T.C.
SELÇUK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MANTIKSAL FONKSİYONLARIN SADELEŞTİRİLMESİ

İbrahim SAVRAN

YÜKSEK LİSANS TEZİ

BİLGİSAYAR MÜHENDİSLİĞİ

ANABİLİM DALI

Konya - 2006

Yüksek Lisans Tezi
Selçuk Üniversitesi Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Yıl: 2006
Sayfa: 82

ÖZET

Mantıksal fonksiyonlarının sadeleştirilmesi tasarımcılara daha kısa zaman süresinde ve daha sade lojik devreler tasarlama imkânı sağlamaktadır. Fonksiyonların sadeleştirilmesi şu avantajları bize sunmaktadır:

- Güç tüketimi azaltılması,
- Daha küçük hacim,
- Daha az maliyet,

Bu konu ile ilgili olarak tek ve çok çıkışlı fonksiyonların sadeleştirilmesi için çeşitli teknikler geliştirilmiştir. Bu tekniklerin çoğu iki ana aşamada gerçekleştirilir. Birinci aşamada, asal implikantların tümü belirlenir. İkinci adımda fonksiyonu sadeleşmiş olarak örtecek, esas asal implikantlar kümesi belirlenir. Anahtarlama fonksiyonlarını sadeleştirecek algoritmaların tümü $O(2^n)$ karmaşıklığına sahiptirler. Araştırmalar göstermiştir ki *n'* in çok yüksek değerlerinde esas asal implikantların tam kümesini belirleme yöntemi pratik olarak gerçekleştirilemez duruma gelmektedir. Bu yüzden bu doktora tezinde asal implikantların belli kıstaslara cevap verecek alt kümeleri oluşturularak, doğrudan örtme (direct cover) prensibine dayanan bir minimumlaştırma yöntemi geliştirilmiştir.

Anahtar Kelimeler - Mantıksal fonksiyon, sadeleştirme, minimumlaştırma, Boole ifadesi, asal implikant, küp cebri, örtme algoritması, algoritmaların karmaşıklığı, Off-küme tabanlı minimumlaştırma, doğrudan örtme prensibi.

Master Thesis
Selçuk University Graduate School of
Natural and Applied Sciences
Departman of Computer Engineering
Year: 2006
Page: 82

ABSTRACT

The Minimization of Boolean functions allows designers these advantages:

- Fewer components
- Reducing the cost of particular system,
- Reducing power consuming,

Most of single-output and multiple-outputs boolean minimization techniques work on a two step principle, the first step identifies all of the prime implicants (PI' s) and the second step selects the subset of PI' s that covers the function(s) being minimized. All procedures for reducing either two-level or multilevel Boolean networks into prime and irredundant form have $O(2^n)$ complexity. Prime Implicants identification step can be computational impractical as n increases. Thus, in this master thesis, subsets of prime implicants that can prove direct cover principle which based on determineted criters use for mimimization method.

Keyword(s): logic functions, simplification, mimimization, boolean expression, Prime implicant, cube algebra, cover algorithm, complexity, direct cover principal.

TEŞEKKÜR

Bu yüksek lisans tez çalışmalarım boyunca bana yol gösterip her türlü yardımını esirgemeyen tez danışmanım değerli hocam Doç. Dr. Şirzad KAHRAMANLI' ya, akademik çalışmalarımda bana sabırla çalışmamı tavsiye eden Bilgisayar Mühendisliği Bölüm Başkanı Prof. Dr. Ahmet ARLAN' a, tez ve makale araştırma aşamasında bana yol gösteren hocalarım Arş. Gör. Ömer KAAN BAYKAN' a ve Arş. Gör. Ömer Harun UĞUZ 'a, yetişmemde emeği geçen tüm hocalarıma, maddi ve manevi yardımlarını hiçbir zaman esirgemeyen ve daima beni teşvik eden anneme ve babama teşekkür ederim.

İÇİNDEKİLER

SİMGELER

✯	Koordinatlı Çarpma (Coordinate Product, Star Product).
#	Koordinatlı çıkarma (Coordinate Subtraction, Sharp product).
∩	Koordinatlı Kesişme (Coordinate Intersection).
Δ	Değişmeli yutma işlemi (commutative absorption operation).
* - d	Belirsiz ya da keyfi değer (Don't Care).
↔	Ancak ve ancak bağlantısı.
∪	Birleşme işlemi.
m	Minterm (Çarpım Terimi) .
M	Maxterm (Toplam Terimi) .

{0, 1, x} Boolean değişkenin tanımlanma uzayı.

{0,1,d} Boolean fonksiyon tanımlama uzayı.

x	Değişken.
n	Fonksiyonun bağlı olduğu değişken sayısı.
k_i	küpün koordinat ekseni.
$\tilde{k}_i$	k_i koordinat ekseni üzerindeki bir değer.

O(g(n)) Karmaşıklık ifadesi.

L	Fonksiyon için gerekli olan mintermler,
Q	Fonksiyon için yasak olan mintermler,
D	Fonksiyon için gereksiz olan mintermler.
S_{ON}	Doğru mintermlerin kümesi.
S_{OFF}	Yanlış mintermlerin kümesi.
S_{DC}	Fonksiyonun belirlenmemiş olduğu mintermlerin kümesi.
X	Örtülmek için seçilen On-minterm.
$AI_i(x)$	X mintermini kapsayan i. asal implikant.
$S_{AI}(x)$	X minterminin kapsadığı tüm asal implikantların kümesi.
EAI(x)	X mintermin esas asal implikantı.
F	On-kümesi (Espresso Algoritması).
R	Off-kümesi (Espresso Algoritması).
D	Belirsizler kümesi (Espresso Algoritması).
v	Düğüm.

KISALTMALAR

AI Asal İmplikant (Prime Implicant).

AİT Asal İmplikantlar Tablosu (Prime Implicant Table).

EAI Esas Asal İmplikant (Essential Prime İmplikant).

DST Dallandırma ve Sınırlandırma Tekniği (Branch and Bound Technique).

KH Karnaugh Haritası.

LSI Büyük Ölçekli Devre (Large Scale Integrated).

VLSI Çok Büyük Ölçekli Devre (Very Large Scale Integrated).

POS Toplam Terimlerinin Çarpımı (Product of Sum).

NPT Örtüdeki çarpım terimlerinin sayısı.

PM Petrik Fonksiyonu.

PLA Programlanabilir Lojik Diziler (Programmable Logic Arrays).

SFs Anahtarlama Fonksiyonları (Switching Functions).

QMM Quine McCluskey Metodu.

YV Yutma Vektörü (Absorption Vector).

KV Kesişme Vektörü (Vector Intersection).

ÇV Çarpım Vektörü (Vector of Product).

SV Çıkarma Vektörü (Vector of Subtraction).

SOP Çarpım Terimlerinin Toplamı (Sum of Product).

NP Belirsiz Polinomal (Non deterministic Polinominal).

YMÖA Yakın-Minimali Örtme Algoritması (Near-Minimal Cover Algortihm).

1 GİRİŞ

Boole cebri olarak bilinen matematiksel sistem üzerine ilk çalışmalar 1854 yılında George Boole tarafından başlatılmıştır. 1904 yılında Amerikalı Matematikçi E.V. Hungtinton, Boole cebrine yeni aksiyomlar eklemiştir. 1938 yılında Shannon, Boole cebrini devre tasarımlarına uygulamıştır. Bunun sonucunda Anahtarlama Cebri (Switching algebra) adı altında yeni bir bilim dalı ortaya çıkmıştır (Brayton ve ark. 1984). Dijital tasarımın başladığı 1950' li yıllarda lojik kapılar (Logic Gates) pahalı devre elemanlarıydı. Bundan dolayı, verilen lojik fonksiyonu daha az sayıda elektronik elemanla (kapılar ve diyot, direnç gibi kapıların temel bileşenleri) gerçekleştirmek için yeni tekniklerin geliştirilmesinin önemi artmıştır. Böylece o yıllarda, lojik fonksiyonların sadeleştirilmesi araştırmaları çok aktif bir Alan haline gelmiştir. Karnaugh haritaları, iki seviyeli lojik fonksiyonların (Two-Level Logic Functions) sadeleştirilmesi için manüel olarak kullanılmıştır. Bu yöntem 1953 yılında Karnaugh tarafından geliştirilmiştir. Daha sonraları, Quine ve McCluskey (McCluskey 1965) tarafından yeni bir teknik geliştirilmiştir. Bu yöntem 1952 yılında Quine tarafından başlatılmış ve 1956 yılında McCluskey tarafından geliştirilmiştir. Bu yöntem iki aşamadan oluşmaktadır:

1- Bütün asal implikantların (prime implicants - AI) üretilmesi

2- Minimum örtünün oluşturulması.

Bütün asal implikantların üretilmesi çok etkili bir hale gelse de, Hong ve Ostapko tarafından IBM' de geliştirilen MINI (Hong ve ark. 1974) programı, n değişkenli lojik fonksiyonun asal implikantlarının sayısının $3^n/n$ kadar büyük olabileceğini göstermiştir. Buna ek olarak, ikinci adım, genellikle dallandırma ve sınırlandırma tekniği ile gerçekleştirilir. Bu teknik NP-karmaşıklık problemleri sınıfına ait olan minimum örtme probleminin çözümünü içermektedir. Bu ise etkili kesin bir Algoritma bulma ümidini azaltır. Örnek olarak, minimum örtme Algoritmasının çalışma zamanı, örtme problemindeki eleman sayılarındaki bir polinom ile sınırlandırılır. Örtüm probleminin elemanları sayısı lojik fonksiyonunun giriş değişkenlerinin sayısıyla logaritmik olarak orantılı olabileceğinden, bu tekniklerin kullanımı orta ölçekteki problemler için bile pratik değildir (10–15 değişken) (Brayton ve ark. 1984).

Lojik fonksiyonların, sadeleştirilmesinden elde edilen çarpım terimlerinin minimumlaştırılması gerekli fiziksel Alanın üzerinde doğrudan güçlü bir etkisi vardır. Çünkü her bir çarpım terimi, PLA' nın bir satırı olarak gerçekleştirilir. Çok Büyük Ölçekli Devre (VLSI) lojik tasarımı sıklıkla otuzdan daha fazla giriş, çıkış ve çarpım terimli lojik fonksiyonları içerir. Bu durumda kesin sadeleştirme pratik değildir. Bu gibi durumlarda gerekli olan en uygun şekle sokma (optimizasyon), farklı tecrübe yaklaşımları, probleme uygulamaktadır.

Bu yaklaşımlardan bir tanesi klasik lojik sadeleştirme tekniklerinin yapısını takip eder ve birinci olarak tüm asal implikantları üretir. Bununla birlikte minimum bir örtü üretmek yerine yakın minimum bir örtü, tecrübelere dayanarak seçilir. Bu prosedür hala çok yüksek sayıda asal çarpan üretme ihtimali içermektedir. İkinci bir yaklaşım eşzamanlı olarak örtü için implikantları tanımlar ve seçmeye uğraşır. Bıı grupta birkaç tane Algoritma ileri sürülmüştür (Hong ve ark. 1974, Rhyne ve ark. 1977).

Son zamanlarda, sezgisel yaklaşımlar, pratik PLA' ların tasarımında geniş uygulama alanları bulmuştur. Bunların çok erken ve çok başarılı olması, 1970' lerin ortasında IBM' de MINI programının geliştirilmesine sebep olmuştur (Hong ve ark. 1974). Sonraları sezgisel sadeleştirme programı PRESTO, Brown tarafından tanıtılmıştır (Brown 1981). Bu, büyük PLA' ların minimumlaştırılmasına imkân verdi.

1981 yılının yaz aylarında ESPRESSO-I (Brayton ve ark. 1982) programı geliştirilmiştir. ESPRESSO-I sırasıyla gelen işleri kontrol etmek için birçok anahtarı olan tek bir programdır. Bir yıl sonra 1982' nin yazında ESPRESSO-II geliştirilmiştir. Sadeleştirilmiş yanlış küme ve totoloji algoritmalarına dayanan iki yeni metot sunulmuştur. Bu metotlarda verilen sonuçlar Espresso' nun sonuçları kadar iyidir. (Brayton ve Somenzi 1989) bu çalışmalarında Quine-McCluskey metoduna benzer bir yöntem sunmuşlardır, (Lin ve Somenzi 1990) sembolik ilişkilerin basitleştirilmesiyle ilgilenmişlerdir.

Çarpım terimlerinin toplamındaki sadeleştirme ikili (binary) sistem içerisinde önemli bir yer tutmuştur (Tirumalai ve Butler 1991). Son zamanlarda sunulan çarpım terimlerinin toplamı şeklinde sadeleştirme yapan Algoritmaların birçoğu doğrudan örtme metodunu kullanmıştır. Doğrudan örtme metodu üç adım halinde gerçekleştirilir (Tirumalai ve Butler 1991):

a- Mintermin seçilmesi,

b- Asal implikantların üretilmesi

c- Esas asal implikantın seçilmesi ve örtme.

1.1 Anahtarlama Fonksiyonlarının Sadeleştirilmesi için Çözüm Yöntemleri

Kesin (exact) ve segisel (heuristic) SOP minimumlaştırma bilgisayar destekli tasarım (computer aided design-CAD) Alanında çok iyi araştırılan problemlerden bir tanesidir (Mishchenco ve Sasao 2003). SOP minimumlaştırma; PLA optimizasyonunda, çok seviyeli lojik sentezde (muti-level logic synthesis), durum şifrelemede, güç kestirimde, test üretmede ve diğer Alanlarda kullanılır (Mishchenco ve Sasao 2003). Kesin SOP minimumlaştırma probleminin üssel doğasından dolayı modem Algoritmalar, (Brayton ve ark. 1984, Coudert ve Madre 1993, Coudert 1994, McGeer ve ark. 1993) minimum SOP kümesinde yüzlerce çarpım terimi oluncaya kadar sadeleştirilmek istenen fonksiyonu işleyebilir. Bu arada pratik uygulamaların ve CAD araçlarının çoğu buluşsal minimumlaştırmaya dayanır (Brayton ve ark. 1984, Rudell ve Sangiovanni-Vincentelli 1987).

Sezgisel Algoritmaların karmaşıklığı çarpımların sayısında yaklaşık olarak kareseldir (Mishchenco ve Sasao 2003). Bu Algoritmalar kesin (exact) olanlardan fark edilebilecek kadar hızlıdır fakat çok çarpımlı fonksiyonlar için yavaş olabilir (Mishchenco ve Sasao 2003).

Sezgisel SOP minimumlaştırmayı hızlandırmak için çeşitli yaklaşımlar önerilmiştir. Örneğin, Off-kümesinin (Sasao 1985) hesaplaması minimum SOP' da az sayıda çarpımlı fonksiyonlar için bile zaman tüketici olabildiği gözlenmiştir (Mishchenco ve Sasao 2003). Bu yüzden sadeleşmiş off-kümesinin hesaplanması önerilmiştir (Malik ve ark. 1991). Lojik sentez araçları için optimizasyonda genişçe kullanılan başka hızlandırma şekli, buluşsal minimumlaştırmanın sadece bir döngüde gerçekleştirilmesidir. Bu tür kısa yolların bedeli çalışma zamanı problemi hala dururken, daha düşük minimumlaştırma kalitesidir. Bir çok

benchmark için optimizasyon programları buluşsal SOP minimumlaştırmanın uzun çalışma zamanından dolayı sona ermez. (Mishchenco ve Sasao 2003).

Başka hızlı buluşsal SOP minimumlaştırma Algoritmaları BDD gösterimini kullanır (Minato 1992). Bu Algoritma, sonuç kalitesinin kritik olmadığı durumlarda dikkat çekecek derecede iyi çalışır. Ancak (Sasao ve Butler 2001) da gösterilen bu Algoritma (Minato 1992) minimum SOP lardan daha fazla çarpım içeren artıksız (irredundant) SOP lar üretir (Mishchenco ve Sasao 2003). Bu yüzden birçok pratik problemler için uygun değildir. İki seviyeli lojik minimumlaştırma lojik sentezin temel problemidir (Sasao ve Butler 2001). Geniş fonksiyon kümeleri için kesin minimum SOP ifadeleri elde edecek Algoritmalar olmasına rağmen (Coudert 1994), pratik sistemlerin çoğunluğu buluşsal lojik minimumlaştırma algoritmalarını kullanır.

Sasao ve Butler (2001) fonksiyonların sınıflarını, değişken sayısının sınırsız olduğu durumlarda en kötü SOP boyutunun minimum SOP boyutuna oranının büyük olduğunu göstermişlerdir. Sasao ve Butler (2001) verilen fonksiyon için bütün gerekli SOP ifadelerini üreten algoritmayı göstermişlerdir.

1.2 Tezin Amacı ve Önemi

Bilgisayar devreleri ve programlarının mümkün olduğu kadar basit ve etkili kılınması yolunda en etkin olan araçlardan biri lojik fonksiyonlarının minimumlaştırılmasıdır. Halen çoklu miktarda minimumlaştırma yöntemleri mevcuttur. Fakat bunların ürettikleri aralık sonuçlarının sayısı, değişken sayısına göre üssel bir fonksiyonla belirlenir. Bu durumda mesela, 20 değişkenli fonksiyonların minimumlaştırılması sırasında meydana çıkabilecek aralık sonuçlarının sayısı bugün mevcut olan bilgisayarların bellek kapasitesini çok fazla aşmaktadır. Pratikte 40’ a kadar değişken değeri olan fonksiyonların minimumlaştırılması ihtiyacı göz önüne alınınca, mümkün olduğu kadar az sayıda aralık sonuçları üreten bir sadeleştirme algoritmasının elde edilmesine ihtiyaç olduğu şüphesizdir. Bu tezde böyle bir algoritmanın meydana çıkarılması hedeflenmiş ve gerçekleştirilmiştir.
Bu tezde geliştirilen algoritma sayesinde daha az lojik elemanlar kullanılarak yapılamayan programlanabilir lojik dizileri (PLA) kolaylıkla tasarlanabilecek ve bu sayede büyük sayısal

sistemlerin tasarlanmasında donanım ve zaman kaybı büyük ölçüde önlenecektir.

Bu tez çalışması yedi bölümden oluşmuştur.

Birinci bölümde: Konunun tarihsel gelişimi anlatılarak, minimumlaştırma problemlerinin bugünkü durumuna değinilmiştir. Çalışmanın amacı ve önemi açıklanmıştır ve kaynak araştırmasına yer verilmiştir.

İkinci bölümde: Boolean fonksiyonları minimumlaştırma metotları özet şeklinde anlatılmıştır. Haritasal ve cebirsel yöntemler gösterilmiş ve bu yöntemlerin avantaj ve dezavantajlarına değinilmiştir.

Üçüncü bölümde: Geliştirilen algoritmada matematik araç olarak kullanılan küp cebri anlatılmıştır. Anahtarlama fonksiyonları için yerel basitleştirme algoritmaları için geliştirilen Yakın Minimal Örtme Algoritması (Near-Minimal Cover Algortihm). Geliştirilen Algoritma örneklerle açıklanmıştır. Matematik araç olarak kullanılan küp cebri işlemlerinin standart bilgisayar işlemleri üzerinden gerçekleştirilmesi gösterilmiştir. Algoritmanın daha iyi anlaşılması için birkaç örnek çözülmüştür.

Dördüncü bölümde: Karmaşıklık değerlendirilmesi yapılmıştır. Quine McCluskey Metodu ile Yakın Minimum Örtme Algoritması karmaşıklık yönünden karşılaştırılmıştır. Geliştirilen yöntemler ESPRESSO ile karşılaştırılmış ve sonuçları bu bölümde verilmiştir.

Beşinci bölümde: Bu tez çalışmasından elde edilen sonuçlara değinilerek bu konuda çalışmak isteyenler için bazı önerilerde bulunulmuştur.

Altıncı bölümde: Bu yüksek lisans çalışmasında yararlanılan kaynaklar verilmiştir.

Yedinci bölümde: Geliştirilen algoritmanın program kodları verilmiştir.

1.3 Materyal ve Metot

Mantık fonksiyonlarının ifade biçimleri, sadeleştirme yöntemleri, Algoritmaları ve programları kullanılacaktır. Bu yolda elde edilmiş son teorik sonuçlara dayanarak ve minterm yöntemiyle küp cebri yöntemleri bir arada kullanılarak daha etkin olan yeni bir yöntem meydana çıkarılacaktır.

Bir lojik fonksiyonun, birden fazla değişik ifadesi bulunabilir. Tüm olası ifadeler arasından minimum ifade bulunmaya çalışılır. Buradaki minimumluk en iyilik ölçütüne göre tanımlanabilir (Çırpan 1992). Bu en iyilik ölçülü;

a- En az sayıda lojik kapı gereksinimi

b- Çarpım Terimlerinin Toplamı (sum of product-SOP) biçiminde en az terim,

c- Toplam Terimlerinin Çarpımı (product of sum-POS) biçiminde en az terim,

d- Giriş ile çıkış arasındaki katman sayısının minimumlaştırılması ve dolayısıyla gecikme zamanını en aza indirebilmeyi sağlamak.

Çarpım terimlerinin toplamı biçimindeki bir fonksiyon, mantıksal değeri değiştirilmeden hiçbir teriminin çıkartılamayacağı biçimde ise, indirgenemezlik özelliğine sahiptir. Genelde indirgenemezlik ve minimumluk birbirlerini içermez ya da gerektirmezler. Sonuç olarak her minimum fonksiyon indirgenemezdir. Fakat her indirgenemez fonksiyon. minimum fonksiyon değildir (Çırpan 1992).

1.4 Kaynak Araştırması

Allahverdi N.M. ve Kahramanlı Ş.Ş. (1995), Küp cebri elemanları ve uygulama biçimlerini belirtmişlerdir. Küp cebri işlemlerini göstermişlerdir.

Beckert ve ark. (1997), çok seviyeli lojik devrelerin minimumlaştırılması için yeni yaklaşımlara değinilmiştir.

Çelikağ M. (1989), çeşitli minimumlaştırma Algoritmaları incelenmiştir. Bu algoritmalar birbirleri ile karşılaştırılmış ve değerlendirme yapılmıştır.

Çölkesen R. (2002), karmaşıklığın (complexity) tanımını belirtmiş ve çeşitli gösterimlerini sunmuştur.

Dagenais M.R. ve ark. (1986), çok çıkışlı fonksiyonların tam minimumlaştırılması için geliştirilen yeni prosedüre değinilmiştir.(McBOOLE prosedürü).

Dietmeyer D.L, (1979), küp cebrini anahtarlama fonksiyonlarının ilk terimlerini (local prime implicants) bulmak için kullanılmıştır. Daha sonra lojik fonksiyonların minimumlaştırılması üzerinde kullanılmıştır.

Fiser P. ve Hlavıcka J. (2003), Yeni bir İki seviyeli Boolean sadeleştirme algoritması geliştirilmiştir (BOOM Boolean Minimizer) .

Kahramanlı S.S. ve Allahverdi N.M. (1993), çok değişkenli l3oolean fonksiyonlar için yeni bir sadeleştirme algoritması sunulmuştur.

Karnaugh, M.(1953), Lojik devrelerin sentezi için harita metodunu sunmuştur. Haritanın oluşturulması ve sadeleştirme işlemi için haritanın nasıl kullanılacağı gösterilmiştir.

Mano M. M. (1984), lojik devreler ve lojik fonksiyonlar ile ilgili bilgiler vermiştir. Bir tablo metodu olan QMM metodu ve K-Haritaları anlatılmıştır. Fonksiyonları minimumlaştırırken elde edilen aralık sonuçlarının sayısını bulmak için gerekli olan formüller verilmiştir.

McCluskey, E.J.(1956), Boolean fonksiyonları sadeleştirmek için Quine tarafından başlatılan metodu geliştirmiş ve sunmuştur.

McGeer P.C. ve ark. (1986). Çok çıkışlı fonksiyonların tam sadeleştirilmesi için geliştirilen yeni prosedüre değinilmiştir (ESPRESSO-SIGNATURE). İşaret küpleri kullanılarak Asal implikantlar kümesi küçültülmüştür. Karmaşık problemlerde Espresso-II algoritmasına göre da iyi sonuçlar vermiştir.

Nadjafov E ve Kahramanlı S.S. (1973), küp cebrini anahtarlama fonksiyonlarına uyarlamışlardır. Daha sonra lojik fonksiyonların minimumlaştırılması üzerinde kullanılmıştır.

Perkins S.R. ve Rhyne T.(1988), Boolean fonksiyonlarının çoklu çıkışları için Asal İmplikantları belirleme ve seçme Algoritmalarını sunmuşlardır.

Sasao ve Butler 2001 ve Mishchenco ve Sasao 2003, minimumlaştırma problemlerinin bugünkü durumları hakkında açıklama yapmışlardır.

Tirumalai P.P.ve Butler J.T. (1991), son zamanlarda sunulan toplam terimlerin çarpımı şeklinde sadeleştirme yapan Algoritmaların birçoğu doğrudan örtme metodunu kullanmıştır. Bu makalede çeşitli doğrudan örtme metotları açıklanmıştır.

Uçar. (1996), lojik devre tasarımları için çeşitli algoritmaları incelemiş ve bu Algoritmalardan yeni bir yöntem geliştirmeye çalışmıştır.

2 MANTIK FONKSİYONLARINI SADELEŞTİRME METOTLARI

2.1 Fonksiyon Tanımları

Boole fonksiyonlarında, fonksiyonun değişken sayısına göre sahip olduğu çıkış durumları değişmektedir. n sayıda değişkene sahip olan fonksiyon 2^n sayıda mintermle ilişkide olur. Bu ilişkinin karakterine göre söz konusu mintermler aşağıdaki gibi çeşitli gruplara bölünür (Kahramanlı ve Özcan 2002)

- Doğru kümesi: Fonksiyonun değerinin 1' e eşit olduğu mintermler,
- Yanlış kümesi: Fonksiyonun değerinin 0' a eşit olduğu mintermler,
- Etkisiz Elemanlar Kümesi: Fonksiyonun değerinin belirsiz olduğu mintermler.

Bu gruplara uygun olarak F, R, D (belirsiz) kümeleri oluşturulur.

Tanım 2.1: Yalnız F ve R kümeleriyle ilişkili olan fonksiyonlara *Tam Belirlenmiş Fonksiyonlar* denir,

Tanım 2.2: F, R ve D ile ilişkili olan fonksiyonlara *Tam Belirlenmemiş Fonksiyonlar* denir. F, R ve D kümelerinin ölçüleri |F|, |R| ve |D| olarak gösterilirse, F, R ve D kümeleri ile onlara bağlı olan F fonksiyonu arasında aşağıdaki değer ilişkilerinin olduğu görülebilir (Kahramanlı ve Özcan2002).

• $|F| = 2^n$, Bu durumda mintermlerin tümünde fonksiyonun değeri 1 olduğu için aslında fonksiyon değil bir sabit (lojik 1) söz konusudur,

• $|R| = 2^n$, Bu durumda mintermlerin tümünde fonksiyonun değeri 0 olduğu için aslında fonksiyon değil bir sabit (lojik 0) söz konusudur,

• $|F| < 2^n$, $|R| < 2^n$, $|D| = 0$; $|F| + |R| = 2$. Bu durumda tam belirlenmiş olan bir fonksiyon söz konusudur,

• $|F| < 2^n$, $|R| < 2^n$, $|D| < 2^n$ ise $|F| + |R| + |D| = 2^n$. Bu durumda bu fonksiyona tam belirlenmemiş fonksiyon denir.

2.2 Karnaugh Haritası Metodu

Her fonksiyonun doğruluk tablosu gösterimi tektir; ancak, cebirsel olarak ifade edildiğinde değişik şekillerde verilebilir (Mano 2002). Boole fonksiyonları, cebirsel yollarla sadeleştirilebilirler. Fakat bu minimumlaştırma yönteminin, sistematik kuralları olmadığından kullanışlı değildir.

Harita metodunun özellikleri sadeleştirilmesine yarayan en basit ve görsel bir yöntemdir. Bu yöntem doğruluk tablosunun şekillendirilmiş bir biçimi veya Venn diyagramlarının gelişmiş bir şekli olarak da görülebilir. Karnaugh tarafından geliştirilen bu metot "Karnaugh Haritası - KH" adıyla bilinir. KH (Karnaugh Haritası) metodu en çok dört, beş değişkenli fonksiyonların sadeleştirilmesi için kullanılır ve temel olarak,

$$f = ax + a\bar{x} = a(x + \bar{x}) = a \qquad (2.1)$$

Kuralına dayanır. Değişken sayısı n olan bir fonksiyon için düzenlenen Karnaugh haritası 2^n tane hücreden oluşur. KH metodu, aslında bir fonksiyonun standart formda ifade edilebileceği tüm şekilleri sunan görsel bir yöntemdir. KH' de her bir hücreye karşılık gelen mintermlerin yazılması yerine, onun varlığını bildiren bir işaret konur. Hücreleri işaretleme yöntemine (Mano 2002, Kahramanlı ve Özcan 2002, Karnaugh 1953) kaynaklarında ayrıntılı bir şekilde yer verilmiştir.

2.2.1 KH Metodunun Kullanımı

Değişken sayısının dört veya beşi geçmediği durumlar için KH metoduyla minimumlaştırma uygun bir yöntem olabilir. Değişken sayısı arttıkça, çok sayıdaki hücre, uygun komşu hücre seçimini zorlaştırır. KH metoduyla minimumlaştırma kullanıcının belirli kalıpları görebilme yeteneğine dayandığından, aslında bir deneme yanılma yöntemidir. Bu durum, KH metodunun en belirgin dezavantajıdır. Ayrıca beş veya altı değişkenli fonksiyonlar için, en uygun seçimin yapılmış olduğundan emin olmak bir hayli zordur. Bu metodu, bilgisayar programlarına uyarlamak oldukça güçtür.

2.3 Cebirsel Sadeleştirme Yöntemleri

2.3.1 Tablo Yöntemi (Quine-McCluskey metodu)

Quine McCluskey Metodu (QMM), bir fonksiyonun minimum sayıda SOP şeklinde ifade edilmesini sağlar. Bu Algoritma iki aşamada gerçekleştirilir (Mano 2002, McCluskey 1956, Coudert 1994, Quine W.V.O. 1952):

a- Fonksiyon için bütün asal implikant (Prime Implicant-AI) ları bulmak,

b- Fonksiyonun bütün mintermlerini örtmek (cover) için gereken minimum sayıda asal implikantlar kümesini seçmek.

Bu aşamalar aşağıda açıklanmıştır.

2.3.1.1 Asal implikantların bulunması

Verilen Boole fonksiyonun AI' larının bulunması süreci, söz konusu fonksiyonun minterm listesinin düzenlenmesi ile başlanır. Mintermler, içerdikleri 1' lerin sayısına göre gruplara ayrılır. Bu gruplar, mintermlerin içerdikleri 1' lerin sayısına göre küçükten büyüğe doğru sıralanır. Bu yöntemle oluşturulabilecek maksimum grup sayısı (m) kombinasyon hesabı gereği ($\binom{n}{0},\binom{n}{1}\ldots\binom{n}{n-i},\binom{n}{n}$ (i=0,1…n)) değişkenlerin sayısından bir fazla olabilir (m=n+1). (Quine 1955).

(2.1) kuralı kullanılarak, *i.* grubun her bir mintermi ile (i+1). grubun her bir mintermi arasında yeni terimlerin elde edilip edilemeyeceğine bakılır. Eğer komşu grup mintermleri arasında sadece bir bitlik farklılık varsa, bu farklılık gösteren bit elde edilecek olan çarpım teriminde tire (-) işareti ile gösterilir. Gruplar arasındaki

karşılaştırma süreci (i-1) ve in çiftine kadar tekrarlanır. Çarpım terimlerinde k tane değişkeni eksik olan terimler yani k tane (-) işareti olanlar k-küp olarak adlandırılır. Bu tanıma göre mintermler 0-küp olarak adlandırılır (Mano 2002, McCluskey 1956, Çelikağ 1989).
0-küp sütunundaki bütün komşu grup mintermlerin karşılaştırılması la l -küp sütunu oluşturulur. Aynı işlemler 1-küp sütununa uygulanır ve buradan 2-küp sütunu oluşturulur. Aynı işlemler sütunlar arasında birleşme yapılamayacak duruma gelinceye kadar tekrarlanır. Bu k-küp sütunların sonunda işaretlenmemiş çarpım terimler, AI' lardır (Mano 2002, McCluskey 1956, McCluskey 1986, Çelikağ 1989).

2.3.1.2 Minimum Aİ kümesinin seçilmesi

Esas Asal İmplikantlar kümesinin bulunması: Minimum kümesi, minimum sayıda esas ve ikincil esas Aİ (essential and secondary essential prime implicant, EAI, İEAI) kümesinden oluşur. LA!' lar, Aİ' lardan seçilir. Eğer fonksiyonun bütün mintermleri, EAI' lar tarafından

örtülmüyorsa, İEAI' ların seçilmesi gerekmektedir. Burada örtülmek, fonksiyonu oluşturan bütün mintermlerin, minimum sayıda AI' lar tarafından kapsanması demektir. Üstünlük (dominance) ve denklik (equivalent) kuralları, AI' ların fazla olanlarını eleyerek, IEAI' ları bulmak için kullanılır (Mano 2002, McCluskey 1956, McCluskey 1986, Çelikağ 1989). AI' ların minimum kümesini bulmayı kolaylaştırmak için Asal İmplikantlar Tablosu (prime implicant table - AİT) kullanılır (Mano 2002, McCluskey 1956, McCluskey 1986, Çelikağ 1989). AIT' de, AI' lar satırlara, mintermler de sütunlara yerleştirilir. Fonksiyonun minimum şeklini oluşturacak Aİ' ları belirlemek için önce EAİ' lar seçilir. Eğer bir minterm sadece bir Aİ tarafından örtülüyorsa, bu Aİ, EAİ' dır ve SOP kümesine dahil edilir. Çünkü bu mintermi örtecek başka bir Aİ yoktur. Bütün EAİ' lar seçildikten sonra, bütün mintermler örtüldüyse minimum SOP kümesi oluşturulmuş demektir. Eğer hala bazı örtülmeyen mintermler varsa, bu mintermleri örtecek olan AI' ların diğer AI' lardan seçilmesi gerekir. Bu yolla seçilecek olan her bir AI, ikincil esas asal implikant (secondary essential prime implicant - İEAI) olarak adlandırılır.

İEAI kümesinin bulunması: İEAI' lar, sadeleştirilmiş AİT' dan seçilir. SAİT' da önceden seçilmiş EAI' lar ve örtülmüş mintermler bulunmaz (McCluskey 1956, Çelikağ 1989, Rudell 1989, Quine 1955).

Baskın satır kuralı (row dominance):

Tanım 1. AH" da bulunan herhangi bir i ve j satırları için, ! satırında bulunan x" işaretlerinin tümü *i* satırında da bulunuyorsa, bu iki satır birbirine eşittir. Tanım 2. AİT' da bulunan *i* ve j satırları için,] satırında bulunan bütün "x" işaretleri *i* satırında da varsa ve *i* satırında en az bir tane fazla "x" işareti varsa, *i* satırı] satırını kapsar denir.
Tanım 3. AI' nın maliyeti, çarpım terimindeki literalı sayısı ile belirlenir. Çarpım teriminde daha fazla literali olan daha fazla maliyete sahiptir.
Yukarıdaki tanıma göre *i* satırı kapsayan satır, *j* satırı kapsanan satırdır. Kapsanan satır SAİT' den çıkarılabilir. Eğer iki satır birbirine eşitse bu satırlardan maliyeti fazla olan satır çıkarılır (Başçiftçi ve ark. 2003).

Baskın sütun kuralı (column dominance):

Tanım 4. AIT' da bulunan i ve j sütunları için, i sütununda bulunan "x" işaretleri j sütununda da bulunuyorsa, bu iki sütun birbirine eşittir. (eşit sütunlar)

Tanım 5. AIT' da bulunan i ve j sütunları için, i sütununda bulunan bütün "x" işaretleri j sütununda da varsa ve i sütununda en az bir tane fazla "x" işareti varsa, i sütunu j sütununu kapsar denir.

Yukarıdaki tanıma göre i sütunu kapsayan sütun, j sütunu kapsanan sütundur. Kapsanan sütun SAİT' den çıkarılır. Eğer iki sütun birbirine eşitse bu sütunlardan maliyeti fazla olan sütun çıkarılır (Başçiftçi ve ark. 2003). Bütün sadeleştirme kuralları uyguladıktan sonra AİT' de birden fazla minterm kalabilir. Bu tür tablolara periyodik tablo denir. Periyodik problemler Dallandırma Metoduyla veya Petrik metodu çözülebilir (Çelikağ 1989, Rudell 1989).

2.3.1.3 QMM kullanım alanları

KH metodunda, beş veya altı değişkenli fonksiyonlar için, en uygun seçimin yapılmış olduğundan emin olmak ve bu metodu, bilgisayar programlarına uyarlamak da bir hayli zordur. Bu zorluklara QMM çözüm getirir. Bu metot, adım adım uygulanarak fonksiyon için minimumlaştırılmış ifadeyi standart bir biçimde elde eder. Bu metot, çok değişkenli fonksiyonlara uygulanabilir ve bilgisayarda programlamaya uygundur. Ancak, rutin ve monoton işlemlerinden dolayı kullanımı oldukça sıkıcıdır ve hata yapma olasılığı yüksektir. QMM çok girişli - çok çıkışlı fonksiyonlar için genişletilebilir. Pratik uygulamalarda, çok çıkışlı problemlerde AI' ların sayısı çok fazladır. Bundan dolayı, bu metot çok fazla hafızaya gereksinim duyar (Chai 2000). Giriş değişkeni sayısı fazla olursa, minterm sayısı fazla olacağından, üretilen AI fazla olacaktır ve bu AI' ların depolanması için çok fazla hafızaya ihtiyaç duyacaktır (Chai 2000). Bundan dolayı bu metot, çok değişkenli problemler için uygun değildir. Bununla birlikte, giriş değişkeni sayısı az olursa diğer metotlara göre daha hızlı olabilir.

2.3.2 Petrick Metodu

Bir minterm sütununda *L* tane AI varsa bu mintermi örtmek için *L* tane farklı AI var demektir. Bu mintermi örtmek için olası AI' ların toplamı *L* tanedir. Periyodik tablodaki *N* tane sütun (veya minterm) *N* tane toplam terimi üretir. AI fonksiyonu veya p-fonksiyon *N* tane toplam terimlerinin çarpımı (POS) şeklinde tarif edilir. Her sütun için AI' lar toplanır ve diğer sütunların AI' larının lojik toplamı ile lojik çarpılır. Çarpımlar sonra toplam olarak düzenlenir. Düzenleme yapıldıktan sonra en az literale sahip olan bileşen veya bileşenler minimum ifadeyi oluşturur (Çelikağ 1989). Minimum ifade tek olabileceği gibi birden fazla da olabilir.

Aşağıda bir örnek verilerek Petrick metodu açıklanmıştır.

1. Adım: tablodaki bütün satırları numaralandır.

   $P_1, P_2 \ldots P_m$

2. Adım: sütunlardaki her X için P değerlerini seç.

			0	1	2	5	6	7
P_1	(0,1)	a'b'	X	X				
P_2	(0,2)	a'c	X		X			
P_3	(1,5)	b'c		X		X		
P_4	(2,6)	bc'			X		X	
P_5	(5,7)	ac				X		X
P_6	(6,7)	ab					X	X

Birinci sütundaki X içeren P değerlerini seçelim (P_1 ve P_2)

$(P_1 + P_2)$

Bu şekilde devam ederek şu sonuca varırız:

$P = (P_1 + P_2)\,(P_1 + P_3)\,(P_2 + P_4)\,(P_3 + P_5)\,(P_4 + P_6)\,(P_5 + P_6)$

$P = (P_1 + P_2)\,(P_1 + P_3)\,(P_4 + P_2)\,(P_5 + P_3)\,(P_4 + P_6)\,(P_5 + P_6)$

$P = (P_1 + P_2)\,(P_1 + P_3)\,(P_4 + P_2)\,(P_4 + P_6)\,(P_5 + P_3)\,(P_5 + P_6)$

$P = (P_1 + P_2 P_3)\,(P_4 + P_2 P_6)\,(P_5 + P_3 P_6)$

$P = (P_1 P_4 + P_1 P_2 P_6 + P_2 P_3 P_4 + P_2 P_3 P_6)\,(P_5 + P_3 P_6)$

$P_{Sdoğru} = P_1 P_3 P_4 P_6 + \boldsymbol{P_1 P_4 P_5} + P_1 + P_2 P_3 P_6 + P_1 P_2 P_5 P_6 + P_2 P_3 P_4 P_6 + P_2 P_3 P_4 P_5 + \boldsymbol{P_2 P_3 P_6} + P_2 P_3 P_5 P_6$

Bu ifadeler bize sadeleşmiş ifadelerin hepsini ifade eder. Yani P_1, P_2 ve P_3 ye çizgi çekilerek sadeleştirme yapılır. Eğer her bir terim eşit maliyete sahip olduğu kabul edilirse (burada eşit), bu fonksiyonda en sade ifade nedir?

İki en sade durum var $P_1 P_4 P_5$ and $P_2 P_3 P_6$:

F = a'b' + bc' + ac ve F = a'c' + b'c + ab

2.3.3 ESPRESSO-II Algoritması

ESPRESSO-II, fonksiyonunun doğru kümesini, belirsizler kümesini ve yanlış kümesini giriş olarak alır (Brayton ve ark. 1984). Bu Algoritma çıkış olarak sadeleştirilmiş bir örtü verir. ESPRESSO-II minimuma yakın çözümü bulmaktadır ve aşağıda verilen 3 sayıyı azaltmaya çalışmaktadır (Brayton ve ark. 1984, McGeer ve ark. 1986, Brayton ve ark. 1993, McGeer ve ark. 1993).

1. NPT: örtüdeki çarpım terimlerinin sayısı.
2. NLI: örtünün giriş kısmındaki terimlerinin sayısı.
3. NLO: örtünün çıkış kısmındaki terimlerinin sayısı.

ESPRESSO-II f (NPT, NLI, NLO) vektörünü kullanarak sadeleştirme süresince (F)' nin bileşenlerini azaltmaya çalışmaktadır (Brayton ve ark. 1984, McGeer ve ark. 1986). Bu işleme, son döngü sırasında, bileşenlerin hiçbirisi değişmediğinde son verilir (Brayton ve ark.

1993, McGeer ve ark. 1993). Sadeleştirme işlemine geçmeden önce sadeleştirilecek olan fonksiyonlara UNWRAP (dağıtma, açma) prosedürü uygulanır. Bu prosedür k tane fonksiyon tarafından paylaşılan bir küpü, her biri sadece bir fonksiyon tarafından paylaşılan k tane küp ile yer değiştirir (Brayton ve ark. 1984, Uçar 1996). Her ne kadar bu şekilde optimaldan daha uzaklaşılsa da böyle bir işlem sonucunda sadeleştirme işlemi girişe daha az bağımlı olur ve EXPAND prosedüründe küplerin daha yararlı bir şekilde hangi fonksiyon tarafından paylaşılacağı bulunabilir (Brayton ve ark. 1984). Bu şekilde F (doğru kümesi), *D* (belirsizler kümesi) ve *R* (yanlış kümesi) örtüleri elde edildikten sonra P vektörü hesaplanır. Bu vektörün bileşenlerinde bir azalma görülemeyinceye kadar genişletme (expand), tekrarsız örtü (irredundant_cover) ve daraltma (reduce) prosedürleri çalıştırılır. (1)' nin bileşenlerinde azalma görülmediğinde LAST_GASP prosedürü çağrılır. Eğer (I' nin bileşenlerinde azalma görülürse tekrar daraltma prosedürü çağrılır. ESPRESSO-II sadeleştirme Algoritması altı tane temel prosedürden oluşur. Bunlar COMPLEMENT, EXPAND, ESSENTIAL_PRIMES, IRREDUNDANT_COVER, REDUCE, LAST_GASP' dır. Bunlara ek olarak yukarıdaki altı algoritmanın pek çoğu önemli bir şekilde TAUTOLOGY algoritmasına dayanır. Bu algoritma ile elemanları küpler olan bir kümenin, bir küpü örtüp örtmediği belirlenir (Brayton ve ark. 1984).

COMPLEMENT Prosedürü: Bu kısımda birden çok fonksiyon için tümleyen alma yöntemi verilmiştir. Bu yöntemde monoton fonksiyonun özelliklerinden yararlanılarak kendisini çağıran (recursive) bir prosedür ile bir fonksiyonun tümleyeni bulunur ve bu işlem her çıkış için tekrarlanır. EXPAND prosedürü, ESPRESSO-II içinde tümleyen alma prosedürünü kullanan tek ana prosedürdür. Teker teker fonksiyonların tümleyenlerini alma işlemi, bazı çarpım terimlerini tekrar kullanacağından daha fazla bellek kullanır. Complement prosedürü, verilen F ve D örtüleri için *R* örtüsünü hesaplar. Bu prosedür EXPAND prosedüründe asal bileşen seçiminde kullanılır (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

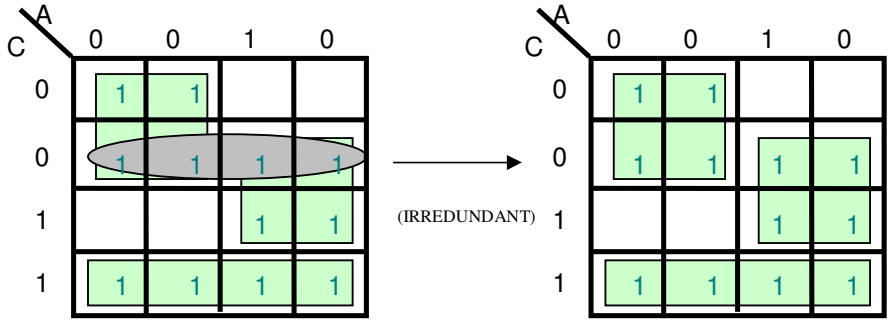
Tanım 1: Bir *f* fonksiyonunun x, değişkeninin değeri 0' dan 1' e değişmesi ile çıkışı da 0 iken 1 (1 iken 0) oluyorsa, fonksiyonu x değişkenine göre monoton artandır (azalandır) denir.

Tanım 2: Bir fonksiyon bütün değişkenlerine göre monoton artan veya azalan ise bu fonksiyona monoton fonksiyon denir. (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

EXPAND Prosedürü: Genişletme işleminin amacı *F* örtüsünden mümkün olduğu kadar çok sayıda küpün atılmasıdır. Bunun için F örtüsünün küpleri teker teker belirli bir sıra ile ele alınır ve ele alınan küp ile *F* örtüsünde bulunan maksimum sayıda küp örtülmeye çalışılır. Daha sonra genişletme işlemi ile elde edilen asal küpler örtüye dahil edilir. Bu küplerin E örtüsündeki kapsadıkları küpler örtüden çıkarılır. EXPAND Algoritmasının sonucu genişletilen küplerin ele alınma sırasına bağlıdır (Brayton ve ark. 1984, McGeer ve ark. 1986, Uçar 1996).

ESSENTIAL_PRIMES Prosedürü: Burada çözülmesi gereken problem, verilen *F* örtüsü için her bir c', f nin bir asal küpü olmak üzere, verilen bir e' asal küpü ‚/' nin bir temel asal bileşeni olup olmadığının belirlenmesidir. Temel asal bileşenler /' nin bütün asal örtülerinde bulunmalıdır. Bu nedenle EXPAND, REDUCE ve RREDUNDANT_COVER prosedürleri yürütülürken temel asal bileşenleri örtüden demek, hesaplama zamanını azaltır (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

IRREDUNDANT_COVER Prosedürü: ESPRESSO-II' nin EXPAND prosedürünün uygulaması ile F asal örtüsü elde edilir. Bu örtüde hiçbir küp diğerini kapsamaz. Bununla birlikte F' nin minimal örtü olduğu kesin değildir. IRREDUNDANT_COVER prosedürü verilen F ve D için, F' nin bazı küplerinden oluşan minimale yakın F2 örtüsünü belirler. Bu prosedür ile F2 F olan ve mümkün olduğu kadar az küpe sahip E2 örtüsü elde edilmeye çalışılır. Bu prosedürden sonra bir minimal örtü elde edilir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).



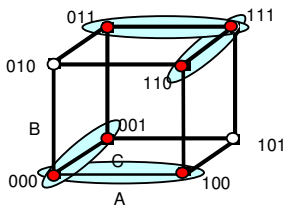
Şekil 1: RREDUNDANT_COVER prosedürü

REDUCE Prosedürü: IRREDUNDANT_COVER prosedürü ile elde edilen örtüdeki küpleri teker teker ele alır. 1-ler e EF küpü için, c küpünün (F) D örtüsü tarafından kapsanmayan mintermlerden oluşan en küçük küp c' yi bulur. Daha sonra E örtüsünde e küpü ile c küpünü değiştirir. Yani F(F-R) e olur. Bu şekilde elde edilen önü EXPAND prosedürü ile daha çok yönde genişletilebilir. Ayrıca F örtüsü bu işlemle daha küçük küplerden oluşur ve genişletilen küpler tarafından kapsanma olasılığı artar. Bu prosedürün sonucu küplerin ele alınma sırasına bağlıdır (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

LAST_GASP prosedürü: Bu algoritma sadeleştirilecek olan örtüden birkaç küp daha çıkarabilmek için kullanılır. LAST_GASP, değiştirilmiş bir REDUCE ve değiştirilmiş EXPAND prosedürlerini içerir. En son sadeleştirilmeye çalışılan küpler en az sadeleştirme şansına sahiptir. Bunun sebebi daha önce sadeleştirilerek kısaltılan küpler nedeniyle örtü zaten az sayıda minterm içermektedir. Sadeleştirilecek küplerin kabaca seçimi, EXPAND işlemi sonunda örtüdeki küp sayısının azalacağını garanti etmez. LAST GASP prosedüründe her bir küp maksimum şekilde sadeleştirilir. Daha sonra sadeleştirilen küpler üzerinde EXPAND işlemi uygulanır. REDUCE prosedürü aynı işlemi yapmaktadır fakat bu prosedürde küpler belirli bir sıra ile ele alınmaz. Her küp bağımsız olarak ele alınarak REDUCE prosedürü ile yapılan işlem tekrarlanır (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

TAUTOLOGY Prosedürü: Bir fonksiyonun sabit–1 olup olmadığının belirlenmesi için ESPRESSO-II tarafından kullanılan temel bir işlemdir. Bu işlem IRREDLNDANTCOVER, REDUCE, ESSENTİAL PRİMES ve LAST GASP prosedürlerinin temel bölümünü oluşturur. Bu nedenle etkili bir TAUTOLOGY Algoritması ESPRESSO-II' nin hızı için önemlidir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

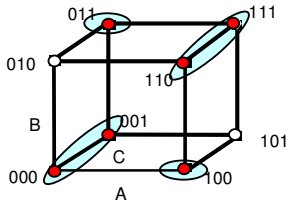
Örnek: Şekilde verilen mantıksal fonksiyonun Espresso algoritması ile çözümünün bulunması



Espresso algoritması tarafından ilk bulunan örtü bu şekilde verilmiştir.

2.3.3.1 Daraltma işlemi (reduce)

Bir implikanta bir literal (değişken) ekleyerek kapsama alanını azaltma işlemidir.

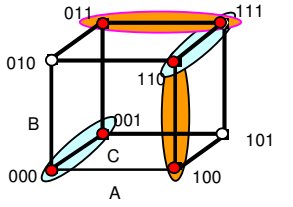


A ile

X $\overline{C}$X implikantı B literali eklenerek yapılan azaltma işlemi AB $\overline{C}$X implikantı oluşur.

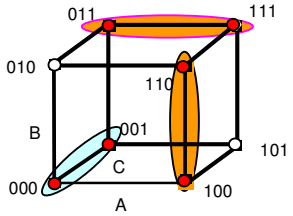
2.3.3.2 Genişletme işlemi (expand)

Bir implikanttan bir literalin (değişken) çıkartılması işlemi ile implikantın kapsama alanını genişletme işlemidir.



$\overline{A}$

BC implikantından $\overline{A}$ literalini çıkartırsak oluşan imlikant XBC olur. Genişletme işleminde hangi literalin çıkarılırsa daha iyi olacağını seçmek için kofaktor kriteri kullanılır



Tekrarsız implikant eleme işlemi

2.3.3.3 Kofaktör

Tanım: bir C implikantının x_j literaline bağlı olarak oluşturulan kofaktor Kx_j olsun Kx_j kofaktoru şu şekilde bulunur:

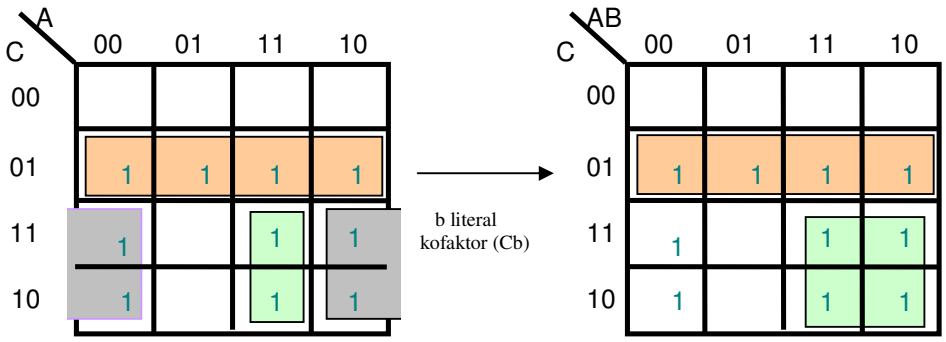
- x_j veya $\overline{x}_j$ değeri C implikantında yoksa sonuç C dir.
- $C\backslash\{x_j\}$ olur eğer x_j literali C implikantında varsa
- C implikantı $\overline{x}_j$ literalini içeriyorsa sonuç $\varnothing$ olur eğer

Örnek: F fonksiyonu için b literalinin kofaktor olduğu zaman oluşan sonucu bulalım:

$F = abc + \overline{b}c + \overline{c}d$

$F_b = ac + \varnothing + \overline{c}d$

$F_b = ac + \overline{c}d$



2.3.3.4 Espresso algoritması

1- Hangi küp diğer küp/küpler tarafından kapsanıyorsa onu kümeden çıkar. (REDUCE)
2- Artık küpler Küp' ten bir değişken çıkartmakla ortaya çıkar.
3- İmplikantları genişlet (EXPAND)
   - Genişletilmiş implikantların kapsadığı (örttüğü) diğer implikantları kümeden çıkar
   - Sonucun iyi olması genişletme işleminin sırasına bağlıdır.
   - Heruistic metotlar en iyi genişletme sırasını bulmaya çalışır.
   - 1, 2, 3 işlemlerini sürekli yaparak alternatif prime imlikantları bul ve fonksiyonun maliyeti düştüğü müddetçe 1, 2, 3 işlemlerini yapmaya devam edilir.

```
Espresso(F,D) //  F Doğru kümesi D don't Care ve R Yanlış kümesi
    {
      R=TERS(F+D);              //Yanlış kümesini bul
      F=Genişlet(F,R);          //F kümesini genişlet
      F=Tekrarsız(F,D);         //Başlangıç tekrarsız örtü bulunur
      E=Temel (F,D);            // Temel birincil implikantlar bulunur
      F=F-E;                    //Bulunan elemanları kümeden çıkar
      D=D+E;
      F' in maliyeti düşüyorken  (while)
      {     R=Daralt(F,D);
            F= Genişlet(F,R);         //F kümesini genişlet
            F= Tekrarsız(F,D);        //Başlangıç tekrarsız örtü bulunur
      }
      F=F+E;
      Sonuç=F;                  // sonuç olarak F kümesini gönder.
    }
```

2.3.3.5 ESPRESSO-II Programı

ESPRESSO verilen fonksiyonu çarpım terimlerinin toplamı (SOP) şeklinde sadeleştiren, çok seçeneği olan bir programdır. ESPRESSO programının kullanım formatı aşağıdaki gibidir:

- Espresso [seçenekler] [dosya] [> çıktı dosyası]
  ESPRESSO programının kullandığı dosya formatı aşağıda gösterilmiştir. Programın tanıdığı anahtar kelimeler belirtilmiştir. [d] ondalık bir sayıyı belirtir. [s] bir string ifadeyi belirtir.

Verilen bu seçenekler her dosya da olması gereken durumlardır.
ESPRESSO programında kullanılan seçeneklerden çok kullanılanlar aşağıda açıklanmıştır.

—Dexact: Exact minimumlaştırma Algoritması (çarpım terimlerinin minimum sayıda olmasını garanti eder ve buluşsal (heuristic) olarak literallerin sayısını minimumlaştırır). Genellikle pahalı olabilecek sonuçlar üretir.

—Dsignature: Küp tabanlı kesin (exact) minimumlaştırma Algoritması (çarpım terimlerinin minimum sayıda olmasını garanti eder ve buluşsal olarak literallerin sayısını minimumlaştırır). Dexact seçeneğine göre daha hızlıdır ve Dexact seçeneğinin takıldığı problemleri çözer (Brayton ve ark. 1993, McGeer ve ark. 1993).

—Dso: 1-ler fonksiyonu tek çıkışlı fonksiyon gibi minimumlaştırır. Terimler fonksiyonlar arasında paylaştırılmaz.

2.3.3.6 Espresso dosya formatı

Espresso algoritması için kabul edilmiş dosya formatı şu şekildedir:

- .i [d] Giriş değişkeninin sayısını belirtir
- .o [d] Çıkış değişkeninin sayısını belirtir.

.e Dosyanın bittiğini gösterir.

F(A,B,C,D)= (4, 5, 6, 8, 9, 10, 13) D(0, 7, 15) fonksiyonu için oluşturulan giriş dosyası,

Giriş		Anlamı:
.i 4		Girişler
.o 1		Çıkışlar
.lb a b c d		Giriş Değişkenleri
.ob f		Çıkış Değişkenleri
.p 10		Ürün Sayısı
0 0 0 0	-	
0 1 0 0	1	
0 1 0 1	1	
0 1 1 0	1	
0 1 1 1	-	
1 0 0 0	1	
1 0 0 1	1	
1 0 1 0	1	
1 1 0 1	1	
1 1 1 1	-	
.e		

Espresso algoritması ile F(A,B,C,D)= (4, 5, 6, 8, 9, 10, 13) D(0, 7, 15) kabul edilmiş dosya formatı şu şekildedir:

F(A,B,C,D)= (4, 5, 6, 8, 9, 10, 13) D(0, 7, 15) fonksiyonu için oluşturulan çıkış dosyası,

Çıkış		Anlamı:
.i 4		Girişler
.o 1		Çıkışlar
.lb a b c d		Giriş Değişkenleri
.ob f		Çıkış Değişkenleri
.p 3		Ürün Sayısı
1 - 0 1	1	
1 0 - 0	1	
0 1 - -	1	
.e		

F(A,B,C,D)=A $\overline{C}$ D+A $\overline{B}$ C + $\overline{A}$ B

3 YAKIN MİNİMALİ ÖRTME ALGORİTMASI

Boole ifadelerinin sadeleştirmesi, mantık devrelerinin ve bilgisayar programlarının daha etkili olmasına yol açmaktadır. Minimumlaştırma ifadeleri önemlidir. Çünkü elektrik devreleri, verilen Boole ifadelerinin her bir terim veya literallerinin uygulanması için bireysel bileşenler içerir. Bu tasarımcıların daha az bileşen kullanmasını ve böylece de belirli sistemlerin maliyetlerinin düşmesini sağlamış olur. Tek çıkışlı veya çok çıkışlı Boolean minimumlaştırma teknikleri (Mano 1984) anlatılmıştır. Bu tekniklerin birçoğu iki adımda çalışır. İlk adımda bütün asal implikantları (prime implicant-Aİ) belirler ve ikinci adımda da verilen Boole ifadesini örtecek (kapsayacak) AI' ların altkümesini seçer (Perkins ve Rhyne 1988).

Bütün AI' ların belirlenmesi sürecinde son sonucun tam olarak belirlenmesi için ayrı durumlarda hesaplama yapılabilir. Özellikle, eğer her bir asal implikant tam olarak k tane 0, k tane 1 ve k tane belirsiz terim içeriyorsa, AI' nın tamamlanmış kümesinin gücü M=(3)!/(k!)3 dür (Kahramanlı ve Başçiftçi 2004). Örneğin k=l,2,3,4 için sırasıyla M=6, 90, 1680 ve 34650 dır. n değişkenli bir fonksiyon için AI' ların sayısı 3fl/ kadar büyük olabilir (Kahramanlı ve Başçiftçi 2004). Sonuç olarak, AI belirleme adımı değişken sayısı n arttıkça elverişsiz bir duruma gelebilir (Perkins ve Rhyne 1988). Açıkça görülmektedir ki ister iki seviyeli veya isterse çok seviyeli Boole ifadelerini sadeleştirme prosedürlerinin hepsi tüm durumlarda 0(2v) karmaşıklığına sahiptir (Allahverdi ve ark. 2000, Kahramanlı ve Başçiftçi 2003, Kahramanlı ve ark. 2005). Burada, tam belirlenmiş Boole fonksiyonunun ON mintermlerini örten AI' ların yerel belirlenmesinin metodu önerilmiştir. n değişkenli Boole ifadelerinin bu tür mintermleri maksimum n tek boyutlu küplere dahil edilebilir. Geçici sonuç küpleri kümesinin gücü n değerini geçmeyebilir (Allahverdi ve ark. 2000). Böylece, AI' ların minimum kümesini bulmak için $O(2^n)$karmaşıklığı yerine O(n) karmaşıklığı metodu kullanılabilir (Kahramanlı ve Başçiftçi 2003, Kahramanlı ve ark. 2005).

Bu çalışmada, 0ff küme tabanlı doğrudan örtme minimumlaştırma metodu (direct cover Minimization Method) tek çıkışlı fonksiyonlar için çarpım terimlerinin toplamı formunda sunulmuştur. Var olan doğrudan örtme metotlarında verilen On- küpü içeren yeterli asal implikantlar kümesini bulmak için, bu küp her defasında bir koordinat için genişletilir. Her genişlemenin doğruluğu, k < 2 Off-küplerin hepsi ile genişletilen küp kesiştirilerek kontrol edilir. Bir küpün genişlemesinin polinominal karmaşıklığa sahip olduğu dikkate

alındığında, bu yaklaşımın toplam karmaşıklığı O(n)O(2 ^{n}) şeklinde olmaktadır. Bu polinominal ve üssel karmaşıklığın çarpımıdır. Verilen On-küpü içeren asal implikantların tam kümesini elde etmek için önerilen metot, bu On-küp tarafından genişletilen 0ff-küpleri kullanır. Bu işlemin karmaşıklığı, yaklaşık olarak bir koordinat için bir On-küpün genişletilme karmaşıklığına eşdeğerdir. Bundan dolayı, verilen On-küpü içeren asal implikantların tam kümesinin hesaplama işleminin karmaşıklığı yaklaşık olarak O(n) kadar azaltılmış olur. Pratik olarak bu yaklaşım bir defada işlenecek olan asal implikant sayısını yüzlerce ve binlerce defa azaltmaktadır. Bu ise halen problem olan bellek kapasitesi darboğazını kolaylıkla aşma imkânı sağlamaktadır.

YMÖA çeşitli problemler üzerinde test edilmiş ve standart MCNC benchmarkları kullanılarak ESPRESSO ile karşılaştırılmıştır. Bu karşılaştırmalar sonucunda geliştirilmiş olan yöntemlerin ESPRESSO' ya göre önemli bir ölçüde hızlı olduğu ve az bellek kapasitesi gerektirdiği görülmüştür. Ayrıca sadeleştirme işlemleri sonucunda karşılaştırılan Algoritmaya göre çarpım terimlerinin toplamı şeklinde daha iyi sonuç buldukları belirlenmiştir.

3.1 İşaretlerin Gösterimi

n girişli ve m çıkışlı bir çoklu çıkışa sahip Boole fonksiyonu aşağıdaki gibi tanımlanır (Kahramanlı ve Başçiftçi 2003):

Giriş: B{0,l },
Çıkış: Y{0,l,d},
Fonksiyon f: $B^n \rightarrow Y^m$

Burada, çıkışta gösterilen d değeri (belirsiz terim) tam belirlenmemiş değer manasındadır ve fonksiyonun istenildiği yerinde 0 veya 1 olarak kabul edilebilir. Böyle bir fonksiyon AI' ların listesiyle temsil edilebilir. Her bir AI giriş ve çıkış kısımlarını içerir (Kahramanlı ve Başçiftçi 2003, Kahramanlı ve ark. 2005).

Giriş kısmı: n sabitler {0,l,x} olabilir;

Çıkış kısmı: m sabitler {0,l,d} olabilir.
Giriş kısmı küpe uygulanacak giriş uzayını belirler. Giriş kısmındaki x değeri bu değişken için 0 veya 1 değeri olabilir.

Bu tezde, tek çıkışlı Boole fonksiyonları için yeni bir sadeleştirme metodu geliştirilmiştir. Tek çıkışlı Boole fonksiyonu aşağıdaki gibi tanımlanır;

Giriş: B={0, l},
Çıkış: Y={0, l,d},
Fonksiyon f: B → Y.

SON: Fonksiyonun değerini 1 yapan ON mintermlerinin kümesi,

SOFF: Fonksiyonun değerini 0 yapan OFF mintermlerinin kümesi,

SDC: Belirsiz terim mintermlerinin kümesi.

Bu tezde sunulan Algoritmada SON kümesi ve SOFF kümesi tamamen kullanılmıştır. SDC kümesi ise kullanılmamıştır.

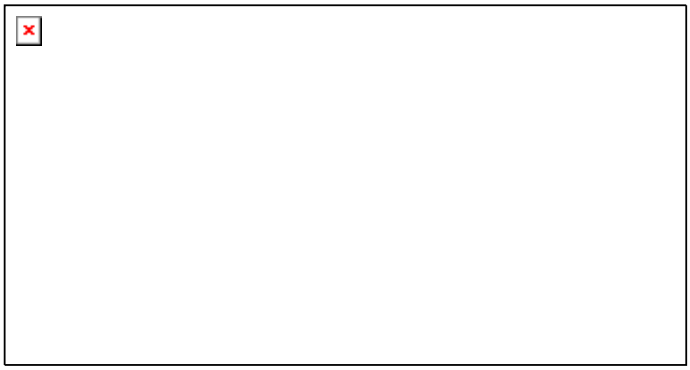
3.2 YMÖA kullanılan Küp Cebri' nin Elemanları ve Uygulama biçimleri

Lojik cebirdeki minimum terimler, küp cebrinin temelini oluşturmaktadır. Ancak küp cebrinde değişken sayısı en az üçtür. Üç değişken bir küpü tanımlamaktadır. Küp cebri ile geometrik olarak; bir minterm ile bir nokta, iki nokta ile bir hat, dört hattın birleşmesi ile bir yüz, altı yüzün birleşmesi ile bir küp tanımlanır (Nadjafov ve Kahramanov 1973, Güneş 2000). Bu küpün her bir koordinatı, 3 değişkenli bir Boole fonksiyonunun bir değişkenidir.

Küp cehri işlemleri, önce anahtarlama fonksiyonlarının (Switching Functions SFs) en son durumunu bulmak için geliştirilmiş ve uygulanmıştır (Roth *1956,* Nadjafov ve Kahramanov 1973). Yine bu işlem SF' nin ilk terimlerini (local prime implicants) bulmak içinde kullanılmıştır. Daha sonra lojik fonksiyonların minimumlaştırılması üzerinde kullanılmıştır (Nadjafov ve Kahramanov 1973).

3.2.1 Küp Cebri Elemanları Ve Uygulama Biçimi

n-boyutlu bir küpün her bir tepe noktası ikili kodlarla belirtilir. Bu küp koordinatlarına sahiptir. doğal olarak k1 koordinatı (0,1)' lerle belirtilir ve i1,2,...,n' dir. Bu yüzden aynı zamanda, belirli bir tepe noktasının kodu, bu tepe noktasının cebirsel ifadesini gösterir. Tepe noktalarına komşu olan diğer tepe noktaları da n bitlik kodlarla belirlenir. n bitlik kodlar, birbirinden sadece 1 hitlik farka sahipse bunlar komşu olarak adlandırılır. Örneğin 0110 kodu ile 0100 kodu komşudur (Nadjafov ve Kahramanov 1973, Allahverdi 1999, Güneş 2000).



Şekil 3.1. Üç boyutlu kodlanmış küp

Küpün elemanları; tepe, doğru, yüz, küp, hiperküp şeklinde adlandırılır. Bu elemanlar üzerinde bir işlem yaparken gelebilecek belirsiz durumların oluşmaması için, bütün koordinatları ve bu koordinatların arasındaki doğruların kullanılması gerekir. Bu amaç için, {0,1} kümesine ait olmayan geçersiz koordinatların konumları " * " ile belirtilmiştir.

3.2.2 Küp Cebrinin İşlemleri

Küp cebri, lojik 0 ve lojik 1 ile yapılan bütün işlemlerin dışında dört işlemi daha içermektedir. Bunlar;

a) Koordinatlı çarpma (star product, ☆-operation),

b) Koordinatlı çıkarma (sharp product #-operation),

e) Koordinatlı kesişme (∩-operation)

d) Dönüşümlü yutma işlemi (commutative absorption operation Δ- operation) işlemleridir (Allahverdi ve Kahramanlı 1995,Güneş 2000).

3.2.2.1 Koordinatlı çarpma işlemi (☆- işlemi)

Koordinatlı çarpma işlemi, aynı boyuta sahip iki küp arasında uygulanır. Fakat çarpımı yapılacak olan alt küpler, aynı boyutta olmak şartı ile değişik değerde olabilirler. Koordinatlı çarpma işlemi iki aşamada gerçekleştirilir. İlk aşamada bir v bileşeninin belirlenmesi için çarpım vektörü (vector of product-ÇV) oluşturulur.

İkinci aşamada, oluşturulan ÇV' nin koordinat değerlerine göre A ve B küpleri koordinatlı çarpıma tabii tutulur (Allahverdi ve Kahramanlı 1995, Allahverdi 1999).
A ve B küpleri aynı boyuta sahip iki küp olsun

$A= a_1, a_2 \ldots a_n$

$B=b_1, b_2 \ldots b_n$

Bu iki küp arasında koordinatlı çarpma işlemi uygulansın. İlk aşamada v bileşeninin belirlenmesi için aşağıdaki işlemler sonucunda çarpım vektörü ÇV oluşturulur;

$ÇV = A ☆ B = v_1, v_2 \ldots v_n$

Olmak üzere, v_i bileşeninde $i \in \{0,1\ldots n\}$ dir. v_i bileşeni;

- Eğer $a_i = b_i$ ise $v_i = a_i$ b olur.
- Eğer $a_i = *$ ve $b_i \neq *$ ise $y_i = b_i$ olur.
- Eğer $a_i \neq *$ ve b = * ise $v_i = a$ olur.
- Eğer a, b$\in \{0,1\}$ ve $a_i \neq b_i$ ise v_i =y olur.

ÇV' nin koordinat değerlerine göre A ve B küplerinin koordinatlı çarpımının sonuçları aşağıdaki gibi olmaktadır;

a. Eğer herhangi bir v_i = y bulunmazsa, A ve B' nin çarpımı sonucu, A ve B' nin alt küpü olan ÇV olmaktadır (yani A ☆B =ÇV).

b. Eğer sadece bir tane i değeri için v_i =y bulunuyorsa ve diğer değerler için $v_i = b_i = a_i$ ise (burada j$\in$ {l, 2... i-l, i+l... n} dir) A ve B küplerinin koordinatlı çarpımı sonucu, ÇV' de v_i yerine * sembolü konularak bulunan bir C küpüdür.

c. Eğer sadece bir tane v_i = y oluşuyor ve a_k =* veya b_k= * için $y_k \neq$ * ise A ve B küplerinin çarpımları sonucu, ÇV' de v_i yerine * sembolü konularak bulunan bir C küpüdür. C küpünün bir bölümü A tarafından, diğer bölümü B tarafından örtülür. Bu durum C küpünün A ve B küpleri ile ilişkisi olduğunu gösterir.

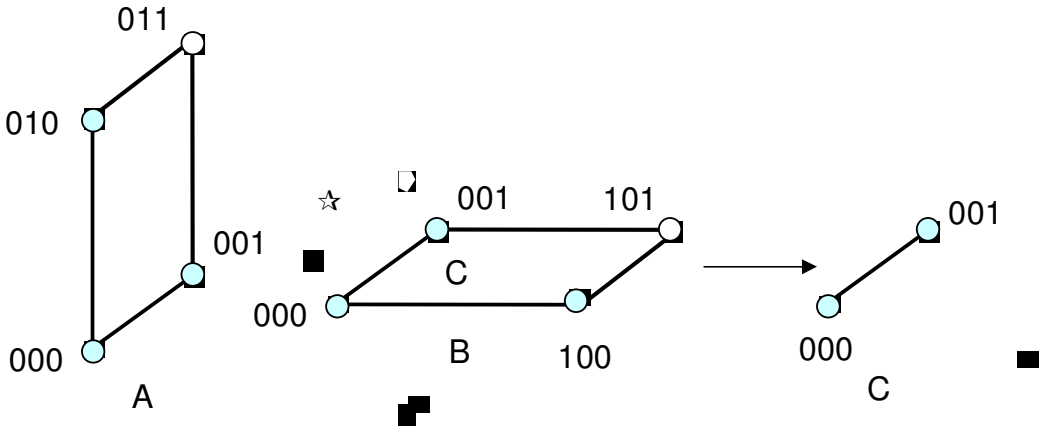
d. Eğer en az iki tane v_i ve v_j bileşeni için, $v_i = v_j$ = y olan A ve B küplerinin koordinatlı çarpımı C = ϕ dir. Burada A ve B küpleri arasında doğrudan bir bağlantı yoktur.

a, b, c, d maddelerine göre A^m (m-küp) ile B^l (l-küp) küplerinin çarpılması sonucu aşağıdaki durumlar oluşmaktadır.

1: Her iki A ve B küpüne giren C küpü (Şekil 3.2).

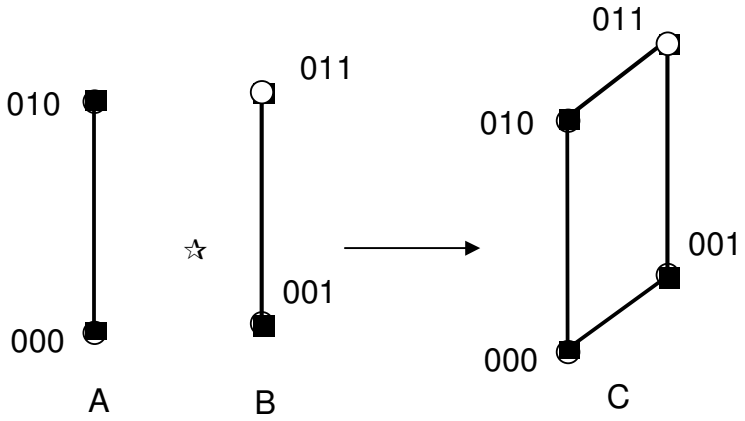
ÇV=A☆B=0** ☆ *0*

C=A*B=VP=00*



Şekil 3.2. C küpünün ortak olması durumu

2: (m+l) kenara sahip olan (m+1-küp) ve A ve B küpünün birleşmesinden oluşan C küpü

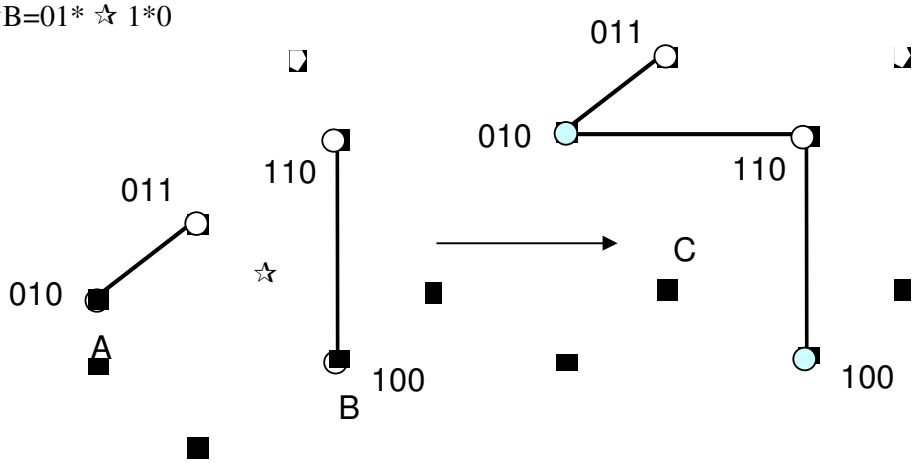


Şekil 3.3. C küpünün birleşim oluşturduğu durum

ÇV=A☆B=0*0☆ 0*1 = 0y*

3: A ve B küpleri arasında birleştirilmiş bir köprü olan C küpü.

ÇV=A☆B=01* ☆ 1*0



Şekil 3.5. C küpünün köprü oluşturduğu durum

3.2.2.2 Koordinatlı çıkarma işlemi (# işlemi)

Koordinatlı çarpmada olduğu gibi, koordinatlı çıkarmada da aynı boyuta sahip iki küp kullanılır. Çıkarma işlemi, küplerin aynı taraflarında (nokta, kenar, yüzeylerinde) veya farklı taraflarında yapılabilir. Koordinatlı çıkarma işlemi ilk olarak, tepe noktası adreslenmemiş SFs nın sınırını hesaplamada kullanılmıştır. Bununla birlikte, SFs nın yerel asal implikantların (local prime implicants) bulunmasına uygulanmıştır (Allahverdi ve Kahramanlı 1995, Kahramanlı ve Allahverdi 1993, , Kahramanlı ve Allahverdi 1996, Güneş ve ark. 2003, Allahverdi 1999).

A ve B gibi aynı boyuta sahip iki vektör verilsin

$A = a_1\, a_2 ... a_n,\quad B = b_1\, b_2 ... b_n$

Çıkarım Vektörü ÇV= $v_1\, v_2 ... v_n$ şu şekilde bulunur.

* Eğer $b_i = x$ VEYA $b_i = a_i$ ise $v_i = Z$
* Eğer $a_i = x$ VE $b_i = a_i$ ise $v_i = b'_i$

* Eğer $a_i = b'_i$ ise $v_i = Y$

İkinci adım olarak koordinat değerlerine göre sonuç şu şekilde bulunur:

* Eğer $v_i = y$ ise çıkarım işlemi olamaz: C=A # B=A

* Eğer hiç $v_i = y$ yoksa ÇV= $v_j ... v_k ... v_m \varepsilon \{ 0,1 \}$ varsa çıkarım operasyonu

Sonucu şu şekildedir:

$$\{a_1 \ldots a_{j-1} a_j \; a_{j+1} \ldots a_n, \; a_1 \ldots a_{k-1} a_k \; a_{k+1} \ldots a_n, \; a_1 \ldots a_{m-1} a_m \; a_{m+1} \ldots a_n\}$$

* Eğer i için $v_i = Z$ ise işlem sonucu boş kümedir. C=A # B=∅

a_i \ b_i	X	1	0
X	Z	0	1
1	Z	Z	Y
0	Z	Y	Z

Koordinatlı Çıkarma İşleminin özellikleri:

$A \# B \neq B \# A$ Değişme özelliği yoktur.

$(A \# B) \# C \neq A \# (B \# C)$ Birleşme özelliği yoktur.

$(A \cup B) \# C = (A \# C) \cup (B \# C)$ Birleşme özelliği üzerinde dağılma özelliği vardır

$(A \cap B) \# C = (A \# C) \cap (B \# C)$ Kesişme özelliği üzerinde dağılma özelliği vardır.

$A \# \{B, C\} = (A \# B) \# C = A \# B (\# C)$ Çıkarma işleminde simetriklik vardır.

3.2.2.3 Dönüşümlü Yutma İşlemi

Bu işlem iki adımda gerçekleşmektedir.

1- Vektör Absorbe işlemi

2- Koordinat değerlerine bağlı olarak kurallar uygulanarak Sonuca varılır.

Vektör absorbe işlemi şu kurala göre yapılmaktadır: $AV=A\nabla B= v_1,v_2... v_i... v_n$

* Eğer $a_i = b_i$ ise $v_i = Z$
* Eğer $a_i = x$ VEYA $b_i \neq a_i$ ise $v_i = G$
* Eğer $a_i = b'_i$ ise $v_i = Y$
* Eğer $a_i \neq x$ VEYA $b_i = a_i$ ise $v_i = L$

a_i / b_i	X	1	0
X	Z	G	G
1	L	Z	Y
0	L	Y	Z

Koordinat değerlerine bağlı olarak Sonuç çıkarma:

* Eğer $\exists i$ için $v_i = Y$ ise absorbe işlemi yapılamaz $C = A\Delta B = \{A,B\}$
* Eğer $\forall i$ için $v_i = Z$ ise A=B dir ve Sonuç $C = A\Delta B = A$
* Eğer ($\exists i$ için $v_i = G$) ve (değil $\exists i$ için $v_i = L$) ise $C = A\Delta B = A$
* Eğer ($\exists i$ için $v_i = L$) ve (değil $\exists i$ için $v_i = G$) ise $C = A\Delta B = B$
* Eğer ($\exists i$ için $v_i = G$) ve ($\exists i$ için $v_i = L$) ise absorbe işlemi yapılamaz $C = A \Delta B = \{A, B\}$

Ör:

1. A=X1XX, B=X1X1 AV= X1XX ∇ X1X1 = ZZZG; C=X1XX
2. A=XX1X, B=X011 AV= XX1X ∇ X011 = ZGZG; C=XX1X

3.2.2.4 Asal İmplikantların Yerel Belirlenmesi

Teorem:

Farz edelim ki A= $a_1a_2...a_j... a_n$ DOĞRU kümesinin elemanı olsun,

B= $b^k_1b^k_2... b^k_j... b^k_n$ ise YANLIŞ kümesi elemanı olsun.

$K_i = K_{i-1} \# B_i$ i=1,2...m K_0 =XX... X

Böylece DOĞRU kümesindeki her bir eleman için YANLIŞ kümesinin bütün elemanları üzerinde bu işlem gerçekleşir. Bu işlem sayesinde a_i değeri uygun dönüşümü sağlar.

İspat:

Eğer $a_i = b^k_i$ ise herhangi bir j koordinatı için. $v_i = x \# b^k_i = a'_i$

Sonuç olarak $a_i = b^k_i$ olduğunda (a_i, b^k_i) ikilisinde fark küpü oluşur (difference cube) yani A minterm' ünü içermez.

Eğer a_i <> x VE b^k_i = x j koordinatı için fark küpü oluşmaz yani b^k_i Değeri değişmez.

Ve yine Eğer $a_i = b'^k_i$ ise $v_i = x \# b^k_i = b'^k_i$ =a j koordinatı (ekseni) için fark küpü a_i değerini kapsar.

Bu teoremi kullanarak şu küp değişimini gerçekleştirebiliriz.

Eğer b^k_i = x ise q^k_i = x

Eğer $a_i = b^k_i$ ise q^k_i = x

Eğer $b^k_i = a'_i$ ise $q^k_i = b^k_i$

3.2.2.5 Koordinatlı Kesişme İşlemi ($\cap$ işlemi)

Bu işlem iki küp arasında mevcut olabilecek altkübün boş olup olmadığını belirlenmesini amaçlamaktadır. C_1 ve C_2 küplerinin kesişme işlemini sonucu bir vektördür. (VK)

$$VK = v_1, v_2 ... v_i ... v_n$$

VK vektörünün değeri şu eşitliklerle verilir:

Eğer $a_i = b_i$ ise $v_i = b_i = a_i$ dir.

Eğer $a_i = *$ ve $b_i \neq *$ ise $v_i = b_i$ dir.

Eğer $b_i = *$ ve $a_i \neq *$ ise $v_i = a_i$ dir.

Eğer $a_i, b_i \in \{0,1\}$ *ve* $a_i \neq b_i$ ise v_i =y dir.

Kesişme işleminin pseudokodu aşağıda verilmiştir.

Procedure KOORDİNATLIKESİŞME ()

$C_L = A_L \oplus B_L$;

$C_R = A_R \oplus B_R$;

Sonuç = C_L & C_R;

END Procedure;

3.2.3 Yakın-Minimali Örtme Algoritması

Bu algoritmanın mantıksal fonksiyonun en sade şekillerinden birini üretir, ama bu algoritmayla üretilen Sonuç en sade Sonuç olmayadabilir. Genelde en sade Sonuç için Sdoğru kümesinin sıralanması gerekmektedir.

Algoritmanın adımları şu şekildedir:

1. i değişkenine sıfır ata (i = 0)
2. Sdoğru kümesinin birinci elemanını al ve i değişkenini artır (i=i+1)
3. Syanlış kümesinin her bir elemanına bağlı olarak kural x ile verilen dönüşümü gerçekleştir. Sonuçlar Q0 kümesinde olsun.
4. Q0 kümesine absorbe işlemini uygula ve Sonuçlar Q1 kümesini oluştursun
5. n boyutlu 'Bütün Küp' ten (xx... x)koordinat çıkarma işlemini uygula. (n fonksiyona giren değişken sayısını gösterir). Sonuc SI olarak adlandır.
6. SI kümesi elemanlarına 'BÜYÜK' ve 'KÜÇÜK' işlemlerini uygula. (not: a, b' den daha büyüktür $\Leftrightarrow$ Sdoğru#a $<$ Sdoğru#b)
7. SI kümesinden bütün 'küçük' elemanları (güçsüz) çıkart Sonuçta tek eleman kalmışsa onu veya birkaç taneden birini seç ve bu elemana EI
8. Sdoğru kümesini yeniden oluştur ve EI elemanını SPI Sonuç kümesine ekle
9. Eğer Sdoğru $\neq \varnothing$ ise 2 ye git
10. Bitir.

3.2.3.1 YMÖA Örneği

QM algoritmasında verilen örneği burada çözelim çözümlersek
F(a,b,c,d)= $\sum$m(0, 1, 2, 5, 6, 7, 8, 9, 10, 14)

Sdoğru = {0, 1, 2, 5, 6, 7, 8, 9, 10, 14}
Syanlış = {3, 4, 11, 12, 13, 15}

h=0000

Syanlış	Q0	Küp durumu	Q1
0011	XX11	Birincil	XX11
0100	X1XX	Birincil	X1XX
1011	1X11	XX11 tarafından yutulur	---
1100	11XX	X1XX tarafından yutulur	---
1101	11X1	X1XX tarafından yutulur	---
1111	1111	XX11 tarafından yutulur	---

Q1={ XX11, X1XX,} Tam Küpten koordinat çıkarma işlemi:

S1=XXXX # Q1 =(XXXX #XX11) #X1XX)= {XX0X, XXX0 }#X1XX = {X00X, X0X0}

S1.1= X00X

S1.2 = X0X0

P1=Sdoğru # X00X ={0000, 0001, 0010, 0101, 0110, 0111, 1000, 1001, 1010, 1110}#X00X

P1={0010, 0101, 0110, 0111, 1010, 1110} (6 elemanlı)

P2=Sdoğru # X0X0 ={0000, 0001, 0010, 0101, 0110, 0111, 1000, 1001, 1010, 1110}#X0X0

Sdoğru={0001, 0100, 0101, 0111, 1001, 1110} (6 elemanlı)

Her ikisi de eşit güçte oldukları için S1.1 seçebiliriz. Böylece Sdoğru kümemiz şu olur Sdoğru={0010, 0101, 0110, 0111, 1010, 1110} (2, 5, 6, 7, 10, 14) ve SPI kümemize S1.1 eklenirse:

SPI={X00X}

Şimdi Sdoğru kümesi Boş küme olmadığı için aynı işlemleri yeniden başlatacaz i=i+1

i = 2

h2=0010

Syanlış	Q0	Küp durumu	Q1
0011	XXX1	Birincil	XXX1
0100	X10X	Birincil	X10X
1011	1XX1	XXX1 tarafından yutulur	---
1100	110X	XXX1 tarafından yutulur	---
1101	1101	XXX1 tarafından yutulur	---
1111	11X1	XXX1 tarafından yutulur	---

S2 = XXXX # Q1 =((XXXX #XXX1)#X10X)= (XXX0 # X10X) =

{ X0X0, XX10 }

S2.1 = X0X0

Sdoğru = Sdoğru # X0X0= {0010, 0101, 0110, 0111, 1010, 1110}# XXX0

Sdoğru ={0101, 0110, 0111, 1110} // 4 elemanlı

S2.2 = XX10

Sdoğru = Sdoğru# XX10= {0010, 0101, 0110, 0111, 1010, 1110}# XXX0

Sdoğru ={0101, 0111} // 2 elemanlı

S2.2, S2.1 elemanından daha güçlüdür.

SPI = {X00X, XX10}

Sdoğru kümesi boş küme olmadığı için aynı işlemler tekrarlanacaktır. i=i+1

i = 3

h3 = 0101

Syanlış	Q0	Küp durumu	Q1
0011	X01X	Birincil	X01X
0100	XXX0	Birincil	XXX0
1011	101X	X01X absorbe eder	---
1100	1XX0	XXX0 absorbe eder	---
1101	1XXX	Birincil	1XXX
1111	1X1X	1XXX absorbe eder	---

S3 = XXXX # Q1 = (((XXXX # X01X) # XXX0) # 1XXX)

= (({X1XX, XX0X} # XXX0) # 1XXX) = ({X1X1,XX01} # 1XXX)

= {01X1,0X01}

S3.1 = 01X1

S3.2 = 0X01

P1 = Sdoğru # 01X1 ={ 0101, 0111} # 01X1

P1 = ∅ S3.1 > S3.2

P2 = Sdoğru # 0X01 ={0101, 0111} # 0X01

P2 = {0111}

S3.1 > S3.2 Olduğu için S3.1 SPI kümesine ekleyelim

SPI = {X00X, XX10, 01X1}

Sdoğru: SPI kümesini değişkenlerle ifade ettiğimizde:

F = B'C' + CD' + A'BD

3.3 Küp Cebri İşlemlerinin Temel Bilgisayar İşlemleri Üzerinden Gerçekleştirilmesi

Bu işlemelerin hepsi küp cebri işlemleri kullanılarak sadeleştirme işlemi paralel bir biçimde bir şekilde yapılmaktadır. Seri bir şekilde bu işlemleri gerçekleştirirken algoritmaların çözüme ulaşma zamanları artmaktadır. Bu yüzden, bu işlemleri temel bilgisayar işlemleri yardımı ile paralel bir şekilde yaparak algoritmaların E çözüm zamanları azaltılmaya çalışılmıştır. Bu işlemler sayesinde algoritmaların daha hızlı olması sağlanmıştır. Çünkü bu işlemler temel bilgisayar işlemleri kullanılarak paralel bir şekilde yürütülmesi sağlanmıştır. Bu bölümde bu işlemlerin nasıl gerçekleştirildiği açıklanacaktır.

Temel bilgisayar işlemlerinin gerçekleştirilmesi sırasında aşağıda gösterilen işlemler kullanılacaktır.

1) Küplerin koordinat değerlerinin gösterilmesi

Bir küpün her bir koordinat değeri aşağıda gösterildiği gibi iki bit ile temsil edilmiştir. Bir lojik Fonksiyon f: $B^n \rightarrow \{0, 1, x\}$ tanımlanabilir. f Fonksiyonun alabileceği değerler olan 0 1 için ve x için dönüşümleri kullanılacaktır.

Esas koordinatın lojik 0 değeri : $0 \rightarrow \begin{matrix} 0 \\ 1 \end{matrix}$,

Esas koordinatın lojik 1 değeri : $1 \rightarrow \begin{matrix} 1 \\ 0 \end{matrix}$,

x veya (-) terimi ile gösterilen esas olmayan koordinatın değeri: $x \rightarrow \begin{matrix} 1 \\ 1 \end{matrix}$

Burada belirtilen bit çiftleri ile küpün koordinat değerlerinin temsil edilmesi sağlanmıştır. Çünkü küpün koordinat değerleri arasında gösterilecek esas olmayan koordinat değerleri için x sembolü kullanılmıştır. Bu işareti bir ve sıfır cinsinden ifade edebilmek için bu şekilde bir gösterim kullanılmıştır. Örnek olarak A= x01 küpü için aşağıda küpün her bir koordinat değeri iki bit ile gösterilmiştir.

Tablo 3.3. Bir küpün her bir koordinat değerinin iki bit ile gösterilmesi:

Küp Değeri	x	0	1
İki bit ile gösterilir	11	01	10

a) Off kümesindeki mintermleri genişletmek için temel bilgisayar işlemleri

Off-küp kümesinde bulunan $B_i = b^i_1, b^i_2 \ldots b^i_n$ e Sof mintermleri On-küp kümesinin

$A_i = a^i_1, a^i_2 \ldots a^i_n$ ile genişleterek $Q_i = q^i_1, q^i_2 \ldots q^i_n$

Bu işlemleri gerçekleştirirken 0ff kümesindeki her bir mintermi On kümesindeki minterm ile bit bit karşılaştırma yapmak suretiyle Q1 küpünü elde ederiz. Bu da zaman açısından büyük kayıplara uğramamıza neden olur. Bu kuralları temel bilgisayar işlemleri ile aşağıdaki gibi gerçekleştirebiliriz.

On kümesindeki mintermi (A), Off kümesindeki mintermle genişleterek C küpü elde edilir. Bu genişletme işlemi için On kümesindeki minterm ile Off kümesindeki minterm bit dizisi çiftleri haline dönüştürülürler.

b) İki küpün (A ve B) kesişimini temel bilgisayar işlemleri ile gerçekleştirmek

Uygulanacak iki küp olsun. Aşağıda gösterilen C küpü A ve **B** küplerinin kesişimi sonucunda elde edilen küptür.

$B_i = b^i_1, b^i_2 \ldots b^i_n$ e Sof mintermleri On-küp kümesinin

$A_i = a^i_1, a^i_2 \ldots a^i_n$ ile genişleterek

$C_i = A_i \cap B_i$

Bu kesişim işlemi sonucunda elde edilen C küpünün bit dizisi çiftleri CL ve CR bit dizileri A ve B küplerinin bit dizilerinin ‘ve” işlemine tabi tutulması ile bulunur. Burada,

CL =Al VE Bl

CR = Ar VE Br

Elde edilen CL ve CR bit dizileri sonucunda, A ve B küplerinin kesişim değeri belirlenmeye çalışılır. C küpünün değerinin belirlenmesi için aşağıdaki işlemler gerçekleştirilir.

1) Sonucun boş küme olup olmadığının kontrol edilmesi

Bulunan C1 ve Cr bit dizileri ve C küpü değeri ile A ve B küplerinin kesişim değerinin herhangi bir değere veya boş kümeye eşit olup olmadığı aşağıdaki işlemlerle kontrol edilir.

$C_L = A_L \oplus B_L$;

$C_R = A_R \oplus B_R$;

Sonuç = C_L & C_R;

Bu işlemler sonucunda elde edilen sonuç küpü sıfıra eşit değilse C küpü hoş kümedir, Yani, A ve B küplerinin kesişiminden bir değer elde edilmemiştir. Sonuç küpü sıfıra eşitse A ve B küplerinin kesişiminden bir değer elde edilmiş olacaktır.

ii) A ve B küplerinin C sonuç küpü ile karşılaştırılması

A ve B küplerinin kesişimi sonucunda elde edilen C küpü herhangi bir değere sahipse, bu değerin A veya 13 küplerinden hangisine ait olduğunu aşağıdaki işlemler doğrultusunda bulabiliriz. Bu işlemler sonucunda kesişim değerinin hangi küpe ait olduğunu bulmakla beraber küpler arasındaki kapsama durumları da bulunmuş olmaktadır.

- Eğer Cl = A1 ve Cr=Ar ise C=A dır.
- Eğer C = A ise A küpünü çıkar değilse A küpünü tut.
- Eğer C= B ise B küpünü çıkar değilse B küpünü tut.

Örnek 3.3: A = 0 x 1 ve B= 0 0 1 iki küp olsun. Bu küpler arasındaki kesişme durumunu ve birbirini kapsama durumunu temel bilgisayar işlemleri ile gerçekleştirirsek;
A ve B küplerinden A1, Ar ve Bl - Br bit dizilerini aşağıdaki gibi oluştururuz.

Al= 0 1 1 Ar = 1 1 0

Bl= 0 0 1 Br = 1 1 0

Elde edilen bit dizilerinden
C1 =A1 VE Bl=0 1 1 VE 0 0 1 = 0 0 1

Cr =Ar VE Br=1 1 0 VE 1 1 0 = 1 1 0

C=A $\cap$ B= 0 0 1

i) Sonucun boş küme olup olmadığının kontrol edilmesi
D=(Cl VEYA **Cr)** $\oplus$ 11… 1 D =(001 v 110) $\oplus$ 111 = 000 olduğu için C $\neq \phi$ dir.
Yani bu işlemler sonucunda A ve B küplerinin kesişiminden bir değer elde edilmiştir.

Küpler arasındaki kapsarna durumuna bakılırsa,
Cl=Bl=011

Cr=Br=110

Olduğundan B küpü kapsanmıştır.

3.4 Yakın Minimali Örtme Algoritması Pseudo Kodu

Tahmini minimali Son kümesi boş kümeden farklı olduğu müddetçe genişletme işlemi, değişmeli absorbe, koordinatlı çıkarma işlemleri verilen sırayla uygulanır. En son olarak asal implikantlar kümesi üzerinde büyük işlemi uygulanarak en büyük asal implikant bulunur ve Sespi kümesine eklenir.

```
Procedure NMİNİMAL(Son, Sof)

        While Son ∈ Φ Do

            SQ0=GENİŞLEME(Son[0], Sof);
            SQ1=DEGİŞMELİABSORBE(SQ0);
            SAI=KOORDİNATLI_ÇIKARMA(SQ1);
            ESPIBÜYÜK (Son, SAI);
            SESPI= SESPI ∪ ESPI;

        END While;
END Procedure;
```

4 SADELEŞTİRME ALGORİTMALARININ KARMAŞİKLIK ANALİZİ

4.1 Karmaşıklık (Complexity)

Bir programın performansı genel olarak programın işletimi için gerekli olan bilgisayar zamanı ve belleğidir. Bir programın zaman karmaşıklığı (time complexity) programın işletim süresidir. Bir programın yer karmaşıklığı (space complexity) programın işletildiği sürece gerekli olan yer miktarıdır. Bir problemin çözümünde, kullanılabilecek olan algoritmalardan en etkin olanı seçilmelidir. En kısa sürede çözüme ulaşan veya en az işlem yapan algoritma tercih edilmelidir. Burada bilgisayarın yaptığı iş önemlidir. Bazı durumlarda da en az bellek harcayan algoritmanın tercih edilmesi gerekebilir. Ayrıca, programcının yaptığı iş açısından veya algoritmaların anlaşılırlıkları bakımından da algoritmalar karşılaştırılabilir. Daha kısa sürede biten bir algoritma yazmak için daha çok kod yazmak veya daha çok bellek kullanmak gerekebilir.

Rakip algoritmaları yaptıkları iş açısından karşılaştırmak için her algoritmaya uygulanabilecek somut ölçüler tanımlanmalıdır. Aynı işi yapan algoritmalardan daha az işlemde sonuca ulaşanın (hızlı olanın) belirlenmesi yani daha genel olarak algoritma analizi teorik bilgisayar bilimlerinin önemli bir alanıdır.

Yazılımcılar, iki farklı algoritmanın yaptıkları işi nasıl ölçüp karşılaştırırlar? İlk çözüm algoritmaları bir programlama dilinde kodlayıp her iki programı da çalıştırarak işletim sürelerini karşılaştırmaktır. İşletim süresi kısa olan daha iyi bir algoritma denilebilir mi? Bu yöntemde işletim süreleri belirli bir bilgisayara özeldir. Dolayısı ile işletim süresi de bu bilgisayara bağlıdır. Daha genel bir ölçüm yapabilmek için olası tüm bilgisayarlar üzerinde algoritmanın çalıştırılması gerekir.

İkinci çözüm, işletilen komut ve deyimlerin sayısını bulmaktır. Fakat bu ölçüm kullanılan programlama diline göre ve programcıların stiline göre değişim gösterir. Bunun yerine algoritmadaki kritik geçişlerin sayısı hesaplanabilir. Her tekrar için sabit bir iş yapılıyor ve sabit bir süre geçiyorsa, bu ölçü anlamlı hale gelir.

Buradan, algoritmanın temelinde yatan bir işlemi ayırarak, bu işlemin kaç kere tekrarlandığını bulma düşüncesi doğmuştur. Örnek olarak bir tamsayı dizisindeki tüm elemanların toplamını hesaplama işleminde gerekli olan iş miktarını ölçmek için tamsayı

toplama işlemlerinin sayısı bulunabilir. 100 elemanlı bir dizideki elemanların toplamını bulmak için 99 toplama işlemi yapmak gerekir. n elemanlı bir listedeki elemanların toplamını bulmak için n–1 toplama işlemi yapmak gerekir diye genelleştirme yapabiliriz. Böylece algoritmaları karşılaştırırken belirli bir dizi boyutu ile sınırlı kalınmaz.

İki gerçel matrisin çarpımında kullanılan algoritmaların karşılaştırılması istendiğinde, matris çarpımı için gereken gerçel sayı çarpma ve toplama işlemlerinin karışımı bir ölçü olacaktır. Bu örnekten ilginç bir sonuca ulaşılır: Bazı işlemlerin ağırlığı diğerlerine göre fazladır. Birçok bilgisayarda bilgisayar zamanı cinsinden gerçel sayı çarpımı gerçel sayı toplamından çok daha uzun sürer. Dolayısı ile tüm matris çarpımı düşünüldüğünde toplama işlemlerinin etkinlik üzerindeki etkisi az olacağından ihmal edilebilirler. Sadece çarpma işlemlerinin sayısı dikkate alınabilir. Algoritma analizinde genelde algoritmada egemen olan bir işlem bulunur ve bu diğerlerini gürültü düzeyine indirger.

4.2 Algoritmalarda Karmaşıklık (Complexity) ve Zaman Karmaşıklığı Analizi

4.2.1 İşletim Zamanı (Running Time)

İşletim zamanını girdi boyutunun bir fonksiyonu olarak ele almak tüm geçerli girdileri tek değere indirir. Bu da değişik algoritmaları karşılaştırmayı kolaylaştırır. En yaygın karmaşıklık ölçüleri "Worst –Case Running Time" (en kötü durum işletim süresi) ve "Average-Case Running Time" (ortalama durum işletim süresi)'dir. (Stockmeyer 1990).

En kötü çalışma süresi:

Bu işletim süresi, her girdi boyutundaki herhangi bir girdi için en uzun işletim süresini tanımlar. Örnek olarak bir programın en kötü ihtimalle ne kadar süreceğinin tahmin edilmesi istenen bir durumdur. n elemanlı bir listede sıralı arama en kötü ihtimalle (aranan bulunamazsa) n karşılaştırma gerektirecektir. Yani worst-case running time (işletim zamanı) T(n) = n' dir. Tüm problemlerde sadece en kötü girdi dikkate alındığı için worst-case running time değerini hesaplamak göreceli olarak kolaydır.

Ortalama çalışma süresi:

Bu işletim süresi, her girdi boyutundaki tüm girdilerin ortalamasıdır. n elemanın her birinin aranma olasılığının eşit olduğu varsayıldığında ve liste dışından bir eleman aranmayacağı varsayıldığında ortalama işletim süresi (n+1)/2'dir. İkinci varsayım kaldırıldığında ortalama işletim süresi [(n+1)/2,n] aralığındadır (aranan elemanların listede olma eğilimine bağlı olarak). Ortalama durum analizi basit varsayımlar yapıldığında bile zordur ve varsayımlar da gerçek performansın iyi tahminleşememesine neden olabilir.

4.2.2 Asimptotik Analiz

Algoritmaların karşılaştırılmasında asimptotik etkinlikleri de dikkate alınabilir. Girdi boyutu sonsuza yaklaşırken işletim süresinin artışı. Asimptotik gösterimin elemanı olan 4 önemli gösterim vardır: O-notasyonu, o- notasyonu, Ω- notasyonu, θ- notasyonu. Burada sadece O gösterimi üzerinde durulacaktır. O gösterimi, fonksiyonların artış oranının üst sınırını belirler. O(f(n)), f(n) fonksiyonundan daha hızlı artmayan fonksiyonlar kümesini gösterir.

4.2.2.1 Büyük-O Gösterimi (notasyonu)

n elemanlı bir listedeki elemanların toplamını bulmak için n-1 toplama işlemi yapmak gerekir diye genelleştirme yapmıştık. Yapılan işi, girdi boyutunun bir fonksiyonu olarak ele almış olduk. Bu fonksiyon yaklaşımını matematiksel gösterim kullanarak ifade edebiliriz: Big-O gösterimi veya büyüklük derecesi (order of magnitude). Büyüklük derecesini problemin boyutuna bağlı olarak fonksiyonda en hızlı artış gösteren terim belirler. Örnek olarak:

$f(n) = n^7 + 100n^2 + 50 = O(n^7)$

Fonksiyonunda n' in derecesi n^4'tür yani n' in büyük değerleri için fonksiyonu en fazla n^4 etkiler. Peki, daha düşük dereceli deyimlere ne olmaktadır? n' in çok büyük değerleri için n^4, $100n^2$'den ve 50'den çok büyük olacağından daha düşük dereceli terimler dikkate

alınmayabilir. Bu diğer terimlerin, işlem süresini etkilemedikleri anlamına gelmez; bu yaklaşım yapıldığında n' in çok büyük değerlerinde önem taşımadıkları anlamına gelir.

n, problemin boyutudur. Yığıt, liste, kuyruk, ağaç gibi veri yapılarında eleman sayılarıdır. n elemanlı bir dizi gibi...

Bir listedeki tüm elemanların dosyaya yazılması için ne kadar iş yapılır : Cevap, listedeki eleman sayısına bağlıdır.

Algoritma

OPEN (Rewrite) the file
WHILE more elements in list DO
    Print the next element

İşlemi yapmak için geçen süre:

(n*(Bir elemanın dosyaya yazılması için geçen süre))+dosyanın açılması sırasında geçen süre Algoritma O(n)'dir (Algoritmanın zaman karmaşıklığı O(n)'dir) . Çünkü n tane işlem + sadece dosya açılması işlemi vardır. Yüzlerce elemanın dosyaya kaydedildiği düşünülürse, dosya açılması sırasında geçen süre miktarı rahatlıkla ihmal edilebilir. Ama az sayıda eleman varsa dosya açılması sırasında geçen süre miktarı önem taşıyabilir ve toplam süreye katılımı daha fazla olur.

Bir algoritmanın büyüklük derecesi, bilgisayarda işletildiğinde sonucun ne kadar sürede alınacağını belirtmez. Bazen de bu tür bir bilgiye gereksinim duyulur. Örnek olarak bir kelime işlemcinin 50 sayfalık bir yazı üzerinde yazım denetimi yapma süresinin birkaç saniye düzeyinden fazla olmaması istenir. Böyle bir bilgi istendiğinde, Big-O analizi yerine diğer ölçümler kullanılmalıdır. Program değişik yöntemlere göre kodlanır ve karşılaştırma yapılır. Programın çalıştırılmasından önce ve sonra bilgisayarın saati kaydedilir. İki saat arasındaki fark alınarak geçen süre bulunur. Bu tür bir "Benchmark" testi, işlemlerin belirli bir bilgisayarda belirli bir işlemci ve belirli kaynaklar kullanılarak ne kadar sürdüğünü gösterir.

Bilgisayarın yaptığı işin programın boyutu ile, örnek olarak satır sayısı ile ilgili olması gerekmez. N elemanlı bir diziyi 0'layan iki program da O(n) olduğu halde kaynak kodlarının satır sayıları oldukça farklıdır:

Program 1:

```
Dizi[0] = 0;
Dizi[1] = 0;
Dizi[2] = 0;
Dizi[3] = 0;
…
Dizi[n–1] = 0;
```

Program 2:

```
for(int i=0; i<n; ++i)
        Dizi[i] = 0;
```

1'den n' e kadar olan sayıların toplamını hesaplayan iki kısa programı düşünelim:

Program 1:

```
Toplam = 0;
for(int i=0; i<n; ++i)
        Toplam = toplam + i;
```

Program 2:

```
Toplam = n * (n+1) / 2;
```

Program 1, O(n)'dir. n=50 olursa programın çalışması sırasında n=5 için harcanan sürenin yaklaşık 10 katı süre harcanacaktır. Program 2 ise O(1)'dir. n=1 de olsa n=50'de olsa program aynı sürede biter.

Şekil 4.1 Büyük O ifadeleri ve anlamları

1.1 Fonksiyon	**1.2 İsim**
1	Sabit
Log(n)	Logaritmik
n	Doğrusal
n^x	Polinomal
x^n	Üssel
n!	Faktöriyel

Şekil 4.1: Büyük O ifadeleri

O(1) : Sabit zaman

Örnek: n elemanlı bir dizinin i. elemanına bir değer atanması O(1)'dir. Çünkü bir elemana indisinden doğrudan erişilmektedir.

O(n) : Doğrusal zaman

Örnek: n elemanlı bir dizinin tüm elemanlarının ekrana yazdırılması O(n)'dir.

Örnek: sıralı olmayan bir dizideki (listedeki) elemanlardan birinin aranması O(n)'dir (en kötü durumda da, ortalama durumda da).

$O(\log_2 n)$: O(1)'den fazla O(n)'den azdır.

Örnek: Sıralı bir listenin elemanları içinde ikili arama (binary search) uygulanarak belirli bir değerin aranması $O(\log_2 n)$'dir.

$O(n^2)$: İkinci dereceli zaman

Örnek: Basit sıralama algoritmalarının birçoğu (selection sort gibi) $O(n^2)$'dir.

$O(n \log_2 n)$: Bazı hızlı sıralama algoritmaları $O(n \log_2 n)$'dir.

$O(n^3)$: Kübik zaman

Örnek: Üç boyutlu bir tamsayı tablosundaki her elemanın değerini artıran algoritmadır.

$O(2^n)$: Üstel zaman, çok büyük değerlere ulaşır.

4.2.2.2 Pratikte Karmaşıklık

Değişik artış fonksiyonlarının aldıkları değerlere göre bir tablo, Şekil 4.2'de gösterilmiştir.

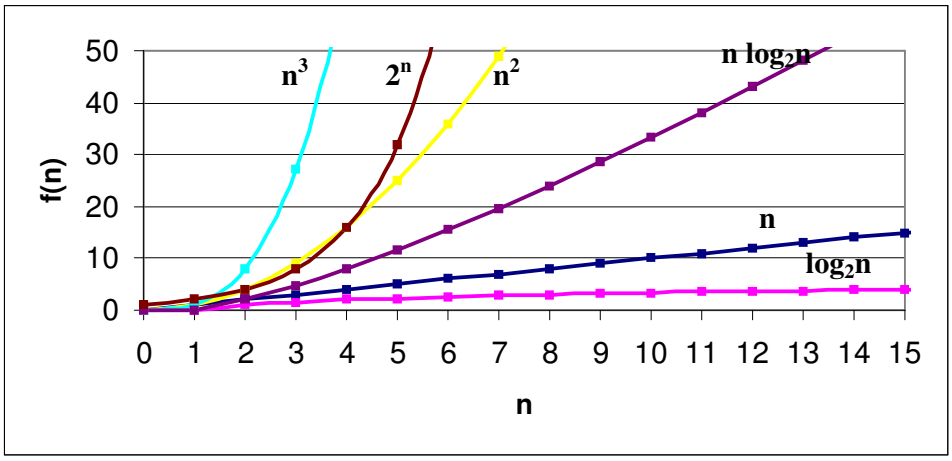
logn	**n**	**nlogn**	**n^2**	**n^3**	**2^n**
0	1	0	1	1	2
1	2	2	4	8	4
2	4	8	16	84	16
3	8	24	64	512	256
4	16	64	256	4096	65536
5	32	160	1024	32768	4294967 296

Şekil 4.2: Değişik fonksiyonların f(n) değişik girdi boyutlarına (n) göre değerleri

Bir programın işletimi n^3 adım sürüyorsa ve n=1000 ise, program 1000^3 adım sürecek demektir. Yani 1.000.000.000 adım.

Kullanılan bilgisayar saniyede 1.000.000.000 adımı gerçekleştirebilecek kadar hızlı ise bu program tam 1 saniye sürecektir.

Şekil 4.2' deki fonksiyonlardan elde edilmiş bir grafik Şekil 4.3'te görülmektedir.



Şekil 4.3: Değişik fonksiyonların grafikleri

4.3 Algoritmaların Karmaşıklık Değerlendirmesi

Bu bölümde Yakın Minimali Örtme Metodunun asal implikantları oluşturan kısmının karmaşıklığı karşılaştırmalı olarak hesaplanacaktır. Çünkü bunların esas asal implikant belirleme ve örtme kısımları var olanlar ile aynıdır.

Verilen On-küpü örtmek için asal implikantların yeterli bir kümesini oluşturan belli buluşsal metotlar doğrudan örtme prensibini kullanarak aşağıdaki algoritmaya göre çalışırlar (Fiser ve Hlavıcka 2003).

1. Yeterli asal implikantlar kümesini elde etmek için bir tane On küp seçmek.

2. Bu On küpü örten implikantları üretmek

3. Verilen implikantı, henüz dokunulmamış literallerden (koordinat değerleri) bir tanesini çıkararak (x ile değiştirerek) genişletmek

4. Sonucu, Off kümesi ile kesiştirmek

5. Eğer kesişme işleminin sonucu boş değilse çıkarılan literali geri koymak

6. Eğer işleme tabi tutulmayan bir literal varsa 3. adıma geri dönmek, değilse devam etmek

7. Genişletilen implikantı, asal implikantlardan bir tanesi olarak kaydetmek

8. Eğer bütün implikantlar genişletildiyse 11. adıma gitmek değilse devam etmek

9. Henüz işleme tabi tutulmamış implikantlardan birisini yeni implikant olarak kabul etmek

10. 3. adıma geri dönmek

11. Örtme problemini çözmek

Bu algoritma SON kümesi boşalıncaya kadar tekrarlanır. Bu algoritmaya dayanarak, verilen terimin literalleri (veya koordinatları) birer birer etkisiz duruma getirilir. Bu işlem söz konusu terim asal implikant oluncaya kadar sürdürülür. Bu işlemler polinornal zamanda (O(n!)) yapılabilir. Bu, matematiksel olarak aşağıdaki gibi formüle edilebilir. On-küpü i esas ve n-i esas olmayan (removed) koordinatla ifade edebilmek amacıyla genişletebilmek için i tane incelemeye ihtiyaç vardır. Bundan dolayı, genişletilen küpün koordinatlarının hepsi için gereken incelemelerin toplam sayısı aşağıdaki gibi ifade edilebilir;

$$\begin{aligned} Q &= \sum_{i=0}^{n-1} i \\ &= (1+n) * n \Big/ 2 \\ &= (n^2 + n) \Big/ 2 \end{aligned} \qquad (4.1)$$

Fakat genellikle, incelenen bir küp $0 < m < n$ koordinatları çıkarıldıktan sonra asal implikant olur. Bunu dikkate alarak, incelemelerin beklenen sayılarının genel ifadesi Tablo

4.4' ün birinci ve ikinci sütunlarında sunulan verilerin tümevarımıyla elde edilmiştir. Yani, (4.1) deki formül yerine aşağıdaki formül kullanılacaktır.

$$Q_0 = (1+m) * n - {(m^2+m)}/{2} \qquad (4.2)$$

Varsayalım ki, minimumlaştırılan n değişkenli bir fonksiyon, On-kümesi için K1 x 2 boyutuyla ve 0ff-kümesi için K2 x 2 boyutuyla gösterilmiş olsun.

Tablo 4.3 n ve m değerlerine bağlı olarak genişletilen küpün incelemelerinin sayısı

Esas olmayan koordinatların sayısı (m)	Gereken incelemenin sayısı (Q0)	Q0' in genel ifadesi
0	n	$Q_0 = \text{n(m+1)-} \sum_{i=1}^{m} i$ $Q_0 = (1+m) * n - {(m^2+m)}/{2}$
1	n+(n–1) 2n-1	
2	n+(n–1)+(n–2)= 3n–3	
3	n+(n–1)+(n–2)+(n–3)= 4n–6	
4	n+(n-l)+(n–2)+(n–3)+(n–4) = 5n-l0	
...	...	
m	n+(n-l)+(n–2)+(n–3)+(n–4)+…+(n-m) =n(m+1)-(1+2+3+…+m)	

İncelenen On-küplerin maksimum muhtemel sayısı, a < K_1 olduğu durumlarda a x 2^n dir ve genişletilmiş küp ile bire bir karşılaştırılan Off-küplerin sayısı K_2 x 2^n dir. Sonuç olarak, On-küplerin genişletilmesi metoduna dayanarak bir asal implikantın oluşturulmasının karmaşıklığı aşağıdaki formül ile ifade edilebilir.

$$Q_1 = a * 2^n Q_0 * K_2 * 2^n = a * K_2 * 2^n (2n - m)(m + 1) \quad (4.3)$$

Fakat genellikle, bir minterm birden fazla asal implikant tarafından örtülebilir. Bu nedenle sezgisel metotların çoğu d' nin başlangıç boyutunun her bir terimi için n-d âdete kadar asal implikant üretir (Fiser ve Hlavıcka 2003). Her bir izole edilmiş minterm bir tek asal implikant tarafından örtüldüğünden, bir minterm için oluşturulmuş asal implikantların ortalama sayısı yaklaşık olarak (n-d)/2 ye eşit varsayılabilir. Bütün mintermler için d sıfır olduğundan, verilen mintermi örten yeterli sayıda asal implikantların (Fiser ve Hlavıcka 2003) oluşturulmasının toplam karmaşıklığı, aşağıdaki formül ile hesaplanabilir:

$$Q_2 = a * n * K_2 * 2^{2n-1} (2n - m)(m + 1) \quad (4.4)$$

Diğer taraftan, burada önerilen metot aşağıdaki algoritmada ki gibi çalışır.

1. Asal implikantlar kümesinin tamamını elde etmek için gerekli bir tane On küp (minterm) seçmek. Bunu genişletici olarak kullanmak

2. Genişletilmesi gereken küp olarak birinci 0ff-küpü seçmek

3. Verilen 0ff-küpü genişletmek (genişletme kuralına göre)

4. Genişletilen küpü, önceden genişletilmiş olanlarla bire bir karşılaştırmak. Eğer genişletilen küp diğerlerini içeriyorsa, kapsanan küpleri silmek. Veya önceden üretilmiş küplerden birisi bu küpü içeriyorsa bu küpü silmek

5. Eğer bütün Off-küpler işleme tabi tutulduysa 8. adıma gitmek değilse devam etmek

6. Henüz işleme tabi tutulmamış bir sonraki On-küpü seçmek

7. 3. adıma geri dönmek

8. n-küp den genişletilmiş asal küpler kümesini çıkartmak

9. Asal olmayan küpleri silmek

10. Örtme problemini çözmek.

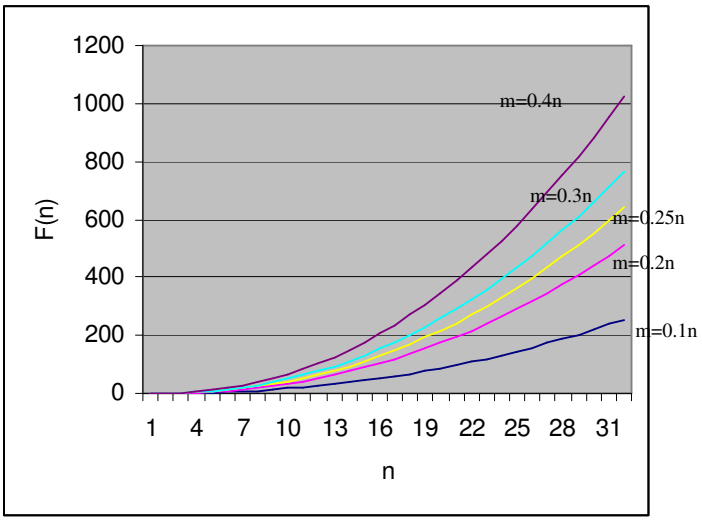
Bu algoritma SON kümesi boşalıncaya kadar tekrarlanır. Bu algoritmaya göre, verilen On-küpü örten tüm asal implikantların kümesini üretmede Off-küplerin hepsinin genişletilmesi için On-küp ile bire bir karşılaştırılması gereklidir. Bundan dolayı, her bir a x 2^n On-küp, genişletilen K_2 x 2^n 0ff-küp ile karşılaştırılır. Bununla, yeni genişletilmiş Off-küp önceden genişletilmiş olanlarla asal olmayanların silinmesi için karşılaştırılır. Yapılan uygulamalar göstermiştir ki buna benzer karşılaştırmaların sayısı n/2' yi geçmemektedir. Sonuç olarak, sunulan metodun karmaşıklığının en kötü zaman değeri:

$$Q_3 = a * 2^n * K_2 * 2^n (1 + \quad {}^{n}\!/_{2}) = a * (2+n) K_2 * 2^{2n-1} \tag{4.5}$$

Bu yüzden, burada sunulan asal implikant üretme algoritmasının var olan herhangi bir algoritmaya göre aşağıdaki değer kadar daha hızlı gerçekleştirilebilir olmasını beklemek mümkündür.

$$F(n,m) = {}^{Q_2}\!/_{Q_3} = \frac{a*n*K_2*2^{2n-1}(2n-m)(m+1)}{a*(2+n)K_2*2^{2n-1}} \tag{4.6}$$

Daha fazla görsellik için, Şekil 4.2' de m' nin farklı değerlerine karşılık gelen F(n)' nin eğrileri gösterilmiştir. Yaklaşık olarak m' nin ortalama değerinin 0.25n alınmıştır.



Şekil 4.2: n ve m değerleri artarken F(n,m) çok hızlı artar

4.4 QMM Aralık Değerleri Sayısının Bulunması ve Karmaşıklık Değerlendirilmesi

Fonksiyonun çıkışlarının 1 olduğu kümenin elemanları (mintermleri) içerdikleri 1 sayısına göre gruplandırılır. n değişkenli fonksiyon için n+1 tane altküme vardır. Birincisi sıfırlar altkümesidir, hiç 1 elemanı içermez. Bir tane 1 elemanı içerenler birler altkümesidir. n tane 1 elemanı içerenler n. altkümedir. Buradan i. altkümenin i tane 1 elemanı içerdiğini söyleyebiliriz. Bundan dolayı i. altkümenin gücü (Mano 2002, Miller 1965),

$$P_i = C_n^{\,i} \tag{4.7}$$

Bir altkümedeki bütün mintermler bir sonraki altkümenin bütün mintermleri ile karşılaştırılır. Örneğin; ikinci altkümenin mintermleri, üçüncü altkümenin mintermleri ile karşılaştırılır. i. küme ile (i+1). kümenin karşılaştırılmasının asimptotik değeri ve bütün kümelerin birbirleri ile karşılaştırılmasından oluşan toplam asimptotik değeri, sırasıyla aşağıdaki formüller ile hesaplanır (Mano 2002, Kahramanlı ve Başçiftçi)

$$W_i = C_n^{\,i} x C_n^{\,i+1} \tag{4.8}$$

Ve

$$W_T = \sum_{i=0}^{n-1} (C_n^{\ i} x C_n^{\ i+1}) \qquad (4.9)$$

i. altküme ile (i+l). Altkümenin karşılaştırılmasının sonucunda boş olmayan sonuçların asimptotik değeri ve bu sonuçların toplam asimptotik değeri aşağıdaki formüller ile hesaplanır (Mano 2002, Kahramanlı ve Başçiftçi)

$$R_i = (n - i)\, x C_n^{\ i} \qquad (4.10)$$

Ve

$$R_T = \sum_{i=0}^{n-1} ((n - i)\, x C_n^{\ i}) \qquad (4.11)$$

Yukarıda da değinildiği gibi, bütün karşılaştırmaların asimptotik değeri ve boş olmayan bütün sonuçlar (AAA) ve (AAA) da gösterilen ve aşağıda tekrar verilen formüller ile hesaplanır.

$$W_T = \sum_{i=0}^{n-1} (C_n^{\ i} x C_n^{\ i+1}) \qquad (4.12)$$

Ve

$$R_T = \sum_{i=0}^{n-1} ((n - i)\, x C_n^{\ i}) \qquad (4.13)$$

Aşağıda 20 değişkene kadar olan değerler bir tablo halinde verilmiştir. Tablo 4.4. YMÖA ile QMM' nun karmaşıklık karşılaştırılması

Tablo 4.4. YMÖA ile QM Metodunun karmaşıklık analizi

Quine-McCluskey metodu					YMÖA
Değ. Say.	Toplam Geçici Sonuçlar		Boş Olmayan Geçici Sonuçlar		Geçici Sonuç. Sayısı
	Asimptotik Değer (Wt)	$0(2^n)$ Karmaş.	Asimptotik (Rt)	$0(2^n)$ Karmaş.	
1	1	$0,5*2^1$	1	$0,50*2^1$	1
2	4	$1,0*2^2$	4	$1,00*2^2$	2
3	15	$1,9*2^3$	12	$1,50*2^3$	3
4	56	$3,5*2^4$	32	$2,00*2^4$	4
5	210	$6,6*2^5$	80	$2,50*2^5$	5
6	792	$12,4*2^6$	187	$2,92*2^6$	6
7	3.003	$235*2^7$	414	$3,23*2^7$	7
8	11.440	$44,7*2^8$	893	$3,49*2^8$	8
9	43.758	$85,5*2^9$	1.930	$3,77*2^9$	9
10	167.960	$164,0*2^{10}$	4.246	$4,15*2^{10}$	10
11	646.646	$315,7*2^{11}$	9.516	$4,65*2^{11}$	11
12	2.496.144	$609,4*2^{12}$	21.542	$5,26*2^{12}$	12
13	9.657.700	$1.178,9*2^{13}$	48.764	$5,95*2^{13}$	13
14	37.442.160	$2.285,3*2^{14}$	109.581	$6,69*2^{14}$	14
15	145.422.675	$44379*2^{15}$	243.554	$7,43*2^{15}$	15
16	565.722.720	$8.632,2*2^{16}$	534.891	$8,16*2^{16}$	16
17	2.203.961.430	$16.814,9*2^{17}$	1.161.526	$8,86*2^{17}$	17
18	8.597.496.600	$32.796,8*2^{18}$	2.497.440	$9,53*2^{18}$	18
19	33.578.000.610	$64.045,0*2^{19}$	5.325.568	$10,16*2^{19}$	19
20	131.282.408.400	$125.200,7*2^{20}$	11.280.076	$10.76*2^{20}$	20

4.5 Metodların Karşılaştırılması

Geliştirilmiş olan Yakın Minimali Örtme Algoritması (YMÖA), ESPRESSO-II algoritması ile karşılaştırılmıştır. Karşılaştırma kriteri olarak üç ana durum belirlenmiştir. Bunlar:

- Algoritmaların çözüm sonucunda buldukları çarpım terimlerinin toplam ifadesinin sayısı (SOP Sayısı),
- Algoritmaların çözüme ulaşma süreleri,
- Algoritmaların çözüme ulaşırken kullandıkları bellek kapasitesi

Gerçekleştirilen bu karşılaştırmalar aşağıdaki Tablo 4.5 Tablo 4.6 ve Tablo 4.7' de verilmiştir.

- YMÖA algoritması C++ programlama dilinde kodlanmıştır.
- Espresso-II algoritması da C programlama dilinde kodlanmıştır.

Bütün algoritmalar aynı dosya formatını kullanmıştır. Yani YMÖA ve Espresso-II algoritmaları için aynı benchmarklar kullanılmıştır. Algoritmaları aynı şartlarda karşılaştırabilmek için Espresso-II algoritmasının belirlediği durumlar dikkate alınmıştır. Karşılaştırmalar tek çıkışlı fonksiyonlar kullanılarak yapılmıştır. Benchmarkların karşılaştırılması için tam tanımlanmamış ve tam tanımlanmış fonksiyonlar kullanılmıştır. Fonksiyonların tam tanımlanmış veya tam tanımlanmamış fonksiyonlar olduğunu belirtmek için Espresso-II algoritmasının. type seçeneği kullanılmıştır. Bu seçenekte (.type fdr) fonksiyonun durumunu belirlemektedir. Bu seçenekteki f doğru kümesi için, r yanış kümesi için ve d belirsizler kümesi için kullanılmaktadır.

Karşılaştırmaların gerçekleştirilmesini kolaylaştırmak için Visual Basic programlama dilinde ara yüz programı hazırlanmıştır.

Aşağıdaki tablolarda benchmarklara ait, giriş değişken sayısı, SON sayısı, SOF sayısı, SOP sayısı, algoritmaların sadeleştirme zamanları ve kullandıkları bellek kapasiteleri verilmiştir.

Tablo 4.5. Standart MCNC benchmarkları için SOP sayısı

Benchmarklar	Değişken sayısı	Aİ sayısı		$\frac{A_E}{A_N}$
		Esp. Zaman	YMÖA zaman	
Addm4	9	18	18	1
b11	12	12	13	0,923
br2	12	9	9	1
Life	9	16	19	0,842
EX5	8	14	14	1
ex51	9	25	25	1
EXPS	9	20	16	1,25
m2	10	11	11	1
max5	9	23	23	1
P3	8	18	21	0,857
prom1	9	22	21	1,047
z9sym	9	18	18	1
root	9	16	15	1,066
SY0	20	136	143	0,961
T10	10	125	130	0,960
test2	11	261	287	0,909
test3	10	133	135	0,985
T4	18	35	35	1

Performans ve sonuç kalitesini karşılaştırmak için standart MCNC Benchmarkları YMÖA algoritması ve ESPRESSO tarafından sadeleştirilmiştir.

Karşılaştırmalar Intel P4 2.26 Ghz işlemcili ve 256 MB RAM belleği olan standart bir kişisel bilgisayarda gerçekleştirilmiştir. Tablo 4.5’ de verilen on sekiz farklı tek-çıkışlı fonksiyon kullanılmıştır.

Fonksiyonlara ait olan değişken sayıları, SON sayıları ve SOF sayıları tablolarda verilmiştir. Çarpım terimlerinin toplamı ifadesi şeklinde verilen sonuçlar (SOP sayısı) açısından algoritmalar karşılaştırıldığında
Tablo 4.5' den elde edilen bilgiler şöyledir:

Yakın Minimali Örtme Algoritması ile Espresso-II karşılaştırıldığında; fonksiyonların %45' inde eşit sayıda SOP sayısına sahip oldukları görülmüştür. Bu algoritmalardan Espresso, fonksiyonların % 38,75' inde daha iyi sonuç bulurken YMÖA % 16,66' inde daha iyi sonuç bulmuştur. SOP sayılarının ortalama değerlerine göre YMÖA ile Espresso' yu karşılaştırdığımızda Espresso' nun daha iyi sonuç bulduğu fonksiyonlarda ortalama % 4,3 daha az SOP bulmuştur. Burada Espresso algoritmasının daha iyi sonuç bulduğu görülse de YMÖA ile Espresso algoritmalarının çözüme ulaşma yöntemleri farklıdır. YMÖA olabilecek ihtimal sonuçları bulurken Espresso algoritması kesin olan sonuçları bulmaya çalışmaktadır. Bu şartlarda dahi YMÖA' nın daha iyi sonuç bulduğu fonksiyonların olması bu algoritmanın güçlü ve geliştirilebilecek yönlerinin olduğunu göstermektedir.

Tablo 4.6. Standart MCNC Benchmarkları için çalışma zamanları

Benchmarklar	Değişken sayısı	Çalışma zamanı (milisaniye)		$\frac{Z_E}{Z_N}$
		Esp. Zaman	YMÖA zaman	
ADDM4	9	43,750	28,120	1,554
b11	12	60,937	23,435	2,599
br2	12	60,930	23,435	2,589
Life	9	64,065	25,002	2,562
ex5	8	43,750	26,562	1,647
ex51	9	25,000	43,750	0,571
Exps	9	62,500	25,122	2,499
m2	10	62,584	26,255	2,451
max5	9	64,065	26,562	2,411
P3	8	60,935	25,240	2,437
prom1	9	60,937	23,437	2,6
Z9sym	9	23,437	43,750	0,535
Root	9	62,500	23,437	2,666
sy0	20	10,625	25,000	4,249
t10	10	60,937	26,562	2,294
test2	11	15,312	51,562	2,969
test3	10	70,312	26,562	2,647
t4	11	60,937	25,000	2,437

Tablo 4.6' da görüldüğü gibi, bu benchmarkları YMÖA ve Espresso tarafından sadeleştirilmiştir. YMÖA ve Espresso' nun sadeleştirme işlemlerini yaparken ihtiyaç duydukları zaman açısından değerlendirilmesi Tablo 4.6' da gösterilmiştir. Bu değerlendirmeye göre;

Yakın Minimum Örtme Algoritması ile Espresso Algoritmasını karşılaştırıldığında YMÖA' sının sadeleştirme işlemlerini çok daha hızlı gerçekleştirdiği görülmektedir. Fonksiyonların % 88,8' sında YMÖA daha hızlı bir şekilde sadeleştirme yapıp sonuca ulaşmıştır. Bu iki algoritma açısından bakıldığında YMÖA' sı Espresso algoritmasına göre çok daha hızlıdır. Ortalama olarak YMÖA Espresso' ya göre 2,31 kat daha hızlı sadeleştirme yapmaktadır.

Tablo 4.7. Standart MCNC benchmarklar için bellek kullanım durumları

Benchmarklar	Değişken sayısı	Bellek Kullanımı (bayt)		$\frac{B_E}{B_N}$
		Esp. Bellek	YMÖA Bellek	
ADDM4	9	151552	274432	0,552
b11	12	282624	307200	0,92
BR2	12	262144	278528	0,941
Life	9	442368	479232	0,930
EX5	8	8192	90112	0,1
ex51	9	180224	237568	0,758
EXPS	9	294912	311296	0,947
m2	10	372736	409600	0,91
MAX5	9	577536	622592	0,927
P3	8	671744	708608	0,945
PROM1	9	745472	724992	1,028
Z9sym	9	778240	806912	0,964
root	9	851968	880640	0,967
SY0	20	917504	950272	0,965
T10	10	937984	970752	0,966
TEST2	11	1028096	1114112	0,922
TEST3	10	1257472	1265664	0,993
T4	11	1331200	1363968	0,975

Algoritmaların sadeleştirme yaparken kullandıkları bellek alanı bakımından değerlendirilmesi yapıldığında, Espresso' un YMÖA' na göre %5,5 fonksiyonda daha iyi olduğu görülmesine rağmen %94,5 fonksiyonda YMÖA daha az bellek alanı kullanmıştır.

5 SONUÇ VE ÖNERİLER

5.1 Sonuç

Bu tez çalışmasında anahtarlama fonksiyonlarını sadeleştirmek için iki tane yeni yöntem sunulmuştur. Bu yöntem Yakın-Minimali Örtme Algoritması (YMÖA) dır.

Sunulan yöntemde küp cebri işlemleri kullanılmaktadır. Sunulan algoritma küp cebrinin koordinatlı çıkarma, koordinatlı kesişim ve dönüşümlü yutma işlemleri kullanılmıştır. Bu işlemlerin gerçekleştirilmesi seri bir şekilde yapılmaktadır. Seri gerçekleşen bu işlemler çözüme ulaşma süresini artırmaktadır. Bu işlemlerden koordinatlı kesişim ve dönüşümlü yutma işlemleri temel bilgisayar işlemleri üzerinden paralel bir şekilde gerçekleştirilmiştir. Bu sayede algoritmaların daha hızlı bir şekilde çözüme ulaşmaları sağlanmıştır. Çünkü küp cebri işlemlerini gerçekleştirebilmek için yapılacak karşılaştırmaların hepsi bit bit yapılmaktadır. Temel bilgisayar işleri üzerinden gerçekleştirildiğinde ise sayıların karşılaştırılması yapılmıştır. Veya sonuçların elde edilmesinde Ve (And), Veya (Or), Değil (Not). Veya Değil (Exor) lojik işlemleri kullanılmıştır. Bu sayede bit bit karşılaştırma yapmaktan kaçınılmıştır.

Sunulan YMÖA da verilen fonksiyonun ON kümesi mintermlerinden bir tanesini rasgele seçilmekte ve bu mintermi kapsayan asal implikantlar (AI) oluşturulmaktadır. YMÖA büyük implikantı seçme işlemi kullanılarak esas asal implikantlar (EAI) bir bir seçilmektedir. Belirlenen asal implikant için eşit sayıda minterm örtülürse üretilmiş AI' lardan bir tanesi seçilmektedir. Bu işlemlerin yapılması ile fonksiyonun sadeleşmiş halini temsil edecek esas asal implikantlar belirlenmiş olur. Sunulan YMÖA önemli bir şekilde var olan metotlardan hızlı çalışmaktadır ve daha az bellek kapasitesine ihtiyaç duymaktadır. Çünkü minimum sayıda geçici sonuçlar üreterek işleme tabi tutmaktadır. Bu özellikler sunulan yönetimi özlü ve son derece verimli yapmaktadır.

Geliştirilen algoritma olan Yakın Minimali Örtme Metodunun asal implikantları oluşturan kısmının karmaşıklığı karşılaştırmalı olarak hesaplanmıştır. Çünkü bunların esas asal implikant belirleme ve örtme kısımları var olanlar ile aynıdır. Verilen algoritma C++ programlama dilinde kodlanmıştır. Karşılaştırması yapılacak olan Espresso programı da C

programlama dilinde kodlanmıştır. Sunulan algoritmada ve karşılaştırması yapılan Espresso programında aynı dosya yapısı kullanılmıştır. Programların kullanımını kolaylaştırmak için Visual Basic programlama dilinde ara yüz programı yazılmıştır. Karşılaştırmalarda tek çıkışlı fonksiyonlar kullanılmıştır. Karşılaştırması yapılan fonksiyonlar tam tanımlanmamış veya tam tanımlanmış fonksiyonlardır. Algoritmaların karşılaştırması üç duruma göre yapılmıştır. Bunlar, algoritmaların verilen fonksiyonları sadeleştirdikten sonra elde ettikleri çarpım terimlerinin toplamı (SOP) sayısına göre, algoritmaların sadeleştirme zamanları ve bellek kullanma durumlarıdır.

Performans ve sonuç kalitesini karşılaştırmak için on sekiz farklı tek-çıkışlı fonksiyon YMÖA ve ESPRESSO tarafından sadeleştirilmiştir. Fonksiyonlara ait olan değişken sayıları tablolarda verilmiştir. Çarpım terimlerinin toplamı ifadesi şeklinde verilen sonuçlar (SOP sayısı) açısından algoritmalar karşılaştırıldığında YMÖA ile Espresso algoritması sonuçlarında; fonksiyonların %75' inde eşit sayıda SOP sayısına sahip oldukları, Espresso programının fonksiyonların %18,75' inde daha iyi SOP sayısı bulduğu, YMÖA' nında fonksiyonların %6,25' inde daha iyi SOP sayısı bulduğu görülmüştür. Algoritmaların buldukları SOP sayılarına göre; Espresso' nun daha iyi sonuç bulduğu fonksiyonlarda ortalama %9,7 daha az SOP bulurken, YMÖA' nın daha iyi sonuç bulduğu fonksiyonlarda ortalama %45 daha az SOP bulunmuştur.

YMÖA algoritmasının ve Espresso' nun fonksiyonları sadeleştirme zamanlarına göre karşılaştırıldığında; YMÖA' sının sadeleştirme işlemlerini çok daha hızlı gerçekleştirdiği görülmektedir. Fonksiyonların %89,6' sında YMÖA daha hızlı bir şekilde sadeleştirme yapıp sonuca ulaşmıştır. %10,4' ünde ise algoritmaların sonuca ulaşma zamanları eşittir. Ortalama olarak YMÖA Espresso' ya göre 7,9 kat daha hızlı sadeleştirme yapmaktadır.

Algoritmaların kullandıkları bellek alanı bakımından karşılaştırıldığında, Espresso' nun YMÖA' na göre %16,7 fonksiyonda daha iyi Olduğu görülmesine rağmen %83,3 fonksiyonda YMÖA tarafından daha az bellek alanı kullanılmıştır. Algoritmaların daha az bellek alanı kullandıkları fonksiyonlardaki durumlarına bakıldığında ise YMÖA %13,2 daha az bellek alanı kullanırken Espresso %9 daha az bellek alanı kullanmıştır.

5.2 Öneriler

Bu yüksek lisans tez çalışmasında geliştirilen algoritma tek çıkışlı fonksiyonlara uygulanmıştır. Bu çalışmanın bir sonraki adımı olarak çok çıkışlı fonksiyonlar için bu algoritmalar geliştirilebilir. Bu algoritmaların çok çıkışlı fonksiyonlar için geliştirilmesi ile çok çıkışlı diğer algoritmalarla çözüme ulaşma zamanları, kullandıkları bellek alanı ve SOP sayıları durumlarına göre karşılaştırılabilirler. Tek çıkışlı fonksiyonlar ile çok çıkışlı fonksiyonların ortak noktaları araştırılarak bu algoritmaların verimlilikleri incelenebilir.

YMÖA verilen fonksiyonun SON kümesinden hedef mintermi seçerken rasgele seçim yapılarak bu mintermi kapsayacak asal implikantlar bulunmaktadır. Hedef mintermi seçerken izole edilmiş mintermler belirlenebilir ve daha sonra bunların arasından bir tanesi seçilebilir. Hedef mintermi seçmek için başka bir prosedür olarak komşuluk faktörleri dikkate alınarak geliştirilebilir. Bu prosedür için önce bütün mintermler için komşuluk faktörleri hesaplanır. Daha sonra en düşük komşuluk faktörüne sahip olan minterm seçilir. Bu işlem, sonucun kesin olması istenen durumlarda iyi sonuçlar vermesi beklenirken sadeleştirme zamanı açısından da kötü sonuçlar ortaya koyabilir.

Bu tez çalışmasında sunulan algoritmalarda küp cebri işlemleri kullanılmıştır. Bu işlemler seri bir şekilde gerçekleştirildiği için bunlardan koordinatlı kesişim ve dönüşümlü yutma işlemleri temel bilgisayar işleri üzerinden gerçekleştirilerek paralel bir duruma getirilmiştir. Bu paralel işlemler sayesinde algoritmalar önemli bir şekilde hızlanmıştır.

6 KAYNAKLAR

Allahverdi N.M. and Kahramanlı S.S., 1995, Routing Algorithm in Hypercube with Application Cube Algebra.

Allahverdi N.M., Kahramanlı Ş.Ş., Erciyeş K., 2000, A Fault Tolerant Routing Algorithm Based On Cube Algebra For Hypercube Systems. Journal of Systems Architecture 46, pp. 201–205.

Atallah M. J., 1998, Algorithms and Theory of Computation Handbook, CRC Press.

Başçiftçi F., Kahramanlı Ş., Tütüncü K., Saraçoğlu R., 2003, Quine McCluskey Lojik Fonksiyonları Sadeleştirme Metodu, II. Ulusal Meslek Yüksekokulları Sempozyumu, 15-17 Ekim, s.365-378, Ege Üniversitesi, İzmir.

Bovet DP., Crescenzi P., 1994, Introduction to the Theory of Complexity. Prentice Hall, Eaglewood Cliffs, New Jersey.

Brayton, R., Hachtel, G.D.. Hemachandra, L.. Newton, A.R. and Sangiovanni Vincentelli, A.L., 1982, A Comparison Of Logic Minimization Strategies Using ESPRESSO. An APL Program Package For Partioned Logic Minimization. Proc. Int. Symp. On Circ. And Sys., pp: 43-49, Rome, May.

Brayton, R.K., Hachtel, G.D., McMullen, C., Sangiovanni-Vincentelli, A.L., 1984, Logic Minimization Algorithms For VLSI Synthesis. ISBN 0–89838–164–9, Kluwer Academic Publishers.

Brayton R.K., McGeer P.C., Sanghavi J., Sangiovanni-Vincentelli, A.L., 1993, A New Exact Minimizer for Two-Level Logic Synthesis, Kluwer Acadernic Publishers, pp: 1-31.

Bryant, R.E., 1986, Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. On Computer, 35 pp: 677–691, Aug.

Bryant R.E., 1995, Binary Decision Diagrams and Bey. En. Tech. for. Ver. The Proc. Int. Conf. On CAD, pp: 236–243, Nov.

Chai, L., 2000, ESOP Circuit Minimization Based On The Function On-Set. Master of Sciences, Mississippi State University.

Coudert O., Madre J. C., 1993, Towards a Symbolic Logic Minimization Algorithms. Proc. VLSI Design, Jan.

Coudert O., 1994, Two-Level Logic Minimization: An Overview, Integration, the VLSI Journal, 17–2, pp: 97–140, Oct.

Çelikağ M.., 1989, An implementation and Assessment of Some of the Boolean Function Minimization Methods. Master Thesis, Middle East Technical University.

Çırpan H.A., 1992, Lojik Fonksiyonların Bilgisayarla Basitleştirilmesi İçin Algoritmalar. Yüksek Lisans Tezi, Fen Bilimleri Enstitüsü, İstanbul Üniversitesi.

Çölkesen R., 2002, Veri Yapıları ve Algoritmalar. Papatya Yayıncılık, Mayıs 2002, ISBN: 975–6797–23–1

Dagenais M.R., Agarwal V.K., Rumin N.C., 1986, McBOOLE: A New Procedure for Exact Logic Minimization. IEEE Transactions On Computer Aided Design, Vol. CAD, No:1, Jan.

Dietmeyer D.L., 1979, Logic Design of Digital Systems. Boston, Bacon.

Fiser P., Hlavıcka J., 2003, BOOM - A Heuristic Boolean Miniinizer. Joumal of Computing and Informatics, pp: 1001–1033 jun.

Gurunath B., Biswas N.N., 1989, An Algorithm for Multiple Output Minimization. IEEE Transactions On CAD, Vol. 8, No:9, Sep.

Güneş S., 2000, Hiperküp Paralel İşlem Sisteminde Arızaya Toleranslı Veri İletimi Yöntemlerinin Analizi Ve Simülasyonu. Doktora Tezi, Selçuk Üni. Fen Bilimleri Enstitüsü.

Hong, S.J., Cam, R.G. and Ostapko, D.L., 1974, MINI: A Heuristic Approach For Logic Minimization. IBM J. of Res. and Dev., Vol..18, pp: 443-458, Sep.

Jacob J., Mishehenko A., 2001, Unate Decomposition of Boolean functions. Proc. IWLS, pp: 66–71.

Johnson baugh R., Schaefer M., 2004, Algorithms. Pearson Prentice Hall.

Kahramanlı S.S. and Allahverdi N.M., 1993, Compact Method of Minimization of Boolean Functions with Multiple Variables. Proc. Inter. Symp. Application of Computers, Selçuk University, Konya, Turkey, 433–440.

Kahramanlı Ş., Allahverdi N., 1996, An Algebraic Approach to Transformations on Hypercube System. Mathematical and Computational Applications, pp: 50–59.

Kahramanlı, Ş., Özcan, M., 2002, Lojik Tasarımın Temelleri ve Uygulamaları. Atlas Yayın Dağıtım. İstanbul.

Kahramanlı Ş., Başçiftçi F., 2003. Boolean Functions Simplification Algorithm Of O(n) Complexity. Mathematical Computational Applications, Volume 8 Num: 3, pp:271-278. ISSN:1300-686X

Kahramanlı Ş., Başçiftçi F., Savran 1., 2005, O(n) Karmaşıklığında Anahtarlama Fonksiyonlarını Sadeleştirme Algoritması. 4. Uluslararası İleri Teknolojiler Sempozyumu, Selçuk Üniversitesi, Konya, 28–30, Eylül, s: 214–219

Karnaugh, M., 1953, A Map Method for Synthesis of Combinational Logic Circuits. Trans. Comm. And Electronics, Vol: 72

Kruse R.L., 1987, Data Structures And Program Design. Prentice Hall.

Lee, C.Y., 1959, Representation of Switching Circuits by Binary Decision Diagrams. Bell System Technical Joumal, pp: 985–999, June.

Lin B., Somenzi F., 1990, Minimization of Symbolic Relations. IEEE İni. Conf. on Computer Aided Design, pp: 88–91.

Malik S., Wang A.R., Brayton R.K., Sangiovanni-Vincentelli A., 1988, Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment. The Proc. Int. Conf. on CAD, pp: 6–9.

Malik A.A., Brayton R.K., Newton A.R., Sangiovanni-Vincentelli A., 1991, Reduced Offset for Two Level Multi-Valued Logic Minimization. IEEE Trans. on Computer-Aided Design, CAD, pp: 413–426.

Mano M. M., 1984, Digital Design, Prentice-Hall Int. Ed. Mano, M.M., 2002, Sayısal Tasarım. Literatür Yayıncılık, İstanbul.

McCluskey, E., 1956, Minimization of Boolean Functions. Beli System Technical Journal, Vol. 35, No.5, pp: 1417-l444.

McCluskey, E.J., 1965, Introduction To The Theory Of Switching Circuit. McGraw Hill,

McCluskey, E.J., 1986, Logic Design Principles with Emphasis on Testable Semicustom Circuits., Englewood Cliffs, New Jersey, Prentice-Hall.

McGeer P., Jagesh S., Robert Brayton, Alberto Sangiovanni Vincentelli, 1986, Espresso-Signature: A New Exact Minimizer for Logic Functions. University of California al Berkeley, CA 94720.

McGeer P., Sanghavi J, Brayton, R.K., Sangiovanni-Vincentelli, 1993, ESPRESSO SIGNATURE: A New Exact Minimizer for Logic Functions. IEEE Transactions on VLSI, Vol. 1, No. 4, pp: 432–440.

McGeer P., Sanghavi 3., Brayton R., Sangiovanni-Vincentelli A., 1993, ESPRESSO SİGNATURE: A new exact Minimizer for Logic Functions. Proc. DAC 93, pp. 618- 624.

Minato, S. 1992, Fast Generation of Irredundant Sum-Of-Product Forms From Binary Decision Diagrams. Proc. 92, pp: 64–73.

Mishchenco A., Sasao T., 2003, Large-Scale SOP Minimization Using Decomposition and Functional Properties. DAC, June 2–6, pp: 49–154.

Nadjafov E.M. and Kahramanov S.S., 1973, On the Synthesis of Multiple Output Switching Scheme. Scientific Notes of Azerbaijan Institute of Petroleum and Chemistry, Baku, Azerbaijan. Vol. IX, No 3 pp: 65–69.

Perkins S.R., Rhyne T., 1988, An Algorithm for Identifying and Selecting The Prime Implicants of a Multiple-Output Boolean Function. İEEE Transactions On Computer Aided Design, Vol. 7, No:1 1, Nov.

Pomper G. and Armstrong J.A., 1981, Representation of Multi Valued Functions Using the Direct Cover Method. IEEE Trans. Comput, pp. 674–679, Sept.

Quine W.L., 1952, The problem of Simplifying Truth Functions. Amerikan Mathematics Monthly, Vol. 59, pp: 521–531.

Quine, W.L., 1955, A Way of Simplifying Truth Functions. Amerikan Mathematics Monthly, Yol. 62, No. 9, pp: 627–631.

Roth J.P.,1 956, Algebraic Topological Methods for the Synthesis of Switching Systems in n-variables. The Ins. for Adv. Study, Princeton, New Jersey.

Roth, J.P., 1980, Computer Logic, Testing and Verification. Computer Sciences Press.
Rudell R.L. and Sangiovanni-Vincentelli A., 1987, Multiple-Valued Minimization for PLA Optimization. IEEE Trans. CAD. Vol. 6(5), pp: 727–750, Sep.

Rudell R.L. 1989, Logic Synthesis for VLSI Design. PhD. Thesis, M89/49.

Rudell R.L., 1993, Dynamic Variable Ordering for Binary Decision Diagrams. The Proccedings İnternational Conference on Computer-Aided Design, pp: 42- 47, Oct.

Sasao T., 1985, An Algorithm to Derive the Complement of a Binary Function With Multiple-Valued inputs. IEEE Trans. Comp. Vol. C- 34, No. 2, pp: 131–140, Feb.

Sasao T., Butler J.T., 2001, Worst and Best Irredundant Sum-of-Product Expressions. IEEE Transactions on Computers, Vol. 50(9), pp. 935–947.

Savoj. Malik A.A., Brayton R.K., 1989, Fast Two-Level Logic Minimizer for Multi-Level Logic Synthesis. IEEE Int. Conf. on Computer Aided Design, pp: 426–429.

Tirumalai P.P., Butler J.T., 1991, Minimization Algorithms for Multiple-Valued Programmable Logic Arrays. IEEE Transactions on Computers, Yol. 40(2), pp: 167–177.

Uçar 0., 1996, Lojik Devre Tasarımı Algoritmaları, İstanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Yüksek Lisans Tezi.

Umans C., 2001, The Minimum Equivalent DNF Problem and Shortest Implicants. Journal of Computer and System Sciences, 63, pp: 597–611.

7 Ek-A YMÖA ALGORİTMASININ PROGRAM KODLARI

```
/*************************************************************************
                              Standart Kütüphaneler
*************************************************************************/
#include<stdio. h>
#include<stdlib. h>
#include<STRING.H>
#include<MATH. H>
/*************************************************************************
                              Değişken Dosyası
*************************************************************************/
#include "DEGISKEN.CPP"
/*************************************************************************
                              Temel Fonksiyonlar Dosyası
*************************************************************************/
#include "T2FONK.CPP"
void GENISLETME()
{  unsigned f;
        Sofsimdiki=Sofkok;
        Q0kok=NULL;
        for(f=1;f<Yeleman;f++)
        {       Q0islenen =(struct sinif *)calloc(1,sizeof(struct sinif));
                F_GENISLETME();
                if (Q0kok==NULL)
                {  Q0kok = Q0islenen;
                        Q0simdiki = Q0islenen;
                }

                Q0simdiki->sonraki = Q0islenen;
                Q0simdiki = Q0simdiki->sonraki;
                Sofsimdiki=Sofsimdiki->sonraki;
        }
}
//----------------
void DEGISMELI_YUTMA()
{
        unsigned Cr, Cl, f, elenir=1;
        Q1kok =(struct sinif*)calloc(1,sizeof(struct sinif));
        Q1=Q1kok->sonraki;
        Q1simdiki=Q1kok;
        Q0islenen=Q0kok;
        for(f=1;f<Yeleman; f++)
        {  Q1=Q1kok->sonraki;
                Q1simdiki=Q1kok;
                while(Q1!=NULL)
                {
                        elenir=0;
                        if(F_DEGISMELI_YUTMA(Q1->R,Q1->L,Q0isl->R,Q0is->L)
                                ==0)
```

```
                        {       elenir=1;
                                break;
                        }
                        if(F_DEGISMELI_YUTMA(Q1->R,Q1->L,Q0isle->R,Q0islenen->L)
                                ==1)
                        {       if(Q1simdiki==Q1)
                                {       Q1->R=Q0islenen->R;
                                        Q1->L=Q0islenen->L;
                                        elenir=1;
                                        break;
                                }
                                Q1simdiki->sonraki=Q1->sonraki;
                                free(Q1);
                                Q1=Q1simdiki->sonraki;
                                continue;
                        }
                        Q1simdiki=Q1;
                        Q1=Q1->sonraki;
                }
                if( !elenir | (Q1kok->sonraki==NULL))
                {       Q1yeni=(struct sinif*) calloc(1,sizeof(struct sinif));
                        Q1yeni->R=Q0islenen->R;
                        Q1yeni->L=Q0islenen->L;
                        Q1simdiki->sonraki=Q1yeni;
                        Q1simdiki=Q1yeni;
                }
                Q0islenen=Q0islenen->sonraki;
}       }
unsigned KOORDINATLI_KESISME(unsigned Ar, unsigned Al, unsigned Br, unsigned Bl)
{
        unsigned Cr, Cl;
        Br = OZELVEYA(Br, sabit);
        Cr = OZELVEYA(Ar, Br);
        Bl = OZELVEYA(Bl, sabit);
        Cl = OZELVEYA(Al, Bl);
        Cr = OZELVEYA(Cr, sabit);
        Cl = OZELVEYA(Cl, sabit);
        return VE(Cr, Cl);
}
void elemanekle(unsigned R,unsigned L)
{       K1yeni=(struct sinif *)calloc(1,sizeof(struct sinif));
        K1son2->sonraki=K1yeni;
        K1son2=K1yeni;
        K1son2->R = R;
        K1son2->L = L;
}
void KOORDINATLI_CIKARMA()
{       unsigned sonuc, D, E;
        K1kok=(struct sinif*)calloc(1,sizeof(struct sinif));
        K1kok->R = K1kok->L=sabit;
```

```
	K1=K1gecici=K1son2=K1kok;
	Q1=Q1kok->sonraki;
	while(Q1!=NULL)
	{
		while(K1!=K1gecici->sonraki)
		{
			sonuc=KOORDINATLI_KESISME(K1->R,K1->L,Q1->R,Q1->L);
			if (sonuc!=0)
			{	elemanekle(K1->R,K1->L);
				Sil=K1;
				K1=K1->sonraki;
				free(Sil);
				continue;
			}
			D = VE(K1->R, K1->L);
			E = VE(Q1->R, Q1->L);
			D = OZELVEYA(D ,E);
			E =OZELVEYA(K1->R, K1->L);
			E=OZELVEYA(E, sabit);
			D = VE(D,E);

			if (D!=0)
			{	for (i=0; i < bituzunluk; i++ )
				{	E=(unsigned)pow(2, (double)i);
					if(E & D)
					if (VE((unsigned)pow(2, (double)i), Q1->L))
						elemanekle(K1->R, VE(~E,K1->L));
					else
						elemanekle(VE(~E,K1->R),K1->L);
				}
			}
			Sil=K1;
			K1=K1->sonraki;
			free(Sil);
		}
		Sil=Q1;
		Q1=Q1->sonraki;
		free(Sil);

		K1gecici=K1son2;
	}
}
void DOSYA_OKU(char *argv[])
{	unsigned tam,D;
	if (((kaynakdosya = fopen(argv[1], "r+b")) == NULL)
		||((SPIdosya = fopen(argv[2], "w+b")) == NULL))
	{	printf("\n...HATA... DOSYALARIN ACILMASINDA HATA OLUSTU...\n");
		exit(0);
	}
	kontrol=fscanf(kaynakdosya, "%s%s", kaynakbilgi, kaynakbilgi);
```

```
	bituzunluk=atoi(kaynakbilgi);
	fprintf(SPIdosya, "%s %s", kaynakbilgi, kaynakbilgi);
	Kontrol=fscanf(kaynakdosya, "%s%s", kaynakbilgi, kaynakbilgi);
	Deleman=atoi(kaynakbilgi);
	Kontrol=fscanf(kaynakdosya, "%s%s", kaynakbilgi, kaynakbilgi);
	Yeleman=atoi(kaynakbilgi);
	tam=sabit<<bituzunluk;

	Sonkok=(struct sinif *) calloc(1,sizeof(struct sinif));
	Sofkok=(struct sinif *) calloc(1,sizeof(struct sinif));

	Sonsimdiki=Sonkok;
	Sofsimdiki=Sofkok;

	Yeleman=0;
	while (!feof(kaynakdosya))
	{
		Kontrol=fscanf(kaynakdosya, "%s %s", kaynakbilgi, fonkdeger);
		İslenen=(struct sinif *) calloc(1,sizeof(struct sinif));
		if(fonkdeger[0] == '1')
		{
			Sonsimdiki->sonraki=islenen;
			Sonsimdiki =islenen;

			Sonsimdiki->R = VEYA(tam, atoi(kaynakbilgi));
			D = OZELVEYA(Sonsimdiki->R, sabit);
			Sonsimdiki->L = VEYA(tam, D);
		}
		else if(fonkdeger[0]=='0')
		{
			Sofsimdiki->sonraki=islenen;
			Sofsimdiki=islenen;

			Sofsimdiki->R=VEYA(tam, atoi(kaynakbilgi));
			D =OZELVEYA(Sofsimdiki->R, sabit);
			Sofsimdiki->L=VEYA(tam, D);
			Yeleman++;
		}
	}
	Sonkok=Sonkok->sonraki;
	Sofkok=Sofkok->sonraki;
}
void main(int arc, char *argv[])
{
	DOSYA_OKU(argv);
	while(1<2)
	{
		if (Sonkok==NULL)
		{	return;
		}
```

```
                Else
                {
                        GENISLETME();
                        DEGISMELI_YUTMA();
                        KOORDINATLI_CIKARMA();
                        F_BUYUK_ASAL_IMP();
                }
        }
}
/*************************************************************************
                                Değişken Dosyası
*************************************************************************/
FILE *kaynakdosya,*SPIdosya;
unsigned i, Sofadet;
int fsimdiki, bituzunluk, Deleman, Yeleman, kontrol, fonkbitti;
unsigned const sabit= 65535;
struct sinif
{       unsigned R,L,
        absorbesayisi;
        struct sinif *sonraki;
}
struct sinif
{       unsigned long R,L,
        absorbesayisi;
        struct sinif *sonraki;
}
struct sinif
{       unsigned char R,L,
        absorbesayisi;
        struct sinif *sonraki;
}
*K1kok=NULL,        *K1yeni=NULL,       *K1=NULL,
*K1gecici=NULL,     *K1simdiki=NULL,    *K1son2=NULL,
*Q0kok=NULL,        *Q0simdiki=NULL,    *Q0islenen=NULL,
*Q1kok=NULL,        *Q1gecici=NULL,     *Q1yeni=NULL,
*Q1simdiki=NULL,    *Q1islenen=NULL,    *Q1=NULL,
*K1islenen=NULL,    *Sonkok=NULL,       *Sofkok=NULL,
*Sofsimdiki=NULL,   *Sonkok2=NULL,      *Sonsimdiki=NULL,
*Sonsimdiki2=NULL,*islenen=NULL,       *Sil=NULL,
*islenen2=NULL,       *Boskok=NULL;
char Rtxt[50], Ltxt[50], kaynakbilgi[30], fonkdeger[30];
```

```
/*************************************************************************
                        Temel Fonksiyonlar Dosyası
```

```
**********************************************************************/
unsigned VE(unsigned a, unsigned b)
{       unsigned c;
        c= a & b;
        return c;
}
unsigned VEYA(unsigned a, unsigned b)
{       unsigned c;
        c= a | b;
        return c;

}
unsigned OZELVEYA(unsigned a, unsigned b)
{       unsigned c;
        c= a ^ b;
        return c;

}
void F_GENISLETME()
{
        unsigned D;
        D= OZELVEYA(Sonkok->L, Sofsimdiki->L);
        D= OZELVEYA(D, sabit);
        Q0islenen->L = VEYA(D,Sofsimdiki->L);
        Q0islenen->R = VEYA(D,Sofsimdiki->R);
}
int F_DEGISMELI_YUTMA(unsigned Ar, unsigned Al, unsigned Br, unsigned Bl)
{
        unsigned Cr, Cl;
        Cr=VE(Ar, Br);
        Cl=VE(Al, Bl);
        if((Cr==Br)&(Cl==Bl))
                return 0;
        if((Cr==Ar)&(Cl==Al))
                return 1;
        return 2;
}

int F_KOORDINATLI_KESISME(unsigned Ar, unsigned Al, unsigned Br, unsigned Bl)
{       unsigned Cr, Cl;
        Br = OZELVEYA(Br, sabit);
        Cr = OZELVEYA(Ar, Br);
        Bl = OZELVEYA(Bl, sabit);
        Cl = OZELVEYA(Al, Bl);
        Cr = OZELVEYA(Cr, sabit);
        Cl = OZELVEYA(Cl, sabit);
        return VE(Cr, Cl);
}
void F_BUYUK_ASAL_IMP()
{  unsigned f, Cr, Cl;
```

```
	Sonsimdiki=Sonkok;
	K1kok=K1;
	while (Sonsimdiki!=NULL)
	{	K1=K1kok;
		while(K1!=NULL)
		{	Cr=VE(K1->R, Sonsimdiki->R);
			Cl=VE(K1->L, Sonsimdiki->L);
			if((Cr==Sonsimdiki->R)&(Cl==Sonsimdiki->L))
				K1->absorbesayisi++;
			K1 = K1->sonraki;
		}
		Sonsimdiki=Sonsimdiki->sonraki;
	}
	K1simdiki= K1= K1kok;
	while(K1simdiki!=NULL)
	{	if((K1simdiki->absorbesayisi >=K1->absorbesayisi))K1=K1simdiki;
		K1simdiki = K1simdiki->sonraki;
	}
	fprintf(SPIdosya,"%u%u",K1->R,K1->L);
	Sonkok2=NULL;
	İslenen=Sonkok;
	while (islenen!=NULL)
	{	Cr=VE(K1->R, islenen->R);
		Cl=VE(K1->L, islenen->L);
		if(!((Cr==islenen->R) & (Cl==islenen->L)))
		{	islenen2=(struct sinif*)calloc(1,sizeof(struct sinif) );
			if(Sonkok2==NULL)
			{	Sonkok2=islenen2;
				Sonsimdiki2=Sonkok2;
			}else
			{	Sonsimdiki2->sonraki=islenen2;
				Sonsimdiki2=islenen2;
			}

			Sonsimdiki2->R=islenen->R;
			Sonsimdiki2->L=islenen->L;
		}
		İslenen=islenen->sonraki;
	}
	while(Sonkok!=NULL)
	{ Sil=Sonkok; Sonkok=Sonkok->sonraki; free(Sil);}
	Sonkok=Sonkok2;
	Sonkok2=NULL;
	K1son2=K1=NULL;
}
```