin FF_{min}'nini DNF'ye çevirmek gerekir. Ne yazık ki bu işlem bir NP-hard problemdir. Bu yüzden genellikle orta ve büyük ölçekli veri tabanları için FF_{min}'in DNF'ye çevrilmesi imkânsızdır (Liu ve Yu, 2005; Wang ve ark., 2007; Jensen ve Shen, 2007). Aşağıda FF_{min}'den iteratif olarak elde edilen $F_1, F_2, \ldots F_{R \leq N-1}$ fonksiyon dizisini kullanarak FF_{min}'i DNF'ye çeviren bir yaklaşım anlatılacaktır. Bu yaklaşım ile birlikte FF_{min}'in işlenmesi sırasında oluşan zaman ve hafıza karmaşıklığı azalacak ve ÖMAK'ların hepsi bulunabilecektir.

Bu yaklaşımı açıklayabilmek için denklem 4.4'deki FF_{min}'in DNF'ye çevrilme işlemini gerçekleştirmemiz gerektiğini varsayalım ve bilgisayarımızın hafıza yetersizliğinden dolayı tüm FF_{min}'ini işleyemeyeceğini düşünelim. Bu probleme çözüm bulabilmek için aşağıdaki sorulara cevap verecek şekilde FF_{min}'i parçalayarak DNF'ye çevirme işlemini gerçekleştirmeliyiz. Bu sorular aşağıdakilerdir:

1. FF_{min}'i alt fonksiyonlara nasıl bölebiliriz?
2. Bölünen her alt fonksiyonu birbirinden ayrı şekilde nasıl işlemeliyiz ki orijinal FF_{min}'den hiçbir bilgi kaybımız olmasın?

Bu sorulara bir sistemin orijinal özelliklerini kaybetmeden sistemi daha küçük ve birbirinden bağımsız alt sistemlere bölebilen yer-değişmeli parçalama (*commutative decomposition*) kavramına dayanarak cevap verebiliriz (Lee ve ark.,2006). Bir FF_{min}'i bu tip bir parçalama işlemine sokmak için FF_{min} 'i ve FF_{min} içerisindeki $H_{q \in \{1,2,\ldots Q\}}$ ifadesini aşağıdaki gibi açıklayabiliriz.

$$D = \bigwedge_{q=1}^{Q} H_q \tag{5.1}$$

$$H_q = h_{q1} \vee h_{q2} \vee \ldots \vee h_{qW(H_q)} \tag{5.2}$$

Denklem 5.2'de $W(H_q)$ değeri H_q'nun büyüklüğüdür. H_q'nün denklem 5.2'deki ifadesi 5.1'de yerine konulursa aşağıda gösterilen denklem elde edilir.

$$D = H_1 \wedge H_2 \wedge \ldots \wedge H_{q-1} \wedge (h_{q1} \vee h_{q2} \vee \ldots \vee h_{qW(H_q)}) \wedge H_{q+1} \wedge H_{q+2} \wedge \ldots \wedge H_Q \tag{5.3}$$

Eğer $H_1 \wedge H_2 \wedge \ldots \wedge H_{q-1} \wedge H_{q+1} \ldots \wedge H_Q$ ifadesine F_1 denilirse denklem 5.3 aşağıdaki gibi yazılabilir:

$$D = (h_{q1} \vee h_{q2} \vee \ldots \vee h_{qW(H_q)}) \wedge F_1 = (h_{q1} \wedge F_1) \vee (h_{q2} \wedge F_1) \vee \ldots \vee (h_{qW(H_q)} \wedge F_1) \tag{5.4}$$

Şu ana kadar oluşan implikantların tekrar meydana çıkmasını engellemek için denklem 5.4'teki $h_{qi} \wedge F_1$ bileşenini hesaplanmadan önce $j < i$ şartını sağlayan bütün F_1'lerin silinmesi gerekir (Øhrn ve ark.,1999; Wang ve Wang, 2001; Zhao ve Wang, 2002). Bu işlem için denklem 5.4 aşağıdaki gibi yazılır.

$$D = (h_{q1} \wedge F_1) \vee (h_{q2} \wedge F_1 \# (h_{q1})) \vee \ldots \vee (h_{qW(H_q)} \wedge F_1 \# (h_{q1}, h_{q2}, \ldots, h_{qW(H_q)-1})) \tag{5.5}$$

Burada, $F_1 \# (h_{q1}, h_{q2}, \ldots)$ işlemi F_1 fonksiyonundan $(h_{q1}, h_{q2}, \ldots)$ bileşenlerinin çıkarılması manasına gelir. Denklem 5.5'te uygulanan bu teknik iyi bilinen *genişlet ve ele* (expand and eleminate) (Brayton ve ark., 1984, Starzyk ve ark., 2000) işleminin tersi olan *ele ve genişlet* işlemidir. Bu teknik sayesinde denklem 5.5'deki terimlerin sayısı DNF'ye çevrilme işleminden önce önemli bir şekilde azalacaktır. Fakat denklem 5.5'e göre D fonksiyonu işlenebilmesi için H_q bileşenine göre paralel olarak parçalanmalıdır. Maalesef böyle bir denklemin bu şekilde çalışabilmesi modern bilgisayarların çalışma mantığına da ters düşmektedir. Bu yüzden aşağıda gösterilen bileşenler kullanılarak denklem 5.5 iteratif olarak parçalanabilir.

$$F_1\#(h_{q1}) \Rightarrow F_2$$
$$F_1\#(h_{q1}, h_{q2}) = F_2\#(h_{q2}) \Rightarrow F_3$$
$$F_1\#(h_{q1}, h_{q2}, h_{q3}) = F_2\#(h_{q2}, h_{q3}) = F_3\#(h_{q3}) \Rightarrow F_4$$
$$\vdots$$
$$F_1\#(h_{q1}, h_{q2}, \ldots, h_{qW(H_q)-1}) = F_2\#(h_{q2}, h_{q3}, \ldots, h_{qW(H_q)-1}) = \ldots = F_{W(H_q)-1}\#(h_{qW(H_q)-1}) \Rightarrow F_{W(H_q)}$$

Burada $F_i\#(h_{qi}) \Rightarrow F_{i+1}$ işleminin anlamı $F_i\#(h_{qi})$ F_i'den F_{i+1} ifadesini oluşturur demektir. Bu gösterim sonrasında F_i ve F_{i+1} ifadeleri arasındaki fonksiyonel ilişki aşağıdaki gibi olur.

$$F_{i+1} = F_i\#h_{qi}, \quad i = 1, 2, \ldots, W(H_q) \tag{5.6}$$

Denklem 5.6'yı kullanarak denklem 5.5'i aşağıdaki gibi yazabiliriz.

$$D = (h_{q1} \wedge F_1) \vee (h_{q2} \wedge F_2) \vee \ldots \vee (h_{qW(H_q)} \wedge F_{qW(H_q)}) = \bigvee_{i=1}^{W(H_q)} (h_{qi} \wedge F_i) \tag{5.7}$$

Denklem 5.7'de, $1 \le W(H_q) \le N$, $q \in \{1,2,\ldots Q\}$ ve h_{qi} ise H_q teriminin i. komponenti manasına gelir.

Şu açıktır ki $\varphi(h_{qi}) > \varphi(h_{qi+1})$ olduğu durumlarda denklem 5.7'nin hesaplama karmaşıklığı $i = 1, 2, \ldots, W(H_q)-1$ değerleri için minimum değerde olacaktır. Fakat gerçek veri kümelerinde bu durum genellikle gerçekleşmez. Bu yüzden denklem 5.7'deki $(h_{qi} \wedge F_i)$ bileşeni $i=1,2, \ldots, W(H_q)$ değerleri için $(A^i_{max} \wedge F_i)$ bileşeni ile değiştirilmelidir. Burada A^i_{max}, F_i içerisinde en yüksek frekansa sahip olan özelliktir. Mesela, $F_i=(A_5 \vee A_6 \vee A_9) \wedge (A_5 \vee A_8) \wedge (A_3 \vee A_6)$ fonksiyonun ele alalım. Bu fonksiyondaki özelliklerin frekansları $\varphi(A_3)=1$, $\varphi(A_5)=2$, $\varphi(A_6)=2$, $\varphi(A_8)=1$ ve $\varphi(A_9)=1$ şeklindedir. En yüksek frekansa sahip olan özellikler A_5 ve A_6 özellikleridir ve bu fonksiyonu bölmek için A_5 ve A_6 özelliklerinden her hangi biri seçilebilir. Bundan başka, özellik sayısı N olan bir veri kümesi için $(A^i_{max} \wedge F_i)$ şeklinde $N-1$ tane değişik bileşen bulunabilir. Yukarıda anlatılanlara göre de denklem 5.7 aşağıdaki gibi yazılabilir.

$$D = \bigvee_{i=1}^{N-1} (A^i_{max} \wedge F_i) \tag{5.8}$$

Denklemler 5.6 ve 5.8'e göre, i. iterasyonda F_i fonksiyonu D_i ve F_{i+1} olmak üzere aşağıdaki iki alt fonksiyona bölünür.

$$D_i = F_i \wedge A^i_{max} \tag{5.9}$$

$$F_{i+1} = F_i \# A^i_{max} \tag{5.10}$$

Burada, $Q(D_i) << Q(F_{i+1})$ olacağı için, D_i alt fonksiyonu aynı iterasyonda DNF'ye çevrilir, F_{i+1} alt fonksiyonu ise bir sonraki iterasyonda kullanılmak üzere hazır bekletilir. D_i'nin i. iterasyonda DNF'ye çevrilmesiyle elde edilen DNF_i daha önce elde edilen DNF'ler ile birleştirilir. Böylece aşağıda gösterilen denklem 5.11 elde edilir.

$$D_{DNF} = \bigvee_{i=1}^{N-1} (DNF_i) \tag{5.11}$$

İterasyonlar $F_{j \in \{1,2,\ldots N-1\}}$ fonksiyonu 0 değerine eşit oluncaya kadar devam eder. Bu yaklaşımı anlatan özyinelemeli (recursive) *ÇEVİR_FF_DNF* algoritması aşağıda Şekil 5.1'de gösterilmektedir.

ÇEVİR_FF_DNF (F_i)

{

i=1

$F_1 = DF_{min}$

$D_{DNF}=0$

1. F_i fonksiyonu içerisindeki bütün özelliklerin frekansları hesaplanır ve en yüksek frekansa sahip A^i_{max} özelliği seçilir. Eğer bu şekilde birden fazla böyle özellik varsa bu özelliklerden her hangi biri A^i_{max} olarak seçilebilir.
2. Denklem 5.9'daki gibi A^i_{max} ile F_i fonksiyonu lojik çarpılarak D_i fonksiyonu elde edilir
3. A^i_{max} tarafından yutulan bütün bileşenler D_i fonksiyonundan silinerek D_i fonksiyonu indirgenir.
4. D_i fonksiyonu DNF_i'ye çevrilir.
5. Denklem 5.11 kullanılarak $D_{DNF} = D_{DNF} \vee DNF_i$ işlemi gerçekleştirilir.
6. Denklem 5.10 kullanılarak $F_{i+1} = F_i \# A^i_{max}$ elde edilir.
7. F_{i+1} içerisindeki diğerleri tarafından yutulan bütün bileşenler silinerek F_{i+1} indirgenir.
8. i=i+1
9. Eğer F_i=0 ise Return (D_{DNF})
10. *ÇEVİR_FF_DNF* (F_i)

}

Şekil 5.1. İndirgenmiş fark fonksiyonunu iteratif olarak bölerek bütün özelliklerin minimal alt kümelerini elde eden algoritma

Örnek: Bu örnekte *ÇEVİR_FF_DNF*(F_i) algoritması kullanılarak $D = (e \vee r) \wedge (d \vee r) \wedge (d \vee e \vee f)$ (Komorowski ve ark., 1999) FF_{min}'inin bütün ÖMAK'ları elde edilecektir. Örnekte diğer bileşenler tarafından yutulan bileşenlerin üzeri çizilmiştir.

İterasyon 1: $i = 1$; $F_1 = D = (e \vee r) \wedge (d \vee r) \wedge (d \vee e \vee f)$; $D_{DNF.} = 0$

1.1. φ(d)=φ(e)=φ(r)=2 ve φ(f)=1

A^1_{max} = d

1.2. D_1=(d)∧F_1=(d)∧(e∨r)∧(d∨r)∧(d∨e∨f)

1.3. D_1=(d)∧(e∨r)∧(d∨r)∧(d∨e∨f) =(d)∧(e∨r)

1.4. DNF_1 =((d)∧(e∨r))=de∨dr

1.5. DDNF =DDNF∨DNF_1=0∨de∨dr = de∨dr

1.6. F_2=F_1#(d)=((e∨r)∧(d∨r)∧(d∨e∨f))#d=(e∨r)∧(r)∧(e∨f)

1.7. F_2=(e∨r)∧(r)∧(e∨f)=(r)∧(e∨f)

1.8. $F_2 \neq 0$ olduğu için öz yinelemeli algoritma i=i+1=2 değeri ile devam eder.

İterasyon 2: $i = 2$; $F_2 = (r) \wedge (e \vee f)$; $D_{DNF.} = de \vee dr$

2.1. φ(r)=φ(e)=φ(f)=1

A^1_{max} = e

2.2. D_2=(e)∧F_2=(e)∧(r)∧(e∨f)

2.3. D_2=(e)∧(r)∧(e∨f) = (e)∧(r)

2.4. DNF_2 =((e)∧(r))=er

2.5. DDNF = DDNF ∨DNF_2 = de∨dr∨er

2.6. F_3 =F_2#(e)=((r)∧(e∨f))#e=(r)∧(f)

2.7. F_3 içerisinde silinecek lüzumsuz bileşen bulunmamaktadır.

2.8. $F_3 \neq 0$ olduğu için öz yinelemeli algoritma i=i+1=3 değeri ile devam eder

İterasyon 3: $i = 3$; $F_2 = (r) \wedge (f)$; $D_{DNF.} = de \vee dr \vee er$

3.6. φ(r)= φ(f)=1

A^1_{max} = r

3.7. D3=(r)∧F3=(r)∧(r)∧(f)

3.8. D3=(r)∧(r)∧(f)=(r) ∧(f)

3.9. DNF3 =((r)∧(f))= fr

3.10. DDNF = DDNF ∨DNF3 = de∨dr∨er∨fr

3.11. F4=F3#(r)=((r)∧(f))#r=(0)∧(f)

3.12. F4=(0)∧(f) = (0)

3.13. F3 = 0 olduğu için algoritma sonlanır.

Sonuç olarak elde edilen $D_{DNF} = de \vee dr \vee er \vee fr$ kümesi örnekte verilen $D = (e \vee r) \wedge (d \vee r) \wedge (d \vee e \vee f)$'nin ÖMAK'ları içermektedir ve $\dot{O}MAK = \{\{d,e\},\{d,r\},\{e,r\},\{f,r\}\}$ şeklinde gösterilir. Bunun anlamı da $D = (e \vee r) \wedge (d \vee r) \wedge (d \vee e \vee f)$'nin elde edildiği veri kümesi alt kümeler $\{d,e\}$, $\{d,r\}$, $\{e,r\}$ ve $\{f,r\}$ kümelerinin herhangi birisiyle orijinal veri kümesinden her hangi bir bilgi kaybına uğramadan temsil edilebilir.

Yukarıdaki örnekte de görülebileceği gibi, her bir iterasyonda elde edilen implikantlardan her biri önceki iterasyonlarda elde edilmemiş olan yeni implikanttır. Çünkü denklem 5.8'deki implikantlar asal implikantlardır ve bu yaklaşıma göre başka bir iterasyonda oluşturulamazlar.

Denklemler 5.8, 5.9 ve 5.10 kullanılarak FF_{min}'in DNF'ye çevirme işleminin EKHK'sı $O(3^N / N)$ değerine düşer (Brayton ve ark., 1984, Malik ve ark., 1991). Yukarıda bu yaklaşım ÖLT'de açıklanmıştır. Bu yaklaşım fark fonksiyonu matrisi ve bit tabanlı lojik operatörler kullanılarak aşağıda açıklanacaktır. Böylece, bölüm 3.3'de verilen bir veri kümesinin doğruluk tablosu görüntüsünün elde edilmesi yöntemi her bir veri kümesine uygulanabilecektir.

5.2. Fark fonksiyonu matrisi

Yukarıda bölüm 3'de bir veri kümesinin FM'sini elde etmek için kullanılan ÖLT'deki ifadelerin nasıl BTİ'ye çevrileceği denklem 3.9'da ve bu işlem için kullanılan veri yapısı denklem 3.11 (Struct_ÖLT-BTİ)'de verildi. Bu denklemler kullanılarak 4.3 ve 4.5 denklemleri aşağıdaki gibi yazılabilir.

$$DF_{min}^{b} = \bigmid_{q=1}^{Q} B_q \tag{5.12}$$

$$S = \prod_{q} W(B_q) \tag{5.13}$$

Denklem 5.12, denklem 4.3'deki önerisel AND işlemi yerine bitsel OR operatörünün konulmasıyla elde edilmiştir. Bu denklemlere göre $(A \vee B) \wedge (B \vee C) \xrightarrow{ÖLT/BTİ} \{100,010\}|\{010,001\} = \{\sout{110},101,010,\sout{011}\} \xrightarrow{BTİ/ÖLT} AC \vee B$ çevrimleri gerçekleştirilebilir. Burada ÖLT/BTİ önerisel lojik şeklindeki bir ifadenin bit tabanlı

ifadeye, BTİ/ÖLT ise bit tabalı bir ifadenin önerisel lojik şekline çevirdiğini göstermektedir. Denklem 5.12 kullanılarak işlenen BTİ'ler fark fonksiyonu matrisi (FFM) şeklinde aşağıdaki gibi gösterilebilir.

	A_1	A_2	…	A_N
B_1	b_{11}	b_{12}	…	b_{1N}
B_2	b_{21}	b_{22}	…	b_{2N}
⋮	⋮	⋮	⋮	⋮
B_Q	b_{Q1}	b_{Q2}	…	b_{QN}

Şekil 5.2. Fark fonksiyonu matrisi yapısı

Şekil 5.2.'de verilen FFM'nin her bir satırı denklem 5.12'deki $B_{q\in\{1,2,..Q\}}$ BTİ'lerdir. Yukarıda FF_{min}'i iteratif olarak bölerek DNF'ye çevirme yaklaşımında özelliklerin frekanslarının çok önemli olduğu anlatıldı. Bu yüzden Şekil 5.2'de gösterilen FFM'ye BTİ'lerin ağırlıklarını gösteren bir sütün ve FF_{min}'i oluşturan özelliklerin frekanslarını gösteren bir satır ekleyerek aşağıdaki gibi genişletilebilir.

	A_1	A_2	…	A_N	W(B)
B_1	b_{11}	b_{12}	…	b_{1N}	$W(B_1)$
B_2	b_{21}	b_{22}	…	b_{2N}	$W(B_2)$
⋮	⋮	⋮	⋮	⋮	⋮
B_Q	b_{Q1}	b_{Q2}	…	b_{QN}	$W(B_Q)$
φ(A)	$\varphi(A_1)$	$\varphi(A_2)$	…	$\varphi(A_N)$	

Şekil 5.3. Genişletilmiş fark fonksiyonu matrisi yapısı

Şekil 5.3'deki en sağdaki sütün her bir BTİ'nin ağırlığı W(B)'yi, en alttaki satır ise FF_{min}'i oluşturan özelliklerin frekanslarını gösterir. Mesela, yukarıdaki örnekte işlenen $FF_{\min} = (e \vee r) \wedge (d \vee r) \wedge (d \vee e \vee f)$ fonksiyonu FFM şeklinde aşağıdaki gibi gösterilir.

	d	e	f	r	W(B)
B_1	0	1	0	1	2
B_2	1	0	0	1	2
B_3	1	1	1	0	3
φ(A)	2	2	1	2	

Şekil 5.4. Örneğin genişletilmiş fark fonksiyonu matrisi yapısı

Şekil 5.3'ü ve denklem 5.13'ü kullanarak FFM'den DNF'ye çevirme işleminin EKHK'sı $S = W(B_1) \times W(B_2) \times W(B_3) = 2 \times 2 \times 3 = 12$ olarak hesaplanır.

5.3. Fark fonksiyonu matrisinin disjunktif normal forma çevrilmesi

Buradaki amaç FFM'yi alt FFM'lere ayırıp bu alt FFM'leri ayrı ayrı işleyerek FFM'in DNF'ye çevrilmesi sırasında oluşan hesaplama karmaşıklığını azaltmaktır. Bütün alt FFM'lerde aslında bir FFM olduğu için alt FFM'leri ayrı ayrı işleyebilmek için aşağıda anlatılacak olan *ÇEVİR_ALT-DFM* algoritması geliştirilmiştir. Bu algoritma yukarıda alt bölüm 3.6.1'de anlatılan bit tabanlı ifadelerin genişletilmesini kullanmaktadır. Mesela Şekil 5.3'deki BTİ'lerin genişletilmiş halleri denklem 3.18'e göre aşağıdaki gibi olur.

$$E(B_1) = \{\mathrm{Pr}_i(0101) \mid d_i = 1\} = \{\mathrm{Pr}_2(0101), \mathrm{Pr}_4(0101)\} = \{0100, 0001\}$$
$$E(B_2) = \{\mathrm{Pr}_i(1001) \mid d_i = 1\} = \{\mathrm{Pr}_1(1001), \mathrm{Pr}_4(1001)\} = \{1000, 0001\}$$
$$E(B_3) = \{\mathrm{Pr}_i(1110) \mid d_i = 1\} = \{\mathrm{Pr}_1(1110), \mathrm{Pr}_2(1110), \mathrm{Pr}_4(1110)\} = \{1000, 0100, 0010\}$$

Elde edilen genişletilmiş BTİ'ler kullanılarak Şekil 5.3'deki veri kümesinin denklem 3.19'a göre $DNF_{BB}(D)$ 'si aşağıdaki gibi elde edilir.

$DNF_{BB}(D) = \bigcup_{q=1}^{3} E(B_q) = \{0100,\ 0001\} \mid \{1000,\ 0001\} \mid \{1000,\ 0100,\ 0010\} = \{1100,$ ~~0101~~, ~~1001~~, 0001} | {1000, 0100, 0010} = {1100, ~~1100~~, ~~1110~~, 1001, 0101, 0011} $\xrightarrow{BTİ/ÖLT}$ de $\vee$ dr $\vee$ er $\vee$ fr.

Aşağıdaki şekilde gösterilen *ÇEVİR_ALT-DFM* algoritması ile denklemler 3.18 ve 3.19'u kullanılarak alt FFM'lerin DNF'ye çevrilmesini sağlanır.

ÇEVİR_ALT_DFM(ALT-DFM, Q, N)
{

1. $DNF_{BB}(D) = \{0\}^N$
2. For q=1 to Q
   {
   2.1. $E(B_q) = \{Pr_j(B_q) \mid j \in \{1,2, \ldots, N\}$ and $b_{qj} = 1\}$
   2.2. $DNF_{BB} = DNF_{BB}(D) \mid E(B_q)$
   2.3. DNF_{BB} içerisindeki diğer BTİ'ler tarafından yutulan BTİ'ler silinir.
   }
3. Return (D^b_{DNF})

}

Şekil 5.5. Fark fonksiyonunun alt matrislerini işleyen algoritma

5.4. Fark fonksiyonu matrisini parçalayarak özelliklerin minimal alt kümelerinin tamamını elde eden algoritma

FFM'in DNF'e çevrilme işlemi yukarıda alt bölüm 5.1'de anlatılan FF_{min}'in DNF'ye çevrilmesi işlemine çok benzemektedir. Bu iki çevrim arasındaki kavramsal farklar aşağıdaki gibidir.

1. FF_{min} DNF'ye dönüştürülürken işlemler fonksiyonlar üzerinde yapılırken, FFM'in DNF'ye çevrilirken işlemler matrisler üzerinde yapılır.

2. FF_{min} DNF'ye çevrilme işleminde ÖLT kullanılırken FFM'in DNF'ye çevrilmesine bitsel lojik operatörler kullanılır.

Bölüm 5.1'deki FFM'in DNF'ye çevrilme işleminin i. iterasyonunda işlenen alt fonksiyon F_i, DNF'ye çevrilen alt fonksiyona D_i ve bir sonraki iterasyonda kullanılacak alt fonksiyona F_{i+1} ifadeleri ile tanımlanmıştı. FFM'nin DNF'ye çevrilme işleminde bu alt fonksiyonlar yerine i. iterasyonda FFM'in alt matrisi işlenecek, FFM'in alt matrisi DNF'ye dönüştürülecek ve FFM'in alt matrisi bir sonraki iterasyonda kullanılacaktır. Bu yüzden iki çevrim yöntemi arasındaki farkları korumak için bu alt fonksiyonlara F_i^b, D_i^b ve F_{i+1}^b ifadeleri verilecektir. Bu farklılıklardan dolayı FFM'in DNF'ye çevrilmesi işleminde kullanılacak ifadeler aşağıdaki gibi tanımlanabilir.

$$D_i^b = F_i^b + A_{max}^i,\ i \in \{1,2,\ldots,N-1\} \tag{5.14}$$

$$F_{i+1}^b = F_i^b \# A_{max}^i,\ i \in \{1,2,\ldots,N-1\} \tag{5.15}$$

$$D_{DNF}^b = \bigcup_{i=1}^{N-1} DNF_i^b \tag{5.16}$$

Bu denklemler aşağıda gösterilen bir veri kümesinin bütün ÖMAK'larını elde eden öz yinelemeli algoritma içerisinde kullanılır.

ÇEVİR_ DFM (F_i^b)

{

$F_1^b = DFM$

$D_{DNF}^b = \varnothing$

$i=1$

1. F_i^b fonksiyonu içerisindeki bütün özelliklerin frekansları hesaplanır ve en yüksek frekansa sahip $A_{\max}^i$ özelliği seçilir. Eğer bu şekilde birden fazla böyle özellik varsa bu özelliklerden her hangi biri $A_{\max}^i$ olarak seçilebilir.
2. *Seçilmiş* $A_{\max}^i$ özelliğindeki bit değeri 1 diğer özelliklerin bit değeri 0 sıfır olacak sekilde BTİ M_i oluşturulur. F_i^b ' e oluşturulan M_i ifadesi eklenerek alt FFM D_i^b elde edilir.
3. D_i^b 'den M_i tarafından yutulan bütün satırlar (BTİ'ler) silinir.
4. ***Çevir_Alt-DFM***(D_i^b , Q_i^b , *N)*
5. $D_{DNF}^b = D_{DNF}^b \vee DNF_i^b$
6. F_i^b 'deki $A_{\max}^i$ olarak seçilen sütun alt FFM'den silinerek F_{i+1}^b elde edilir.
7. Alt DFM F_{i+1}^b 'de diğer satırlar tarafından yutulan bütün satırlar silinir.
8. Alt DFM F_{i+1}^b için S_{i+1}^b hafıza karmaşıklığı *hesaplanır.*
9. i=i+1
10. *Eğer* $S_i^b = 0$ *ise* Return (D_{DNF}^b)
11. *ÇEVİR_ DFM* (F_i^b) }

}

Şekil 5.6. Fark fonksiyon matrisini özelliklerine göre parçalayıp özelliklerin minimal alt kümelerini elde eden fonksiyon

Örnek 5.1. : Bu örnekte, D = (A1∨A2∨A4∨A5∨A6∨A7) ∧ (A5∨A6∨A8) ∧ (A5∨A6∨A9) ∧ (A8∨A9) ∧ (A3∨A6∨A9) ∧ (A3∨A6∨A8) FF_{min} 'ini ÇEVİR_DFM algoritmasını kullanarak DNF'sini elde edilecektir. Öncelikle D fonksiyonu içerisindeki

ÖLT'de bulunan bileşenler denklem 3.11'deki veri yapısı kullanılarak aşağıda gösterildiği gibi BTİ'ye çevrilir.

$$H_1 = (A_1 \vee A_2 \vee A_4 \vee A_5 \vee A_6 \vee A_7) \xrightarrow{\text{ÖLT/BTİ}} B_1 = 110111100$$

$$H_2 = (A_5 \vee A_6 \vee A_8) \xrightarrow{\text{ÖLT/BTİ}} B_2 = 000011010$$

$$H_3 = (A_5 \vee A_6 \vee A_9) \xrightarrow{\text{ÖLT/BTİ}} B_3 = 000011001$$

$$H_4 = (A_8 \vee A_9) \xrightarrow{\text{ÖLT/BTİ}} B_4 = 000000011$$

$$H_5 = (A_3 \vee A_6 \vee A_9) \xrightarrow{\text{ÖLT/BTİ}} B_5 = 001001001$$

$$H_6 = (A_3 \vee A_6 \vee A_8) \xrightarrow{\text{ÖLT/BTİ}} B_6 = 001001010$$

Yukarıda açıklanan tarzda elde edilen bu BTİ'ler satır satır FFM'ye eklenir. Ekleme işlemi bittikten sonra her BTİ'nin ağırlığı ve FFM'yi oluşturan her özelliğin frekansı hesaplanır. Sonuç olarak elde edilen FFM aşağıda gösterildiği gibi olur.

	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	W(B)
B_1	1	1	0	1	1	1	1	0	0	6
B_2	0	0	0	0	1	1	0	1	0	3
B_3	0	0	0	0	1	1	0	0	1	3
B_4	0	0	0	0	0	0	0	1	1	2
B_5	0	0	1	0	0	1	0	0	1	3
B_6	0	0	1	0	0	1	0	1	0	3
φ(A)	1	1	2	1	3	5	1	3	3	

Şekil 5.7. D fonksiyonuna ait olan fark fonksiyon matrisi

Elde edilen FFM'in DNF'ye çevrilme işlemi adım adım aşağıda anlatılmaktadır.

İterasyon 1: $i = 1$; $D_{DNF}^{B} = 0$; F_1^B Şekil 5.6'da gösterilen FFM'dir.

1.1. $\varphi(A_1) = \varphi(A_2) = \varphi(A_4) = \varphi(A_7) = 1$, $\varphi(A_3) = 2$, $\varphi(A_5) = \varphi(A8) = \varphi(A9) = 3$ ve $\varphi(6)=5$

$A_{\max}^1 = A_6$

1.2. A_6 özelliğini temsil eden $M_1 = 000001000$ BTİ'si oluşturulur.

1.3. M_1 BTİ'si F_1^B alt FFM'sine eklenerek ve M_1 tarafından yutulan B_1, B_2, B_3, B_5 ve B_6 BTİ'leri F_1^B alt FFM'sinden silinerek D_1^b alt FFM'si aşağıda gösterildiği gibi elde edilir.

	A1	A2	A3	A4	A5	A6	A7	A8	A9	W(B)
B4	0	0	0	0	0	0	0	1	1	2
M1	0	0	0	0	0	1	0	0	0	1

Şekil 5.8. $I_1^b = 2 \times 1 = 2$ hafıza karmaşıklığı değerine sahip D_1^b alt FFM'si

1.4. D_1^b alt FFM'si DNF_1^b'ye çevrilir.

E(B4) = {000000010, 000000001}

$DNF_1^b = E(B_4) \mid M_1$ = {000000010, 000000001} | {000001000} = {000001010, 000001001}

1.5. D_{DNF}^b genişletilir.

$D_{DNF}^b : D_{DNF}^b = D_{DNF}^b \cup DNF_1^b$ = {000001010, 000001001}

1.6. A_6 özelliğini içeren sütun F_1^b'den çıkartılarak F_2^b alt FFM'si oluşturulur.

	A1	A2	A3	A4	A5	-	A7	A8	A9	W(B)
B1	1	1	0	1	1	0	1	0	0	5
B2	0	0	0	0	1	0	0	1	0	2
B3	0	0	0	0	1	0	0	0	1	2
B4	0	0	0	0	0	0	0	1	1	2
B5	0	0	1	0	0	0	0	0	1	2
B6	0	0	1	0	0	0	0	1	0	2
φ(A)	1	1	2	1	3	0	1	3	3	

Şekil 5.9. Alt FFM F_2^b

1.7. F_2^b alt FFM'sinde hiçbir satır diğerleri tarafından yutulamaz.

1.8. F_2^b alt FFM'sinin hafıza karmaşıklığı $F_2^b : S_2^b = 5 \times 2 \times 2 \times 2 \times 2 \times 2 = 160$ değerindedir. $S_2^b > 0$ olduğu için i=i+1=2 değeri ile devam edilir.

İterasyon 2: $i = 2$; $D_{DNF}^B = \{000001010, 000001001\}$; F_2^B Şekil 5.8'da gösterilen alt FFM'dir

2.1. $\varphi(A_1)=\varphi(A_2)=\varphi(A_4)=\varphi(A_7)=1$, $\varphi(A_3)=2$, $\varphi(A_5)=\varphi(A_8)=\varphi(A_9)=3$ olduğu için A_5, A_8 ve A_9 özelliklerinden her hangi biri seçilebilir.

$A^1_{\max} = A_5$

2.2. A_5 özelliğini temsil eden $M_2 = 000001000$ BTİ'si oluşturulur.

2.3. M_2 BTİ'si F_2^B alt FFM'sine eklenerek ve M_2 tarafından yutulan B_1, B_2 ve B_3 BTİ'leri F_2^B alt FFM'sinden silinerek D_2^b alt FFM'si aşağıda gösterildiği gibi elde edilir.

	A_1	A_2	A_3	A_4	A_5	-	A_7	A_8	A_9	W(B)
M_2	0	0	0	0	1	0	0	0	0	1
B_4	0	0	0	0	0	0	0	1	1	2
B_5	0	0	1	0	0	0	0	0	1	2
B_6	0	0	1	0	0	0	0	1	0	2

Şekil 5.10. $I_2^b = 1 \times 2 \times 2 \times 2 = 8$ hafıza karmaşıklığı değerine sahip D_2^b alt FFM'si

2.4. D_2^b alt FFM'si DNF_2^b 'ye çevrilir.

E(B4) = {000000010, 000000001}

E(B5) = {001000000, 000000001}

E(B6) = {001000000, 000000010}

$DNF_2^b = E(B_4) \mid E(B_5) \mid E(B_6) \mid M_2$ = {000000010, 000000001} | {001000000, 000000001} | {001000000, 000000010} = {001010010, 001010001, 000010011}

2.5. D^b_{DNF} genişletilir.

$D^b_{DNF} : D^b_{DNF} = D^b_{DNF} \cup DNF_2^b$ = {000001010, 000001001, 001010010, 001010001, 0000010011}

2.6. A_5 özelliğini içeren sütun F_2^b 'den çıkartılarak F_3^b alt FFM'si elde edilir.

	A_1	A_2	A_3	A_4	A_5	-	A_7	A_8	A_9	W(B)
B_1	1	1	0	1	1	0	1	0	0	5
B_2	0	0	0	0	1	0	0	1	0	2
B_3	0	0	0	0	1	0	0	0	1	2
B_4	0	0	0	0	0	0	0	1	1	2
B_5	0	0	1	0	0	0	0	0	1	2
B_6	0	0	1	0	0	0	0	1	0	2
φ(A)	1	1	2	1	3	0	1	3	3	

Şekil 5.11. F_3^b alt FFM'si

2.7. B_2 ve B_3 satırları tarafından yutulan B_4, B_5 ve B_6 satırları F_3^b alt FFM'sinden silinerek F_3^b indirgenir.

	A_1	A_2	-	A_4	-	-	A_7	A_8	A_9	W(B)
B_1	1	1	0	1	0	0	1	0	0	4
B_2	0	0	0	0	0	0	0	1	0	1
B_3	0	0	0	0	0	0	0	0	1	1
φ(A)	1	1	0	1	0	0	1	1	1	

Şekil 5.12. İndirgenmiş F_3^b alt FFM'si

2.8. F_3^b alt FFM'sinin hafıza karmaşıklığı $F_2^b : S_2^b = 4 \times 1 \times 1 = 4$ değerindedir. $S_2^b > 0$ olduğu için i=i+1=2 değeri ile devam edilir.

İterasyon 3: $i = 3$; $D_{DNF}^b =$ {000001010, 000001001, 001010010, 001010001, 000010011} ; F_3^B Şekil 5.11'da gösterilen alt FFM'dir.

3.1. $\varphi(A_1) = \varphi(A_2) = \varphi(A_4) = \varphi(A_7) = \varphi(A_8) = \varphi(A_9) = 1$ olduğu için A_1, A_2, A_4, A_7, A_8, ve A_9 özelliklerinden her hangi biri seçilebilir.

$A_{\max}^1 = A_8$

3.2. A_8 özelliğini temsil eden $M_3 = 000000010$ BTİ'si oluşturulur.

3.3. M_3 BTİ'si F_3^B alt FFM'sine eklenerek ve M_3 tarafından yutulan B_2 BTİ'si F_3^B alt FFM'sinden silinerek D_3^b alt FFM'si aşağıda gösterildiği gibi elde edilir.

	A_1	A_2	-	A_4	-	-	A_7	A_8	A_9	W(B)
B_1	1	1	0	1	0	0	1	0	0	4
M_3	0	0	0	0	0	0	0	1	0	1
B_3	0	0	0	0	0	0	0	0	1	1

Şekil 5.13. $I_2^b = 4 \times 1 \times 1 = 4$ hafıza karmaşıklığı değerine sahip D_3^b alt FFM'si

3.4. D_3^b alt FFM'si DNF_3^b'ye çevrilir.

E(B1) ={100000000, 010000000, 000100000, 000000100}

E(B3)={000000001}

$DNF_3^b = E(B_1) \,|\, E(B_3) \,|\, M_3 =$ {100000000, 010000000, 000100000, 000000100} | {000000010} | {000000001} = {100000011, 010000011, 000100011, 000000111}

3.5. D_{DNF}^b genişletilir.

$D_{DNF}^b : D_{DNF}^b = D_{DNF}^b \cup DNF_3^b$ = {000001010, 000001001, 001010010, 001010001, 000010011, 100000011, 010000011, 000100011, 000000111}

3.6. A_8 özelliğini içeren sütun F_3^b'den çıkartılarak F_4^b alt FFM'si elde edilir.

	A_1	A_2	-	A_4	-	-	A_7	-	A_9	W(B)
B_1	1	1	0	1	0	0	1	0	0	4
B_2	0	0	0	0	0	0	0	0	0	0
B_3	0	0	0	0	0	0	0	0	1	1
φ(A)	1	1	0	1	0	0	1	0	1	

Şekil 5.14. F_4^b alt FFM'si

3.7. F_4^b alt FFM'sinde hiçbir satır diğerleri tarafından yutulamaz.

3.8. F_4^b alt FFM'sinin hafıza karmaşıklığı $F_4^b : S_4^b = 4 \times 0 \times 1 = 0$ değerindedir. $S_2^b = 0$ olduğu için algoritma sona erer.

Bu örneğin sonucu son iterasyonun 5. adımında elde edilen D_{DNF}^b'dir. D_{DNF}^b 3.11'de verilen veri yapısına göre ÖLT'ye çevrilerek aşağıda gösterilen ÖMAK'lar elde edilir.

ÖMAK={{A_6,A_8}, {A_6,A_9}, {A_3,A_5,A_8}, {A_3,A_5,A_9}, {A_5,A_8,A_9}, {A_1,A_8,A_9}, {A_2,A_8,A_9}, {A_4,A_8,A_9}, {A_7,A_8,A_9}}

Bu örnekten de anlaşılacağı gibi ÇEVİR_DFM algoritması kullanılarak FF_{min}'in DNF'ye çevrilme işleminde $S_2^b = 972$ olan hafıza karmaşıklığı çevirme işleminin 3 iterasyona bölünmesiyle $\max\{2,8,4\} = 8$ değerine düşmüştür.

5.5. ÇEVİR_DFM algoritmasının en kötü hafıza karmaşıklığının hesaplanması

Yukarıda anlatıldığı gibi FF_{min}'in DNF'ye işleminin EKZK'lığı EKHK'lığının karesidir. Bu yüzden, bu çevrimde ÇEVİR_DFM algoritmasının sadece EKZK'lığını hesaplamak yeterlidir.

Yukarıdaki örnekte ve Şekil 5.5'deki ÇEVİR_DFM algoritmasından da anlaşılacağı gibi, birinci iterasyonda F_1^b alt FFM'sinin parçalanması sonucunda D_1^b ve F_2^b alt FFM'leri oluşur. Birinci iterasyonda, D_1^b alt FFM'si DNF'ye çevrildiği için D_1^b'in DNF_1'e çevrilme işleminin EKHK'lığını hesaplamalıyız.

F_1^b içerisinde $A_{k\in\{1,2,\ldots,N\}}$ özelliğinin en çok kullanılan özellik olduğunu varsayalım. ÇEVİR_DFM algoritmasına göre F_1^b'ye M_1 BTİ'si eklenerek ve M_1 BTİ'si tarafından yutulan BTİ'ler silinerek D_1^b alt FFM'si elde edilir. Sonuç olarak D_1^b, $Q-V+1$'den daha fazla satır içeremez. Burada V özelliklerin ortalama frekansıdır ve $V=(n\times Q)/N$ şeklinde elde edilir. Alt FFM D_1^b en fazla $Q-V+1$ satır içerdiği için denklem 4.7 aşağıdaki gibi yazılır.

$$S_{parça} = O(n^{Q-V}) = O(n^{Q-Q\times n/N}) = O(n^{Q(1-n/N)}) \tag{5.18}$$

Burada $1 \le Q \le \binom{N}{N/2}$ olduğu için (Komorowski ve ark., 1999; Wang ve ark., 2007; Nguyen, 2003) $n^{Q(1-n/N)}$ maksimum değerine $Q_{\max} = \binom{N}{N/2}$ ve $n = N/2$ değerlerindeyken ulaşır. Bu değerleri denklem 5.18'de yerlerine koyarsak aşağıdaki denklem 5.19'u elde ederiz.

$$S_{parça} = O((0.5 \times N)^{0.5 \times (\binom{N}{N/2})}) \tag{5.19}$$

Bu birinci iterasyonda elde edilen D_1^b'in DNF'ye çevrilmesi sırasında oluşan EKHK'dır. Denklem 4.5'e göre tüm FFM'in DNF'ye çevrilmesi işleminin EKHK'lığı $O(n^Q)$'dür. Burada n değeri yerine $n = N/2$ ve Q değeri yerine $Q_{\max} = \binom{N}{N/2}$ değerlerini koyarsak aşağıdaki denklem 5.20'yi elde ediriz.

$$S_{tüm} = \mathrm{O}(\mathrm{n}^{\mathrm{Q}}) = \mathrm{O}((0.5 \times N)^{\binom{N}{N/2}}) \tag{5.20}$$

Denklemler 5.19'u ve 5.20'yi karşılaştırırsak aşağıdaki denklemi elde ediriz.

$$S_{parça} = (0.5 \times N)^{0.5 \times (\binom{N}{N/2})} = \sqrt{(0.5 \times N)^{\binom{N}{N/2}}} = \sqrt{S_{tüm}} \tag{5.21}$$

Buradan da ÇEVİR_DFM algoritmasının ilk iterasyonunda oluşan EKHK'lığın algortimanın tamamında oluşan EKHK'lığının karekökü olduğunu söyleyebiliriz (Şirzat ve ark., 2011). Yukarıdaki hesaplamalarda kullanılan F_1^b ve D_1^b arasındaki ilişki bütün F_i^b ve D_i^b $i \in \{1,2,\ldots N-1\}$ çiftleri için geçerlidir.

6. DENEYSEL SONUÇLAR

Bu bölümde, Bölüm 3'te anlatılan Boole fonksiyon yaklaşımı ile bir veri kümesinin indirgenmiş FF'sinin bulunması ve Bölüm 5'te anlatılan indirgenmiş FF_{min}'in iteratif olarak bölünmesiyle ÖMAK'ların bulunması için yapılan deneysel çalışmaların sonuçları verilmektedir.

Geliştirilen yöntemin fark fonksiyonu tabanlı özellik seçme yöntemlerine göre daha hızlı ve daha az hafıza miktarına ihtiyaç duyarak çalıştığını göstermek için deneysel çalışmalar büyük veri kümeleri üzerinde gerçekleştirilmiş ve geliştirilen yöntemin performansı test edilmiştir. Bu deneysel çalışmalarda büyük veri kümelerinin seçilmesindeki amaç fark fonksiyonu tabanlı özellik seçme yöntemlerinin veri kümesindeki özellik sayısı arttıkça ihtiyaç duydukları hafıza miktarının üstel olarak arttığını göstermek ve geliştirilen yönteminin daha az hafızaya ihtiyaç duyarak özellik seçme işlemini gerçekleştirdiğini ispatlamaktır. Deneyler, UCI makine öğrenmesi ambarındaki değişik özelliklere sahip 13 veri kümesi kullanılarak gerçekleştirilmiştir. Kullanılan kümeler ve bu kümelere ait özellikler Çizelge 6.1'de verilmiştir.

Çizelge 6.1. Deneysel çalışmalarda kullanılan veri kümelerinin özellikleri

Veri kümesi	Şart özelliklerinin sayısı	Obje sayısı	Sınıf sayısı
Shuttle	9	43500	5
Heart	13	270	2
Lymn	18	148	4
Mushroom	22	8124	2
Statlog (GCD)	24	1000	2
Chess	36	3196	2
Ionosphere	33	351	2
Statlog (LS)	36	4435	7
Annealing	38	798	6
Spectf Heart	44	187	56
LungCancer	56	32	3
DNA	57	106	2
Sonar	60	208	2

Geliştirilen yöntemin performansının değerlendirilebilmesi için, elde edilen sonuçlar RSES programının sonuçlarıyla karşılaştırılmıştır. Yapılan deneylerde kullanılan bilgisayar Intel Core2Quad@2.83 GHz işlemcili, 4GB RAM'lı olup Microsoft XP Professional sürümü İşletim Sistemi ile desteklenmektedir.

Bölüm 3'te yapılan deneylerde de olduğu gibi elde edilen sonuçlar geliştirilen yöntem ile RSES arasında 3 esas değer temel alınarak karşılaştırılmaktadır. Bunlar:

Elde edilen ÖMAK sayısı, kullanılan hafıza miktarları ve tüketilen işlemci zamanlarıdır. Bunlardan birincisi elde edilen ÖMAK'ların karşılaştırılması aşağıda Çizelge 6.2'de gösterilmektedir.

Çizelge 6.2. Elde edilen ÖMAK'lara göre geliştirilen yöntem ve RSES'in karşılaştırılması

Veri kümesi	**Elde edilen ÖMAK sayısı**		**Elde edilen reduct sayısı**		**Elde edilen reductların büyüklüğü**	
	RSES	**Geliştirilen yöntem**	**RSES**	**Geliştirilen yöntem**	**RSES**	**Geliştirilen yöntem**
Shuttle	19	19	19	19	4	4
Heart	109	109	18	18	3	3
Lymn	424	424	1	1	6	6
Mushroom	292	292	13	13	4	4
Statlog (GCD)	2424	2424	1	1	5	5
Chess	4	4	4	4	29	29
Ionosphere	**Başarısız**	**4257**	**Başarısız**	**6**	**Başarısız**	**2**
Statlog (LS)	**Başarısız**	**1088611**	**Başarısız**	**1145**	**Başarısız**	**5**
Annealing	**Başarısız**	**275**	**Başarısız**	**6**	**Başarısız**	**6**
Spectf Heart	**Başarısız**	**26454**	**Başarısız**	**1**	**Başarısız**	**2**
LungCancer	**Başarısız**	**9007859**	**Başarısız**	**6**	**Başarısız**	**4**
DNA	**Başarısız**	**6259767**	**Başarısız**	**1**	**Başarısız**	**3**
Sonar	**Başarısız**	**31844**	**Başarısız**	**168**	**Başarısız**	**2**

Geliştirilen yöntem ve RSES 13 tane veri kümesinde ayrı ayrı özellik seçme işlemine tabi tutulmuştur. Kullanılan bu 13 tane veri kümesinin 6 tanesi her iki yöntem ile işlenebilmiştir ve sonuç olarak aynı ÖMAK'lar elde edilmiştir. Örneğin Mushroom veri kümesinin her iki yöntemle de işlenmesi sonucu 292 tane ÖMAK elde edilmiştir. Bu elde edilen ÖMAK'ların 13 tanesi Reduct'tır ve Reduct'ların 4 tane özellikten oluşmaktadır. Yani orijinalde 22 özellikten oluşan Mushroom veri kümesi 4 özellikle ifade edilebilecek şekilde indirgenmiştir. Kalan 7 veri kümesinde ise RSES hafıza yetersizliğinden dolayı başarısız olmasına rağmen geliştirilen yöntem bu veri kümelerinin ÖMAK'larını bulmuştur. Yani bir veri kümesinin özellik sayısı arttıkça geliştirilen yöntemin RSES'e göre daha iyi performans gösterdiği görülmüştür.

Çizelge 6.3. Kullanılan hafıza miktarına göre geliştirilen yöntem ve RSES'in karşılaştırılması

Veri kümesi	**Kullanılan Hafıza (MB)**	
	RSES (W_1)	**Geliştirilen yöntem (W_2)**
Shuttle	8,3	8
Heart	1	1
Lymn	1	1
Mushroom	2,3	2
Statlog (GCD)	1	1
Chess	664	2
Ionosphere	**>1800**	**3,2**
Statlog (LS)	**>1800**	**295**
Annealing	**>1800**	**2,8**
Spectf Heart	**>1800**	**2,7**
LungCancer	**>1800**	**500**
DNA	**>1800**	**265**
Sonar	**>1800**	**1**

Geliştirilen yöntem ile RSES'in kullandığı hafıza miktarlarına göre karşılaştırılması yukarıda Çizelge 6.3' de gösterilmektedir. Özellik sayıları 20 civarında olan ilk 5 veri kümesi için geliştirilen yöntem ve RSES yaklaşık aynı miktarda hafıza kullanarak özellik seçme işlemini çalıştırabilmiştir. Fakat kalan 8 veri kümesinde geliştirilen yöntem RSES'e göre çok daha az hafıza miktarı kullanmıştır. Bu veri kümelerinde RSES 1800 MB hafızaya ulaştıktan sonra hafıza yetersizliği hatası vererek programı sonlandırmıştır, bu veri kümeleri Çizelge 6.3' de koyu renk ile gösterilmiştir. Bu satırlardan geliştirilen yöntemin RSES'e göre ne kadar az hafızaya ihtiyaç duyduğu görülebilmektedir.

Çizelge 6.4. Kullanılan işlemci zamanına göre geliştirilen yöntem ve RSES'in karşılaştırılması

Veri kümesi	**Kullanılan işlemci zamanı (saniye)**	
	RSES (T_1)	**Geliştirilen yöntem (T_1)**
Shuttle	1960	3284
Heart	0,39	0,13
Lymn	19,8	0,16
Mushroom	132,4	142,8
Statlog (GCD)	246	2,1
Chess	54,7	29,1
Ionosphere	**>14160**	**1,6**
Statlog (LS)	**>92**	**16320**
Annealing	**>787**	**2,4**
Spectf Heart	**>324000**	**21,120**
LungCancer	**>2670**	**1044210**
DNA	**>4920**	**492726**
Sonar	**>201600**	**27,6**

Geliştirilen yöntem ile RSES'in kullandığı işlemci zamanlarına göre karşılaştırılması yukarıda Çizelge 6.4'de gösterilmektedir. Shuttle ve Mushroom veri kümelerinde RSES'in geliştirilen yönteme göre daha az işlemci zamanına ihtiyaç duyarak özellik seçme işlemini gerçekleştirdiği görülmektedir. Bu iki veri kümesi, özellik sayıları az olmasına rağmen obje sayıları çok olan veri kümeleridir. Bu veri kümelerine bakarak RSES'in az özellik sayılı fakat çok objeli veri kümelerini geliştirilen yönteme göre daha hızlı işleyebildiği söylenebilir. Kalan 11 veri kümesine baktığımız zaman ise özellikle RSES tarafından özellik seçme işlemi tamamlanamamış koyu renk ile gösterilen büyük veri kümelerinin geliştirilen yöntem ile çok daha az işlemci zamanına ihtiyaç duyarak özellik seçme işlemi uygulandığı görülmektedir. Lung Cancer ve DNA veri kümeleri bu deneyde özellik seçme işlemi için en fazla işlemci zamanına ihtiyaç duyulan veri kümeleridir. Bu veri kümeleri için ölçülen işlemci zamanı RSES'in kullandığı işlemci zamanından fazla görünse de Çizelge 6.4'de işlenemeyen veri kümeleri için yazılan T_1 zamanı RSES'in hata verdiği ana kadar ölçülen zamandır. Yani RSES'in bu veri kümelerinde özellik seçme işlemini tamamlayabilmesi için gerekli olan zaman programın hafıza yetersizliği hatası vermesinden dolayı ölçülememiştir.

Sonuç olarak geliştirilen yöntem, veri kümelerinin büyük çoğunluğunda RSES'e göre çok daha iyi performans sergileyerek ÖMAK'ları bulmuştur. Bu sonuçtan da anlaşılacağı gibi geliştirilen yöntem özellik seçme işleminin hesaplama karmaşıklığını azaltarak diğer özellik seçme yöntemleri tarafından çözülemeyen veri kümelerini rahatlıkla çözebilmektedir.

7. SONUÇ

Günümüzde özellik seçme işlemi sıklıkla kullanılan bir konu haline gelmiştir. Özellik seçme işlemi temel olarak iki çeşit yöntemle gerçekleştirilir. Bunlar: sezgisel ve fark fonksiyonu tabanlı özellik seçme yöntemleridir. Sezgisel yöntemlerle yapılan özellik seçme işleminin sonunda bir veri kümesine ait bir veya birkaç tane ÖMAK elde edilir. Fakat bu elde edilen ÖMAK'ın veya ÖMAK'ların veri kümesini temsil eden en iyi ÖMAK olup olmadığı hiçbir zaman bilinemez. Bir veri kümesine ait en iyi ÖMAK'I bulabilmek için, o veri kümesine ait bütün ÖMAK'ları bulmak ve bunlar içerisinden en uygun olanı seçmek gerekir. Buda sadece fark fonksiyonu tabanlı özellik seçme yöntemleri kullanılarak elde edilir. Bu yöntemlerde ki temel problem ise orta ve büyük ölçekli veri kümeleri için ihtiyaç duyulan hafıza ve zaman miktarlarının çok fazla olmasıdır. Bunun sebebi ise fark fonksiyonu tabanlı özellik seçme yöntemlerinin özellik seçme işlemini 2 aşamada gerçekleştirmesidir. Birinci aşamada bir veri kümesine ait fark fonksiyonu elde edilir. İkinci aşamada elde edilen fark fonksiyonu disjunktif normal forma çevrilerek veri kümesine ait ÖMAK veya ÖMAK'lar elde edilir. Birinci aşamada elde edilen fark fonksiyonunun büyük çoğunluğu gereksiz terimlerden oluşur. 19 tane veri kümesi kullanılarak yapılan deneylerde gereksiz terimleri içeren fark fonksiyonundaki terim sayısının indirgenmiş fark fonksiyonundaki terim sayısından en az 8.29 ve en fazla 118000000 kat daha fazla olduğu görülmüştür. Bu gereksiz terimleri içeren fark fonksiyonun disjunktif normal forma çevrilmesi esnasında fazladan işlem yapılır ve böylece ihtiyaç duyulan hafıza ve zaman miktarları üstel olarak artmaktadır. Bu sebeplerden dolayı orta ve büyük ölçekli veri kümelerinde fark fonksiyonu tabanlı özellik seçme yöntemleri hafıza taşmasına uğrayarak özellik seçme işlemini tamamlayamamaktadırlar.

Bu tez çalışmasında fark fonksiyonu tabanlı özellik seçme yöntemlerinde karşılaşılan bu problemlere çözüm bulmak amacıyla yeni bir özellik seçme yöntemi geliştirilmiştir. Bu geliştirilen yeni özellik seçme yöntemi iki aşamadan oluşmaktadır. Birinci aşamada fark fonksiyonu tabanlı özellik seçme yöntemlerinden farklı olarak bir veri kümesinin indirgenmiş fark fonksiyonu Boole fonksiyonu yaklaşımıyla elde edilir. İkinci aşamada ise geliştirilen özellik seçme yöntemi içerisinde tasarlanan fark fonksiyonu matrisi yardımıyla birinci aşamada elde edilen indirgenmiş fark fonksiyonu işlenerek ilgili veri kümesine ait ÖMAK'ların tamamı elde edilir. Geliştirilen bu yeni özellik seçme yöntemi, indirgenmiş fark fonksiyonun oluşturulmasından ÖMAK'ların

elde edilmesine kadar olan bütün işlemleri lojik operatörler kullanarak gerçekleştirir. Buda geliştirilen özellik seçme yöntemin diğer fark fonksiyonu tabanlı özellik seçme yöntemlerine göre daha hızlı ve daha az hafızaya ihtiyaç duyarak çalışmasını sağlayan etkenlerden biridir.

Geliştirilen özellik seçme yöntemin birinci aşamasında öncelikle ilgili veri kümesinin tipine (nümerik, kategorik veya karışık) bakılmaksızın veri kümesindeki özelliklerin değerleri ikili kodlanır. Bu tip bir ikili kodlama sayesinde her türlü veri kümesinin doğruluk tablo görüntüsü elde edilebilir ve geliştirilen özellik seçme yöntemi ile ÖMAK'ları bulunabilir. Veri kümesi ikili kodlanmış hali ilgili veri kümesinin doğruluk tablo görüntüsü verir. Doğruluk tablo görüntüsü elde edilmiş veri kümesi artık lojik operatörler yardımıyla işlenebilir. Daha sonra doğruluk tablo görüntüsündeki mintermler karar değişkenin aldığı değere göre S_{on} ve S_{off} kümelerine ayrılır. S_{on} ve S_{off} kümelerindeki BTİ'ler XOR lojik işlemine tabi tutularak veri kümesinin BT_FF'si elde edilir. Yalnız bu aşamada her XOR işleminden çıkan BTİ, BT_FF'deki diğer BTİ'ler ile karşılaştırılır. Eğer karşılaştırılan BTİ'lerden herhangi biri diğerini yutarsa, yutulan BTİ FF'den silinir. Böylece gereksiz terimlerden arındırılmış indirgenmiş BT_FF elde edilmiş olur. Bu indirgeme işlemi ikinci aşamada yapılacak işlem miktarını azaltır ve böylece geliştirilen yöntemin hafıza ve zaman karmaşıklığı da azaltılmış olur. Buraya kadar yapılan işlemler geliştirilen yöntemin birinci aşamasını oluşturmaktadır. Birinci aşamanın özellik seçme işlemine olan katkısını görebilmek için bu aşama sonucunda elde edilen CNF'deki BT_FF fark fonksiyonu tabanlı özellik seçme yöntemlerinde yapıldığı gibi DNF'ye çevrilmiştir. Bu çevirme işlemi kullanılarak 19 veri kümesi üzerinde yapılan deneylerde fark fonksiyonu tabanlı özellik seçme yöntemleriyle işlenemeyen 6 veri kümesi işlenebilmiştir. Buda bize gereksiz terimlerin FF içerisinden silinmesinin ve işlemlerin lojik operatörler yardımıyla BTİ'ler kullanılarak yapılmasının özellik seçme işlemine olan katkısını göstermektedir. Fakat bu aşamada yapılan deneylerde veri kümesi büyüdükçe CNF'nin DNF'ye çevrilme işleminin ihtiyaç duyduğu hafıza miktarının üstsel olarak arttığı görülmüştür. Bu üstsel olarak artan karmaşıklığa çözüm bulmak amacıyla bu çevirme işlemi fark fonksiyonu matrisi adı verilen bir matris yardımıyla yapılmıştır. FFM'nin oluşturulması ve FFM yardımıyla ÖMAK'ların elde edilmesi de geliştirilen özellik seçme yöntemin ikinci aşamasını oluşturmaktadır.

FFM elde edilen indirgenmiş BT_FF'ye bir matris olarak yorumlamaktadır. BT_FF içerisindeki her BTİ'nin FFM içerisine yerleştirilmesiyle FFM'nin tamamı elde

edilir. Bu aşamadan sonra FFM ilgili veri kümesinin en önemli veya gerekli özelliğinden başlanarak iteratif olarak işlenir. FFM için en önemli özellik en yüksek frekansa sahip özelliktir. Her iterasyonda en önemli özelliğin işlenmesinde ki amaç ilgili iterasyon sonunda elde edilen ÖMAK'ların ilgili veri kümesine ait bütün ÖMAK'ların sayısının büyük çoğunluğunu oluşturması ve bir sonraki iterasyona daha basit ve indirgenmiş bir FFM'nin bırakılmasıdır. Bunun yanı sıra FFM'nin iteratif olarak işlenmesindeki amaç ise özellik seçme işleminde ihtiyaç duyulan hafıza karmaşıklığının azaltılmasıdır. Böylece daha düşük hafıza karmaşıklığına sahip geliştirilen özellik seçme yöntemi daha büyük veri kümelerini daha hızlı ve daha az hafızaya ihtiyaç duyarak işleyebilir. Bu avantajı göstermek için büyük ölçekli 13 veri kümesi üzerinde deneyler yapılmıştır. Yapılan deneylerin sonucunda fark fonksiyonu tabanlı özellik seçme yöntemiyle işlenemeyen 7 veri kümesi geliştirilen özellik seçme yöntemi ile başarıyla işlenebilmiştir.

Sonuç olarak fark fonksiyon tabanlı özellik seçme yöntemlerde görülen hafıza karmaşıklığı problemine geliştirilen özellik seçme yöntemi ile çözüm bulunmuştur. Özellikle, veri kümesinin özellik sayısı arttıkça fark fonksiyonu tabanlı özellik seçme yöntemlerinin ihtiyaç duyduğu hafıza miktarının üstsel olarak arttığı görülürken, geliştirilen özellik seçme yönteminde bu miktarın çok fazla değişmediği görülmüştür. Böylece fark fonksiyonu tabanlı özellik seçme yöntemleriyle işlenemeyen birçok veri kümesi geliştirilen özellik seçme yöntemi ile işlenebilmiştir.

Geliştirilen özellik seçme yöntemiyle elde edilen ÖMAK'lar ile yapılan sınıflandırma deneylerinde en iyi sınıflandırma başarısını her zaman Reductların vermediği görülmüştür. Bunun için gelecekte geliştirilen özellik seçme yöntemi ile elde edilen ÖMAK'lar içerisinden en iyi sınıflandırma başarısını veren ÖMAK'ı otomatik olarak seçen bir yöntem geliştirilebilir.

KAYNAKLAR

Bazan, J., Nguyen, S.H. ve Synak, P., Wroblewski, J.,2000, Rough set algorithms in classification problem, *In L. Polkowski, S. Tsumoto, & T. Y. Lin (Eds.), Rough set methods and applications* Heidelberg, New York: Physica-Verlag, 49–88.

Bazan, J.G., 1998, A comparison of dynamic and non-dynamic rough set methods for extracting laws from decision tables, in Rough Sets, *In Knowledge Discovery 1: Methodology and Applications, L. Polkowski and A. Skowron (eds.)*, New York, Physica-Verlag.

Bazan, J.G., Skowron, A. ve P. Synak, 1994, Dynamic reducts as a tool for extracting laws from decision tables, *Lecture Notes in Artificial Intelligence*, 869, 346-355.

Bazan J.G., Szczuka, M.S. ve Wróblewski J., 2002, A new version of rough set exploration system. In: James J. Alpigini, James F. Peters, Andrzej Skowron, Ning Zhong, redaktorzy, Third International Conference on Rough Sets and Current Trends in Computing RSCTC, volume 2475, *Lecture Notes in Artificial Intelligence*, Malvern, PA, October 14-16, 397-404.

Bazan J.G., Szczuka M.S., 2000, RSES and RSESlib - A Collection of Tools for Rough Set Computations, *Extended version of paper presented at RSCTC'2000*

Bensch M., Schroder M., Bogdan M., Rosenstiel W., Czerner P., Montino R., Soberger G., Linke P. ve Schmidt R., 2005, Feature selection for high-dimensional industrial data, *ESANN* Brugge.

Biesiada, J. ve Duch, W., 2008, Feature selection for high-dimensionaldata, Pearson redundancy based filter, *Advances in Soft Computing*, 45, 242–249.

Blum, A. ve Langley, P., 1997, Selection of relevant features and examples in machine learning, *Artificial Intelligence*, 97(1-2), 245-271.

Brayton, R.K., Hachtel, G.D., McMullen, C.T. ve Singiovanni-Vincentelli, A., 1984, Logic Minimization Algorithms for VLSI Synthesis, *Kluwer Academic Publishers*, Boston, 1-194.

Chen W., Tseng S. ve Hong T., 2008, An efficient bit-based feature selection method, *Expert Systems with Applications*, 34(4), 2858-2869.

Colak, S. ve Isik, C., 2003, Feature subset selection for blood pressure classification using orthogonal forward selection, *Proceedings of 2003 IEEE 29th Annual Bioengineering Conference*, 122–123.

Cotter, S.F., Kreutz-Delgado K. ve Rao B.D., 2001, Backward sequential elimination for sparse vector selection, *Signal Processing*, 8, 1849–1864.

Chouchoulas, A., 2001, Incremental feature selection based on rough set theory, *PhD Disseration of the University of Edinburgh*.

Das S. Filters, 2001, Wrappers and a boosting-based hybrid for feature selection, *Proceedings of the International Conference on Machine Learning*, 74–81.

Dash, M. ve Liu, H., 1997, Feature selection for classifications, *Intelligent Data Analysis: An International Journal*, 1, 131-156.

Degang, C., Changzhong, W. ve Qqingha, H., 2007, A new approach to attribute reduction of consistent and inconsistent covering decision systems with covering rough sets. *Information Sciences*, 177, 3500–3518.

Fatourechi M., Birch G. ve Ward R.K., 2007, Application of a hybrid wavelet feature selection method in the design of a self-paced brain interface system, Journal of Neuro engineering and Rehabilitation, 4(11).

Gheyas, I.A. ve Smith L.S., 2010, Feature subset selection in large dimensionality domains, *Pattern Recognition*, 43(1), 5-13.

Hacibeyoglu, M., Basciftci, F.ve Kahramanli S., 2011, A logic method for efficient reduction of the space complexity of the attribute reduction problem, *Turkish Journal of Electrical Engineering and Computer Sciences*, 19(4), 643-656.

Hall M.A., 1998, Correlation-based feature selection for machine learning, *Ph.D. thesis, Department of Computer Science, University of Waikato*, Hamilton, New Zealand.

Hall, M.A. ve Holmes, G., 2003, Benchmarking attribute selection techniques for discrete class data set mining, IEEE Transactions on Knowledge and Data Set Engineering, 15(3).

Hua, J.P., Tembe, W. ve Dougherty E.R., 2008, Feature selection in the classification of high-dimension data, *IEEE International Workshop on Genomic Signal Processing and Statistics*, 1–2.

Huang J., Cai Y. ve Xu X., 2006, A wrapper for feature selection based on mutual information, *18th International Conference on Pattern Recognition*, 2, 618–621.

Jensen, R. ve Shen, Q., 2007, Rough set based attribute selection: A review, *http://cadair.aber.ac.uk/ dspace/handle/2160/490*.

Jensen, R. ve Shen, Q., 2004, Semantics-preserving dimensionality reduction: Rough and fuzzy-rough based approaches, *IEEE Transactions on Knowledge and Data Engineering*, 16(12), 1457-1471.

Jin, X., Xu, A., Bie, R. ve Guo, P., 2006, Machine learning techniques and chi-square feature selection for cancer classification using SAGE gene expression profiles, *LectureNotes in Computer Science*, 3916, 106–115.

Johnson, D.S., 1974, Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences*, 9(3), 256-278.

Johnsonbaugh, R. ve Schaefer, M., 2004, Algorithms, Pearson Education Inc., 1-752

Kahramanli, S., Hacibeyoglu, M. ve Arslan, A., 2011, A Boolean function approach to feature selection in consistent decision information systems, *Expert Systems with Applications*, 38(7), 8229-8239.

Kahramanli, S., Hacibeyoglu, M. ve Arslan, A., 2011, Attribute Reduction by Partitioning the Minimized Discernibility Function, *International Journal of Innovative Computing Information and Control*, 7(5A), 2167-2186.

Kohavi, R. ve John, G., 1997, Wrappers for feature subset selection, *Artificial Intelligence*, 97, 273-324.

Komorowski, J., Polkowski, L. ve Skowron, A., 1999, Rough set: A tutorial, *http://folli.loria.fr/cds/1999/library/pdf/skowron.pdf*.

Lee, T.T., Lo T.Y. ve Wang J., 2006, An information-lossless decomposition theory of relational information systems, *IEEE Transactions on Information Theory*, 52(5), 1890–1903.

Li X.L., Du Z.L., Wang T. ve Yu D.C., 2005, Audio Feature Selection Based on Rough Set, *International Journal of Information Technology*, 11(6), 117-123.

Liao C., Li S. ve Luo Z. ,2007, Gene selection using Wilcox on rank sum test and support vector machine for cancer, *Lecture Notes in Computer Science*, 4456, 57–66.

Liu, H. ve Yu, L., 2005, Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Set Engineering*, 17(4), 491–502.

Malik, A., Brayton, R.K., Newton, A.R. ve Sangiovanni-Vincentelli, A., 1991, Reduced offsets for minimization of binary-valued functions, IEEE Transactions on Computer-Aided Design, 10(4), 413-426.

Ng H.T., Goh W.B. ve Low K.L., 1997, Feature selection, perceptron learning, and asusability case study for text categorization, *20th Annual International ACMSIGIR Conference on Research and Development in Information Retrieval*, Philadelphia, 67–73.

Nguyen H.S., 2003, On the desision table with maximal number of reducts, *Electronic Notes in Theoretical Computer Science*, 82(4), 1-8.

Ohrn, A., Komorowski, J., Skowron, A. ve Synak, R., 1999, The design and implementation of a knowledge discovery toolkit based on rough sets: The Rosetta system, *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, Physica Verlag, Heidelberg, 376-399.

Osei-Bryson K.M., Giles K. ve Kositanurit B., 2003, Exploration of a hybrid feature selection algorithm, *Journal of the Operational Research Society*, 54, 790–797.

Pawlak, Z., 1991, Rough sets. Theoretical aspects of reasoning about data, *Boston: Kluwer Academic Publishers*.

Peng H., Long F. ve Ding C., 2005, Feature selection based on mutual information criteria of max-dependency, max-relevance, and min redundancy, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27, 1226–1238.

Pudil, P., Novovicov, J. ve Kittler, J., 1994, Floating search methods in feature selection, *Pattern Recognition Letters*, 15(11), 1119–1125.

Rocchi L., Chiari L. ve Cappello, A., 2004, Feature selection of stabile metric parameters based on principal component analysis, *Medical and Biological Engineering and Computing*, 42, 71–79.

Ronen, M. ve Jacob Z., 2006, Using simulated annealing to optimize feature selection problem in marketing applications, *European Journal of Operational Research*, 171, 842–858.

Shazzad K.M. ve Park J.S., 2005, Optimization of intrusion detection through fast hybrid feature selection, *Proceedings of the Sixth International Conference on Parallel and Distributed Computing*, IEEE Computer Society, Washington, DC, USA, 264–267.

Sivagaminathan R.K. ve Ramakrishnan S., 2007, A hybrid approach for feature subset selection using neural networks and ant colony optimization, *Expert Systems with Applications*, 33(1), 49-60.

Skowron, A., 1990, The rough sets theory and evidence theory, *Fundamenta Informaticae*, 13, 245–262.

Skowron A. ve Rauszer C., 1992, The discernibility matrices and functions in information systems, Fun-damenta Informaticae, 15(2), 331-362.

Somol P., Novovicova J., Pudil P., 2007, Notes on the evolution of feature selection methodology, Kybernetika, 43(5), 713-730.

Starzyk, J.A., Nelson, D.E. ve Sturtz, K., 2000, A mathematical foundation for improved reduct generation in information systems, *Journal of Knowledge and Information Systems*, vol.2, no.2, 131-146.

Swiniairski, R.W., 2001, Rough sets methods in feature reduction and classification. *International Journal of Applied Mathematics and Computer Science*, 11(3), 565–582.

Swiniarski, R.W. ve Skowron, A., 2003, Rough set methods in feature selection and recognition, *Pattern Recognition Letters*, 24(6), 833-849.

Tan F., Fu X., Wang H., Zhang Y. ve Bourgeois A., 2006, A hybrid feature selection approach for micro array gene expression data, *Lecture Notes in Computer Science*, 3992, 678–685.

Tan, S., Cheng, X. ve Xu, H., 2007, An efficient global optimization approach for rough set based dimen-sionality reduction, *International Journal of Innovative Computing*, Information and Control, 3(3), 725-736.

Thangavel, K., Jaganathan, P., Pethalakshmi, A. ve Karnan, M., 2005, Effective classification with improved quick reduct for medical database using rough system. *BIME Journal*, 05(1).

Vieira S.M., Sousa M.C. ve Runkler T.A., 2007, Ant colony optimization applied to feature selection in fuzzy classifiers, *Lecture notes in computer science*, 4529, 778–788.

Vinterbo S., ve Ohrn, A., 2000, Minimal approximate hitting sets and rule templates, International Journal of Approximate Reasoning, 25(2), 123-143.

Wang, J., ve Wang, J., 2001, Reduction algorithms based on discernibility matrix: The ordered attributes method, Journal of Computer Science & Technology, 16(6), 489-504.

Wang, X.Y., Yang, J., Teng, X.L., Xia, WJ., ve Jensen, R., 2007, Feature selection based on rough sets and particle swarm optimization, *Pattern Recognition Letters*, 28, 459–471.

Xie, Z. X., Hu, Q. H. ve Yu, D.R., 2006, Improved feature selection algorithm based on SVM and correlation, *Lecture notes in computer science*, 3971, 1373–1380.

Yan Z. ve Yuan C., 2004, Ant colony optimization for feature selection in face recognition, *Lecture notes in Computer Science*, 3072, 221–226.

Yang, J. ve Honavar, V., 1998, Feature subset selection using a genetic algorithm, IEEE Intelligent Systems and their Applications, 13, 44–49.

Yao, Y. ve Zhao, Y., 2009, Discernibility matrix simplification for constructing attribute reducts, Information Sciences, 179(7), 867-882.

Yu, L. ve Liu, H., 2003, Feature selection for high-dimensional data: A fast correlation-based filter solution, *Proc. 20th Internat. Conf. Machine Learning*, pp. 856–863.

Zhao, K. ve Wang, J., 2002, A reduction algorithm meeting users requirements, *Computer Science and Technology*, 17, 578-593.

EKLER

EK-1 Geliştirilen yöntem ile Annealing veri kümesinin çözümü

Aşağıda geliştirilen yöntem ile edilen edilmiş olan 38 şart özelliğinden ve 798 objeden oluşan Annealing veri kümesine ait 55 elemandan oluşan FFM gösterilmektedir.

B_1	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_2	0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_3	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_4	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
B_5	0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_6	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_7	0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
B_8	0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0
B_9	0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0
B_{10}	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
B_{11}	0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{12}	0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{13}	1 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{14}	0 0 1 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{15}	0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{16}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{17}	0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{18}	0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
B_{19}	0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{20}	0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{21}	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{22}	0 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{23}	0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{24}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{25}	0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{26}	0 0 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{27}	0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{28}	0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{19}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
B_{30}	0 0 1 0 0 0 1 1 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{31}	0 0 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{32}	0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{33}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{34}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{35}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{36}	0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{37}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0
B_{38}	0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{39}	0 0 1 0 1 1 0 1 0 1 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{40}	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
B_{41}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{42}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
B_{43}	0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_{44}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
B_{45}	0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{46}	0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0
B_{47}	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 1 0 0 0
B_{48}	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{49}	0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_{50}	0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_{51}	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
B_{52}	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
B_{53}	0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0
B_{54}	0 0 0 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B_{55}	0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Burada, Annealing veri kümesinin özellikleri ile FFM'in her satırındaki bitler aşağıdaki veri yapısına göre ilişkilendirilir.

Struct.Annealing{ Unsign a_i; 1}$_{i=1}^{38}$

Geliştirilen yöntem ile yukarıda gösterilen FFM iteratif olarak bölünür. Sonuç olarak Annealing veri kümesine ait aşağıda gösterilen 275 adet ÖMAK elde edilir. Reduct olan ÖMAK'lar koyu renk ile gösterilmiştir.

1	a7a11a15a32a33a34a35	139	a3a6a11a12a14a31a32a33a35
2	a7a11a15a27a32a33a34	140	a3a6a11a12a14a27a32a33a34
3	a7a11a15a19a24a31a32a33a35	141	a3a6a11a12a14a27a31a32a33
4	a7a11a15a19a24a27a31a32a33	142	a3a6a8a10a11a12a15a20a32a33a35
5	a7a11a14a32a33a34a35	143	a3a6a8a10a11a12a15a20a27a32a33
6	a7a11a14a27a32a33a34	144	a3a6a8a10a11a12a14a32a33a35
7	a7a11a14a19a24a31a32a33a35	145	a3a6a8a10a11a12a14a15a27a32a33
8	a7a11a14a19a24a27a31a32a33	146	a3a5a11a12a15a20a24a32a33a34a35
9	a7a9a11a15a31a32a33a35	147	a3a5a11a12a15a20a24a31a32a33a35
10	a7a9a11a15a27a31a32a33	148	a3a5a11a12a15a20a24a27a32a33a34
11	a7a9a11a14a31a32a33a35	149	a3a5a11a12a15a20a24a27a31a32a33
12	a7a9a11a14a27a31a32a33	150	a3a5a11a12a14a20a24a32a33a34a35
13	a7a8a11a15a19a20a24a32a33a35	151	a3a5a11a12a14a20a24a31a32a33a35
14	a7a8a11a15a19a20a24a27a32a33	152	a3a5a11a12a14a20a24a27a32a33a34
15	a7a8a11a14a19a20a24a32a33a35	153	a3a5a11a12a14a20a24a27a31a32a33
16	a7a8a9a11a15a32a33a35	154	a3a5a10a11a12a15a32a33a34a35
17	a7a8a9a11a15a27a32a33	155	a3a5a10a11a12a15a31a32a33a35
18	a7a8a9a11a14a32a33a35	156	a3a5a10a11a12a15a27a32a33a34
19	a6a11a12a15a24a32a33a34a35	157	a3a5a10a11a12a15a27a31a32a33
20	a6a11a12a15a24a27a32a33a34	158	a3a5a10a11a12a14a32a33a34a35
21	a6a11a12a15a19a24a31a32a33a35	159	a3a5a10a11a12a14a31a32a33a35
22	a6a11a12a15a19a24a27a31a32a33	160	a3a5a10a11a12a14a27a32a33a34
23	a6a11a12a14a24a32a33a34a35	161	a3a5a10a11a12a14a27a31a32a33
24	a6a11a12a14a24a27a32a33a34	162	a3a5a8a11a12a15a32a33a34a35
25	a6a11a12a14a19a24a31a32a33a35	163	a3a5a8a11a12a15a31a32a33a35
26	a6a11a12a14a19a24a27a31a32a33	164	a3a5a8a11a12a15a27a32a33a34
27	a6a10a11a12a15a32a33a34a35	165	a3a5a8a11a12a15a27a31a32a33
28	a6a10a11a12a15a27a32a33a34	166	a3a5a8a11a12a14a32a33a34a35
29	a6a10a11a12a14a32a33a34a35	167	a3a5a8a11a12a14a31a32a33a35
30	a6a10a11a12a14a27a32a33a34	168	a3a5a8a11a12a14a27a32a33a34
31	a6a9a11a12a15a32a33a34a35	169	a3a5a8a11a12a14a27a31a32a33
32	a6a9a11a12a15a31a32a33a35	170	a3a5a8a10a11a12a15a32a33a35
33	a6a9a11a12a15a27a32a33a34	171	a3a5a8a10a11a12a15a27a32a33
34	a6a9a11a12a15a27a31a32a33	172	a3a5a8a10a11a12a14a32a33a35
35	a6a9a11a12a14a32a33a34a35	173	a3a4a11a20a24a32a33a34
36	a6a9a11a12a14a31a32a33a35	174	a3a4a11a20a24a31a32a33
37	a6a9a11a12a14a27a32a33a34	175	a3a4a10a11a32a33a34
38	a6a9a11a12a14a27a31a32a33	176	a3a4a10a11a31a32a33
39	a6a8a11a12a15a32a33a34a35	177	a3a4a8a11a32a33a34
40	a6a8a11a12a15a27a32a33a34	178	a3a4a8a11a31a32a33
41	a6a8a11a12a14a32a33a34a35	179	a3a4a8a10a11a32a33
42	a6a8a11a12a14a27a32a33a34	180	a3a4a8a9a11a12a27a32a34
43	a6a8a9a10a11a12a15a32a33a35	181	a3a4a8a9a11a12a27a31a32
44	a6a8a9a10a11a12a15a27a32a33	182	a3a4a7a8a11a12a33a34
45	a6a8a9a10a11a12a14a32a33a35	183	a3a4a7a8a11a12a31a33
46	a6a7a11a15a31a32a33a35	184	a3a4a7a8a11a12a30a33
47	a6a7a11a15a27a31a32a33	185	a3a4a7a8a11a12a28a33
48	a6a7a11a14a31a32a33a35	186	a3a4a7a8a11a12a27a33
49	a6a7a11a14a27a31a32a33	187	a3a4a7a8a11a12a26a33
50	a6a7a8a11a15a32a33a35	188	a3a4a6a8a9a11a12a33a34
51	a6a7a8a11a15a27a32a33	189	a3a4a6a8a9a10a11a12a31a33
52	a6a7a8a11a14a32a33a35	190	a3a4a6a8a9a10a11a12a30a33

53	a5a8a11a12a15a16a19a24a32a33a34a35	191	a3a4a6a8a9a10a11a12a28a33
54	a5a8a11a12a15a16a19a24a27a32a33a34	192	a3a4a6a8a9a10a11a12a27a33
55	5a8a11a12a15a16a19a20a24a31a32a33a35	193	a3a4a6a8a9a10a11a12a26a33
56	5a8a11a12a15a16a19a20a24a27a31a32a33	194	a3a4a5a8a11a12a33a34
57	a5a8a11a12a14a16a19a24a32a33a34a35	195	a3a4a5a8a11a12a32a34
58	a5a8a11a12a14a16a19a24a27a32a33a34	196	a3a4a5a8a11a12a31a32
59	5a8a11a12a14a16a19a20a24a31a32a33a35	197	a3a4a5a8a10a11a12a32
60	a5a8a11a12a14a16a19a20a24a27a31a32a33	198	a3a4a5a8a10a11a12a31a33
61	a5a8a9a11a12a15a19a24a32a33a34a35	199	a3a4a5a8a10a11a12a30a33
62	a5a8a9a11a12a15a19a24a27a32a33a34	200	a3a4a5a8a10a11a12a28a33
63	a5a8a9a11a12a15a19a20a24a31a32a33a35	201	a3a4a5a8a10a11a12a27a33
64	a5a8a9a11a12a15a19a20a24a27a31a32a33	202	a3a4a5a8a10a11a12a26a33
65	a5a8a9a11a12a15a16a32a33a34a35	203	a2a8a11a12a14a15a16a19a21a24a32a33a34a35
66	a5a8a9a11a12a15a16a27a32a33a34	204	a2a8a11a12a14a15a16a19a21a24a31a32a33a35
67	a5a8a9a11a12a15a16a20a24a31a32a33a35	205	2a8a11a12a14a15a16a19a21a24a27a32a33a34
68	a5a8a9a11a12a15a16a20a24a27a31a32a33	206	a2a8a11a12a14a15a16a19a21a24a27a31a32a33
69	a5a8a9a11a12a14a19a24a32a33a34a35	207	a2a8a9a11a12a14a21a32a33a34a35
70	a5a8a9a11a12a14a19a24a27a32a33a34	208	a2a8a9a11a12a14a21a31a32a33a35
71	a5a8a9a11a12a14a19a20a24a31a32a33a35	209	a2a8a9a11a12a14a21a27a32a33a34
72	a5a8a9a11a12a14a19a20a24a27a31a32a33	210	a2a8a9a11a12a14a21a27a31a32a33
73	a5a8a9a11a12a14a16a32a33a34a35	211	a2a6a10a11a12a15a31a32a33a35
74	a5a8a9a11a12a14a16a27a32a33a34	212	a2a6a10a11a12a15a27a31a32a33
75	a5a8a9a11a12a14a16a20a24a31a32a33a35	213	a2a6a10a11a12a14a31a32a33a35
76	a5a8a9a11a12a14a16a20a24a27a31a32a33	214	a2a6a10a11a12a14a27a31a32a33
77	a5a8a9a10a11a12a15a19a20a24a32a33a35	215	a2a6a8a11a12a15a31a32a33a35
78	a5a8a9a10a11a12a15a19a20a24a27a32a33	216	a2a6a8a11a12a15a27a31a32a33
79	a5a8a9a10a11a12a14a19a20a24a32a33a35	217	a2a6a8a11a12a14a31a32a33a35
80	a5a6a10a11a12a15a31a32a33a35	218	a2a6a8a11a12a14a27a31a32a33
81	a5a6a10a11a12a15a27a31a32a33	219	a2a5a8a11a12a15a16a19a24a31a32a33a35
82	a5a6a10a11a12a14a31a32a33a35	220	a2a5a8a11a12a15a16a19a24a27a31a32a33
83	a5a6a10a11a12a14a27a31a32a33	221	a2a5a8a11a12a14a16a19a24a31a32a33a35
84	a5a6a8a11a12a15a31a32a33a35	222	a2a5a8a11a12a14a16a19a24a27a31a32a33
85	a5a6a8a11a12a15a27a31a32a33	223	a2a5a8a9a11a12a15a32a33a34a35
86	a5a6a8a11a12a14a31a32a33a35	224	a2a5a8a9a11a12a15a31a32a33a35
87	a5a6a8a11a12a14a27a31a32a33	225	a2a5a8a9a11a12a15a27a32a33a34
88	a4a8a11a19a24a32a33a34	226	a2a5a8a9a11a12a15a27a31a32a33
89	a4a8a11a19a20a24a31a32a33	227	a2a5a8a9a11a12a14a32a33a34a35
90	a4a8a11a16a32a33a34	228	a2a5a8a9a11a12a14a31a32a33a35
91	a4a8a11a16a20a24a31a32a33	229	a2a5a8a9a11a12a14a27a32a33a34
92	a4a8a10a11a19a20a24a32a33	230	a2a5a8a9a11a12a14a27a31a32a33
93	a4a8a9a11a12a19a20a24a27a32a34	231	a2a5a8a9a10a11a12a15a32a33a35
94	a4a8a9a11a12a19a20a24a27a31a32	232	a2a5a8a9a10a11a12a15a27a32a33
95	**a4a7a11a32a33a34**	233	a2a5a8a9a10a11a12a14a32a33a35
96	**a4a7a11a31a32a33**	234	a2a4a8a11a32a33a34
97	**a4a7a8a11a32a33**	235	a2a4a8a11a31a32a33
98	**a4a7a8a11a12a32**	236	a2a4a8a10a11a32a33
99	**a4a6a11a32a33a34**	237	a2a4a8a9a11a12a27a32a34
100	**a4a6a11a31a32a33**	238	a2a4a8a9a11a12a27a31a32
101	a4a6a8a11a12a32a34	239	a2a4a5a8a11a12a33a34
102	a4a6a8a11a12a31a32	240	a2a4a5a8a11a12a32a34
103	a4a6a8a10a11a32a33	241	a2a4a5a8a11a12a31a32
104	a4a6a8a9a10a11a12a32	242	a2a4a5a8a10a11a12a32
105	a4a5a8a11a12a19a20a24a33a34	243	a2a4a5a8a10a11a12a31a33
106	a4a5a8a11a12a19a20a24a32a34	244	a2a4a5a8a10a11a12a30a33
107	a4a5a8a11a12a19a20a24a31a32	245	a2a4a5a8a10a11a12a28a33
108	a4a5a8a10a11a12a19a20a24a32	246	a2a4a5a8a10a11a12a27a33
109	a4a5a8a10a11a12a19a20a24a31a33	247	a2a4a5a8a10a11a12a26a33
110	a4a5a8a10a11a12a19a20a24a30a33	248	a2a3a11a12a15a21a32a33a34a35
111	a4a5a8a10a11a12a19a20a24a28a33	249	a2a3a11a12a15a21a31a32a33a35
112	a4a5a8a10a11a12a19a20a24a27a33	250	a2a3a11a12a15a21a27a32a33a34
113	a4a5a8a10a11a12a19a20a24a26a33	251	a2a3a11a12a15a21a27a31a32a33
114	a4a5a7a11a12a33a34	252	a2a3a9a11a12a14a21a32a33a34a35
115	a4a5a7a11a12a31a33	253	a2a3a9a11a12a14a21a31a32a33a35
116	a4a5a7a8a11a12a30a33	254	a2a3a9a11a12a14a21a27a32a33a34
117	a4a5a7a8a11a12a28a33	255	a2a3a9a11a12a14a21a27a31a32a33
118	a4a5a7a8a11a12a27a33	256	a2a3a6a8a10a11a12a15a32a33a35
119	a4a5a7a8a11a12a26a33	257	a2a3a6a8a10a11a12a15a27a32a33
120	a4a5a6a8a11a12a33a34	258	a2a3a5a11a12a15a32a33a34a35
121	a4a5a6a8a10a11a12a32	259	a2a3a5a11a12a15a31a32a33a35

122	a4a5a6a8a10a11a12a31a33	260	a2a3a5a11a12a15a27a32a33a34
123	a4a5a6a8a10a11a12a30a33	261	a2a3a5a11a12a15a27a31a32a33
124	a4a5a6a8a10a11a12a28a33	262	a2a3a5a11a12a14a32a33a34a35
125	a4a5a6a8a10a11a12a27a33	263	a2a3a5a11a12a14a31a32a33a35
126	a4a5a6a8a10a11a12a26a33	264	a2a3a5a11a12a14a27a32a33a34
127	a3a7a11a15a31a32a33a35	265	a2a3a5a11a12a14a27a31a32a33
128	a3a7a11a15a27a31a32a33	266	a2a3a4a11a32a33a34
129	a3a7a11a14a31a32a33a35	267	a2a3a4a11a31a32a33
130	a3a7a11a14a27a31a32a33	268	a0a2a8a11a12a14a16a19a21a24a32a33a34a35
131	a3a7a8a11a15a32a33a35	269	a0a2a8a11a12a14a16a19a21a24a31a32a33a35
132	a3a7a8a11a15a27a32a33	270	a0a2a8a11a12a14a16a19a21a24a27a32a33a34
133	a3a7a8a11a14a32a33a35	271	a0a2a8a11a12a14a16a19a21a24a27a31a32a33
134	a3a6a11a12a15a32a33a34a35	272	a0a2a3a11a12a14a21a32a33a34a35
135	a3a6a11a12a15a31a32a33a35	273	a0a2a3a11a12a14a21a31a32a33a35
136	a3a6a11a12a15a27a32a33a34	274	a0a2a3a11a12a14a21a27a32a33a34
137	a3a6a11a12a15a27a31a32a33	275	a0a2a3a11a12a14a21a27a31a32a33
138	a3a6a11a12a14a32a33a34a35		

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı	**:**	Mehmet HACIBEYOĞLU
Uyruğu	**:**	T. C.
Doğum Yeri ve Tarihi	**:**	Konya 1981
Telefon	**:**	0505 445 54 99
Faks	**:**	
e-mail	**:**	hacibeyoglu@selcuk.edu.tr

EĞİTİM

Derece		**Adı, İlçe, İl**	**Bitirme Yılı**
Lise	:	Karatay Anadolu Lisesi	1999
Üniversite	:	Selçuk Üniversitesi Bilgisayar Mühendisliği	2003
Yüksek Lisans	:	Selçuk Üniversitesi Bilgisayar Mühendisliği	2006
Doktora	:	Selçuk Üniversitesi Bilgisayar Mühendisliği	2012

İŞ DENEYİMLERİ

Yıl	**Kurum**	**Görevi**
2004	**Selçuk Üniversitesi Bil. Müh**	**Arş. Gör.**

UZMANLIK ALANI

Yapay zekâ, fark fonksiyon tabanlı özellik seçme

YABANCI DİLLER

İngilizce

YAYINLAR

Kahramanli, S., Hacibeyoglu, M. ve Arslan, A., 2011, A Boolean function approach to feature selection in consistent decision information systems, *Expert Systems with Applications*, 38(7), 8229-8239. (Doktora tezinden yapılmıştır)

Kahramanli, S., Hacibeyoglu, M. ve Arslan, A., 2011, Attribute Reduction by Partitioning the Minimized Discernibility Function, *International Journal of Innovative Computing Information and Control*, 7(5A), 2167-2186. (Doktora tezinden yapılmıştır)

Hacibeyoglu, M., Basciftci, F.ve Kahramanli S., 2011, A logic method for efficient reduction of the space complexity of the attribute reduction problem, *Turkish Journal of Electrical Engineering and Computer Sciences*, 19(4), 643-656. (Doktora tezinden yapılmıştır)