T.C.

SELÇUK ÜNIVERSITESI

FEN BILIMLERI ENSTITÜSÜ

ANAHTARLAMA FONKSIYONLARI IÇIN YEREL BASITLESTIRME ALGORITMALARI

Fatih BASÇIFTÇI

DOKTORA TEZI

ELEKTRIK ELEKTRONIK MÜHENDISLIGI ANABILIM DALI

Konya, 2006

T.C.
SELÇUK ÜNIVERSITESI
FEN BILIMLERI ENSTITÜSÜ

ANAHTARLAMA FONKSIYONLARI IÇIN YEREL BASITLESTIRME ALGORITMALARI

Fatih BASÇIFTÇI

DOKTORA TEZI

ELEKTRIK ELEKTRONIK MÜHENDISLIGI ANABILIM DALI

Bu tez … / … / 2006 tarihinde asagidaki jüri tarafindan oybirligi/oyçoklugu ile kabul edilmistir.

.....	.....	.....
Prof. Dr.	**Prof. Dr.**	**Prof. Dr.**
Novruz ALLAHVERDI	**Inan GÜLER**	**Ahmet ARSLAN**
(Üye)	**(Üye)**	**(Üye)**

.....	….....
Doç. Dr.	**Yrd. Doç. Dr.**
Sirzat KAHRAMANLI	**Nihat YILMAZ**
(Danisman)	**(Üye)**

ÖZET

Doktora Tezi

ANAHTARLAMA FONKSIYONLARI IÇIN YEREL BASITLESTIRME ALGORITMALARI

Fatih BASÇIFTÇI
Selçuk Üniversitesi
Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisligi Anabilim Dali
Danisman : Doç. Dr. Sirzat KAHRAMANLI
2006, 138 sayfa
Jüri : Prof. Dr. Novruz ALLAHVERDI
Prof. Dr. Inan GÜLER
Prof. Dr. Ahmet ARSLAN
Doç. Dr. Sirzat KAHRAMANLI
Yrd. Doç. Dr. Nihat YILMAZ

Anahtarlama fonksiyonlarinin sadelestirilmesi tasarimcilara daha kisa zaman süresinde ve daha sade lojik devreler tasarlama imkani saglamaktadir. Sadelestirilmis olan bir fonksiyon daha az güç tüketimi, daha az hacim ve daha az maliyet gerektirir. Bu konu ile ilgili olarak tek ve çok çikisli fonksiyonlarin sadelestirilmesi için çesitli teknikler gelistirilmistir. Bu tekniklerin çogu iki ana asamada gerçeklestirilir. Birinci asamada, asal implikantlarin tümü belirlenir. Ikinci adimda fonksiyonu sadelesmis olarak örtecek, esas asal implikantlar kümesi belirlenir. Anahtarlama fonksiyonlarini sadelestirecek algoritmalarin tümü $O(2^n)$ karmasikligina sahiptirler. Arastirmalar göstermistir ki n'in çok yüksek degerlerinde esas asal implikantlarin tam kümesini belirleme yöntemi pratik olarak gerçeklestirilemez duruma gelmektedir. Bu yüzden bu doktora tezinde asal implikantlarin belli kistaslara cevap verecek alt kümeleri

olusturularak, dogrudan örtme (direct cover) prensibine dayanan bir minimumlastirma yöntemi gelistirilmistir.

Var olan dogrudan örtme metotlarinda verilen On-küpü içeren yeterli asal implikantlar kümesini bulmak için, bu küp her defasinda bir koordinat için genisletilir. Her genislemenin dogrulugu, $k < 2^n$ Off-küplerin hepsi ile genisletilen küp kesistirilerek kontrol edilir. Bir küpün genislemesinin polinominal karmasikliga sahip oldugu dikkate alindiginda, bu yaklasimin toplam karmasikligi $O(n^p)O(2^n)$ seklinde olmaktadir. Bu polinominal ve üssel (exponansiyel) karmasikligin çarpimidir. Verilen On-küpü içeren asal implikantlarin tam kümesini elde etmek için önerilen metot, bu On-küp tarafindan genisletilen Off-küpleri kullanir. Bu islemin karmasikligi, yaklasik olarak bir koordinat için bir On-küpün genisletilme karmasikligina esdegerdir. Bundan dolayi, verilen On-küpü içeren asal implikantlarin tam kümesinin hesaplama isleminin karmasikligi yaklasik olarak $O(n^p)$ kadar azaltilmis olur. Pratik olarak bu yaklasim bir defada islenecek olan asal implikant sayisini yüzlerce ve binlerce defa azaltmaktadir. Bu ise halen problem olan bellek kapasitesi darbogazini kolaylikla asma imkani saglamaktadir. Böylece en çok yirmi degisken siniri da asilmis olmaktadir.

Sunulan metot çesitli problemler üzerinde test edilmis ve 48 adet standart MCNC bencmarklari kullanilarak, dünyaca örnek olarak kabul edilmis olan ESPRESSO programi ile karsilastirilmistir. Bu karsilastirmalar sonucunda, fonksiyonlarin %89,6'sinda YMÖA, %10,4'ünde ise esit hizda sadelestirme yapmislardir. KMÖA'sinda ise fonksiyonlarin %12,5'inde esit zamanda, %16,7'sinde Espresso, %70,8'inde ise KMÖA daha hizli sadelestirme yapmistir. Kullandiklari bellek alani bakimindan degerlendirilmesi yapildiginda, ESPRESSO'nun YMÖA'na göre %16,7 fonksiyonda daha iyi oldugu görülmesine ragmen %83,3 fonksiyonda YMÖA daha az bellek alani kullanmistir. KMÖA'sinda ise ESPRESSO %8,3 fonksiyonda iyi olmasina karsin %91,7 fonksiyonda KMÖA daha az bellek alani kullanmistir.

Anahtar Kelimeler – Boole fonksiyonu, sadelestirme, minimumlastirma, Boole ifadesi, asal implikant, küp cebri, örtme algoritmasi, algoritmalarin karmasikligi, Off-küme tabanli minimumlastirma, dogrudan örtme prensibi.

ABSTRACT

PhD Thesis

LOCAL SIMPLIFICATION ALGORITHMS FOR SWITCHING FUNCTIONS

Fatih BASÇIFTÇI

Selçuk University

Graduate School of Natural and Applied Sciences

Department of Electrical-Electronics Engineering

Supervisor : Assoc. Prof. Dr. Sirzat KAHRAMANLI

2006, 138 Pages

Jury : Prof. Dr. Novruz ALLAHVERDI

Prof. Dr. Inan GÜLER

Prof. Dr. Ahmet ARSLAN

Assoc. Prof. Dr. Sirzat KAHRAMANLI

Asist. Prof. Dr. Nihat YILMAZ

The minimization of Boolean functions allows designers to make use of fewer components, thus reducing the cost of particular system. 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 Phd thesis, subsets of prime implicants that can prove direct cover principle which based on determineted criters use for mimimization method. To find the sufficient set of prime implicants including given On-cube on the existing direct-cover minimization methods, this

cube is expanded for one coordinate at a time. The correctness of each expanding is controlled by the way in which the cube being expanded is intersected with all of $k<2^n$ Off-cubes. If to take into consideration that the expanding of one cube has a polynomial complexity then the total complexity of this approach can be expressed as $O(n^p)O(2^n)$ that is the product of polynomial and exponential complexities. To obtain the complete set of prime implicants including given On-cube, the proposed method uses Off-cubes expanded by this On-cube. The complexity of this operation is approximately equivalent to a complexity of an intersection of one On-cube expanded by existing methods for one coordinate. Therefore, the complexity of the process of calculating of the complete set of prime implicants including given On-cube is reduced approximately to $O(n^p)$ times. This presently obtains solving problem of memory capacity that seems as a bottle neck In this way, number of maximum variables limit that is 20, could be exceeded.

The method has been tested on several different kinds of problems and on standard MCNC benchmarks results of which were compared with ESPRESSO.

Keywords – Boole function, simplification, minimization, Boole expression, prime implicant, cube algebra, covering algorithm, complexity of algorithms, Off-set based minimization, direct-cover principle.

TESEKKÜR

Doktora tez çalismalarim esnasinda bana yol gösterip her türlü yardimini esirgemeyen tez danismanim degerli hocam Doç. Dr. Sirzat KAHRAMANLI' ya, Elektronik ve Bilgisayar Egitimi Bölüm Baskani ve tez izleme komitesi üyesi sayin Prof. Dr. Novruz ALLAHVERDI' ye, tez izleme komitesi üyesi saygideger hocamiz Yrd. Doç. Dr. Abdullah ÜRKMEZ' e, programin yazilimi esnasinda zamanindan feragat edip gösterdigi yardimlardan dolayi degerli arkadasim Ars. Gör. Ridvan SARAÇOGLU' na, yetismemde emegi geçen tüm hocalarima, maddi ve manevi yardimlarini hiçbir zaman esirgemeyen ve daima beni tesvik eden anneme, babama ve esime tesekkür ederim.

IÇINDEKILER

ÖZET iii
ABSTRACT v
TESEKKÜR vii
IÇINDEKILER viii
SIMGELER x
KISALTMALAR xii
1. GIRIS 1
1.1. Lojik Minimumlastirma Problemlerinin Bugünkü Durumu 5
1.2. Tezin Amaci ve Önemi 7
1.3. Materyal ve Metot 8
1.4. Lojik Minimumluk 8
1.5. Kaynak Arastirmasi 9
2. BOOLE FONKSIYONLARINI MINIMUMLASTIRMA METOTLARININ ÖZETI 12
2.1. Fonksiyon Tanimlari 12
2.2. Karnaugh Haritasi Metodu 13
2.2.1. KH Metodunun uygulama durumlari 13
2.3. Asal Implikantlara Dayanan Minimumlastirma Yöntemleri 14
2.3.1. Quine McCluskey Metodu 14
2.3.1.1. Asal implikantlarin bulunmasi 14
2.3.1.2. Minimum AI kümesinin seçilmesi 15
2.3.1.3. Periyodik AIT 17
2.3.1.4. QMM uygulama durumlari 17
2.3.2. ESPRESSO-II Algoritmasi 18
2.3.2.1. ESPRESSO-II programi ve dosya formati 21
2.4. Ikili Karar Diyagrami Kullanarak Minimumlastirma Algoritmasi 23
2.4.1. BDD siralama ve sadelestirme 24
3. ANAHTARLAMA FONKSIYONLARI IÇIN YEREL BASITLESTIRME ALGORITMALARI 26

3.1. Isaretler gösterimi 27
3.2 Küp Cebri' nin Lojik Minimumlastirma Amaci ile Uygulanmasi 28
3.2.1. Küp cebrinin elemanlari ve uygulama biçimi.....29
3.2.2. Küp cebrinin islemleri.....31
3.2.2.1. Koordinatli çarpma islemi (☆ islemi) 31
3.2.2.2. Koordinatli çikarma islemi (# islemi) 35
3.2.2.3. Koordinatli kesisme islemi (Ç islemi) 39
3.2.2.4. Dönüsümlü yutma islemi (? islemi) 40
3.3. KCI Temel Bilgisayar Islemleri Üzerinden Gerçeklestirilmesi.....42
3.4. Asal Implikantlarin Yerel Sekilde Belirlenmesi.....50
3.4.1. Yakin-Minimum Örtme Algoritmasi 51
3.4.2. Kesin-Minimum Örtme Algoritmasi.....56
3.4.2.1. EAI seçme kurallari 57
3.4.2.2. Rasgele minterm seçme prensibi 58
3.4.2.3. Komsuluk faktörlerine dayali minterm siralama 61
3.4.2.4. Yeni KDO kurali ve EAI' larin tam belirlenmesini
saglamak için HM' lerin islemesini erteleme yöntemi.....64
3.4.2.5. On-Küme kapsama ve EHM' lerin yeniden baslama prosedürü.....66
4. SUNULAN YÖNTEMIN MEVCUT YÖNTEMLER ILE KARMASIKLIGA GÖRE KARSILASTIRILMASI 83
4.1. Karmasiklik (Complexity) 83
4.2. Algoritmalarin Karmasiklik Degerlendirmesi.....86
4.3. Yöntemlerin Karsilastirilmasi.....90
5. SONUÇ VE ÖNERILER.....101
5.1. Sonuç.....101
5.2. Öneriler 105
6. KAYNAKLAR 106
7. EKLER.....121
EK A.....114
EK B 121
EK C.....128

SIMGELER

m_i	Minterm (Standart Çarpim)
M_i	Maxterm (Standart Toplam)
v	Nonterminal (dügüm)
lo(v)	Degiskenin 0 degerine atandigi duruma karsilik gelir
hi(v)	Degiskenin 1 degerine atandigi duruma karsilik gelir
*	Belirsiz ya da keyfi deger (don't care)
☆	Koordinatli çarpma (coordinate product, star product)
#	Koordinatli çikarma (coordinate subtraction, sharp product)
∩	Koordinatli kesisme (coordinate intersection)
?	Dönüsümlü yutma islemi (commutative absorption operation)
{0,1,x}	Boolean degiskenin tanimlanma uzayi
x	esas olmayan koordinat veya degisken
{0,1,d}	Boolean fonksiyon tanimlama uzayi
d	fonksiyonun belirlenmemis degeri
n	Fonksiyonun bagli oldugu degisken sayisi
↔	Ancak ve ancak baglantisi
∪	Birlesme (uniting) islemi
S_{ON}	On mintermlerin kümesi
S_{ONi}	Örtülmüs On-kümesinin geçerli durumu
S_{OFF}	Off mintermlerin kümesi
S_{DC}	Fonksiyonun belirlenmemis oldugu mintermlerin kümesi
X	Örtülmek için seçilen On-minterm

$AI_i(x)$	X mintermini kapsayan asal implikant
$S_{AI}(x)$	X minterminin kapsadigi tüm asal implikantlarin kümesi
EAI(x)	X mintermin esas asal implikanti
S_{EAI}	Esas asal implikantlarin kümesi
S_{SW}	Durum sözlerinin kümesi
|S|	S kümesinin kardinalligi
DEF_i	Dahil Etme Fonksiyonu
L_{im}	DEF_i ' nin bir terimi
N(x)	HM X' in (Hedef Minterm X) komsularinin sayisi
F	On-kümesi (Espresso algoritmasi)
R	Off-kümesi (Espresso algoritmasi)
D	Belirsizler kümesi (Espresso algoritmasi)
k_i	küpün koordinat ekseni
$\tilde{k}_i$	k_i koordinat ekseni üzerindeki bir deger
O(g(n))	Karmasiklik ifadesi
L	Fonksiyon için gerekli olan mintermler
Q	Fonksiyon için yasak olan mintermler
D	Fonksiyon için gereksiz olan mintermler
W(X)	X kümesinin büyüklügü

KISALTMALAR

AI	Asal Implikant (Prime Implicant)
EAI	Esas Asal Implikant (Essential Prime Implicant)
IEAI	Ikincil Esas Asal Implikant (Secondary Essential Prime Implicant)
AIT	Asal Implikantlar Tablosu (Prime Implicant Table)
SAIT	Sadelestirilmis Asal Implikantlar Tablosu (Reduced AIT)
BDD	Ikili Karar Diyagrami (Binary Decision Diagram)
OBDD	Sirali BDD (Ordered BDD)
CAD	Bilgisayar Destekli Tasarim (Computer Aided Design)
KCI	Küp Cebri Islemleri
DEF	Dahil Etme Fonksiyonu (Inclusion Function)
DÖY	Dogrudan Örtme Yöntemi (Direct Covering Principle)
DST	Dallandirma ve Sinirlandirma Teknigi (Branch and Bound Technique)
HM	Hedef Minterm (Target Minterm)
AHM	Aktif HM (Active HM)
EHM	Ertelenmis HM (Postponed HM)
KDO	Kismi Düzenleme Operasyonu (Partial Ordering Operation)
KH	Karnaugh Haritasi
LSI	Büyük Ölçekli Devre (Large Scale Integrated)
VLSI	Çok Büyük Ölçekli Devre (Very Large Scale Integrated)
POS	Toplam Terimlerinin Çarpimi (Product of Sum)
SOP	Çarpim Terimlerinin Toplami (Sum of Product)
NPT	Örtüdeki çarpim terimlerinin sayisi

NLI	Örtünün giris kismindaki terimlerinin sayisi
NLO	Örtünün çikis kismindaki terimlerinin sayisi
PF	Petrick Fonksiyonu
PLA	Programlanabilir Lojik Diziler (Programmable Logic Arrays)
SFs	Anahtarlama Fonksiyonlari (Switching Functions)
QMM	Quine McCluskey Metodu
AV	Yutma Vektörü (Absorption Vector)
VI	Kesisme Vektörü (Vector Intersection)
VP	Çarpim Vektörü (Vector of Product)
VS	Çikarma Vektörü (Vector of Subtraction)
NP	Belirsiz Polinomal (Non deterministic Polinominal)
YMÖA	Yakin-Minimum Örtme Algoritmasi (Near-Minimal Cover Algortihm)
KMÖA	Kesin- Minimum Örtme Algoritmasi (Exact-Minimal Cover Algortihm)

1. GIRIS

Günümüzde Boole cebri olarak bilinen matematiksel sistem üzerine ilk çalismalar 1854 yilinda George Boole tarafindan baslatilmistir. 1904 yilinda Amerikali Matematikçi E. V. Hungtinton, Boole cebrine yeni aksiyomlar eklemistir. 1938 yilinda Shannon, Boole cebrini devre tasarimlarina uygulamistir. Bunun sonucunda Anahtarlama Cebri (Switching Algebra) adi altinda yeni bir bilim dali ortaya çikmistir (Brayton ve ark. 1984).

Dijital tasarimin basladigi 1950'li yillarda lojik kapilar (logic gates) pahali devre elemanlariydi. Bundan dolayi, verilen lojik fonksiyonu daha az sayida elektronik elemanla (kapilar ve diyot, direnç gibi kapilarin temel bilesenleri) gerçeklestirmek için yeni tekniklerin gelistirilmesinin önemi artmistir. Böylece o yillarda, lojik fonksiyonlarin sadelestirilmesi arastirmalari çok aktif bir alan haline gelmistir. Karnaugh ve Veitch haritalari, iki seviyeli lojik fonksiyonlarin (two-level logic functions) sadelestirilmesi için manüel olarak kullanilmistir. Bu yöntem ilk olarak 1952 yilinda Veitch tarafindan önerilmis ve 1953 yilinda Karnaugh tarafindan gelistirilmistir. Daha sonralari, Quine ve McCluskey (McCluskey 1965) tarafindan yeni bir teknik gelistirilmistir. Bu yöntem 1952 yilinda Quine tarafindan baslatilmis ve 1956 yilinda McCluskey tarafindan gelistirilmistir. Bu yöntem iki asamadan olusmaktadir:

1) Bütün asal implikantlarin (prime implicants - AI) üretilmesi,

2) Minimum örtmenin (covering) olusturulmasi.

Bütün asal implikantlarin üretilmesi çok etkili bir hale gelse de, Hong, Cain ve Ostapko tarafindan IBM'de gelistirilen MINI (Hong ve ark. 1974) programi, n degiskenli lojik fonksiyonun asal implikantlarinin sayisinin $3^n/n$ kadar büyük olabilecegini göstermistir. Buna ek olarak, ikinci adim, genellikle dallandirma ve sinirlandirma teknigi ile gerçeklestirilir. Bu teknik NP-karmasiklik (Non deterministic Polinominal-complete) problemleri sinifina ait olan minimum örtme probleminin çözümünü içermektedir. Bu ise etkili kesin bir algoritma bulma ümidini azaltir. Örnek olarak, minimum örtme algoritmasinin çalisma zamani, örtme problemindeki eleman sayilarindaki bir polinom ile sinirlandirilir. Örtüm

probleminin elemanlari sayisi lojik fonksiyonunun giris degiskenlerinin sayisiyla logaritmik olarak orantili olabileceginden, bu tekniklerin kullanimi orta ölçekteki problemler için bile pratik degildir (10-15 degisken) (Brayton ve ark. 1984).

1959 yilinda Ikili Karar Diyagramlari (Binary Decision Diagram - BDD) kavrami ilk olarak Lee (Lee 1959) tarafindan ileri sürülmüstür. 1978 yilinda, Akers (Akers 1978) ilk kez "ikili karar diyagrami" terimini benimsemistir. Akers ayni zamanda ilk kez BDD indirgeme kurallari kümesini sunmustur. Buna ragmen, BDD 1986 yilinda Bryant (Bryant ve ark. 1986) tarafindan etkin operatörler kümesi sunuluncaya kadar genis kullanim alanlari bulamamistir. 1986'dan sonra BDD'nin gelistirilmesi ve arastirilmasi çok büyük ilerlemelere ulasmistir (Ochi ve ark. 1991, Ochi ve ark. 1993). Bu konuda yapilan teknik makaleler, arastirma projeleri ve BDD paketleri, BDD'nin verimli kullanilmasina ve anlasilmasina katkida bulunmustur.

1960'larin sonunda ve 1970'lerin basinda lojik kapilarin fiyatlari düsmüstür. Bundan dolayi lojik sadelestirme, önceki yillardaki önemini kaybetmistir. 1970'lerin sonunda ise Büyük Ölçekli Devrelerin (Large Scale Integrated-LSI) düzenli yapilar halinde yapilmasi (örnegin PLA'lar), tasarimda harcanan zamani indirgediginden dolayi, lojik fonksiyonlarin uygulanmasinda istenilir bir hale gelmistir. Böylece, lojik tasarim ortaminda iki seviyeli lojik sadelestirme tekrar önemli bir araç haline gelmistir. Lojik fonksiyonlarin, sadelestirilmesinden elde edilen çarpimlarin toplami terimlerinin minimumlastirilmasi gerekli fiziksel alanin üzerinde dogrudan güçlü bir etkisi vardir. Çünkü her bir çarpim terimi, PLA'nin bir satiri olarak gerçeklestirilir. Çok Büyük Ölçekli Devre (Very Large Scale Integrated-VLSI) lojik tasarimi siklikla 30'dan daha fazla giris, çikis ve çarpimlarin toplami terimli lojik fonksiyonlari içerir. Bu durumda kesin sadelestirme pratik olmamaktadir. Bu gibi durumlarda gerekli olan en uygun sekle sokma (optimizasyon), farkli tecrübe yaklasimlari, probleme uygulamaktadir.

Bu konudaki yaklasimlardan bir tanesi klasik lojik sadelestirme tekniklerinin yapisini takip eder ve birinci olarak tüm asal implikantlari üretir. Bununla birlikte minimum bir örtü üretmek yerine yakin minimum bir örtü, tecrübelere dayanarak seçilir. Bu prosedür hala çok yüksek sayida asal çarpan üretme ihtimali içermektedir. Ikinci bir yaklasim eszamanli olarak örtü için implikantlari tanimlar ve seçmeye

ugrasir. Bu grupta birkaç tane algoritma ileri sürülmüstür (Hong ve ark. 1974, Roth 1980, Rhyne ve ark. 1977, Arevalo ve Bredeson 1978, Brown 1981).

Son zamanlarda, tecrübesel yaklasimlar, pratik PLA'larin tasariminda genis uygulama alanlari bulmustur. Bunlarin çok erken ve çok basarili olmasi, 1970'lerin ortasinda IBM'de MINI programinin gelistirilmesine sebep olmustur (Hong ve ark. 1974). Sonralari sezgisel sadelestirme programi PRESTO, D.Brown tarafindan tanitilmistir (Brown 1981). MINI'nin son zamanlardaki sürümü, SPAM olarak adlandirilmistir. Bu, büyük PLA'larin minimumlastirilmasina imkan verdi ve Kang (Kang ve VanCleemput 1981) tarafindan tamamlandi.

1981 yilinin yaz aylarinda ESPRESSO-I (Brayton ve ark. 1982) programi gelistirilmistir. ESPRESSO-I sirasiyla gelen isleri kontrol etmek için birçok anahtari olan tek bir programdir. Bir yil sonra 1982'nin yazinda ESPRESSO-II gelistirilmistir. Sadelestirilmis Off-küme ve tautology algoritmalarina dayanan iki yeni metot sunulmustur. Bu metotlarda verilen sonuçlar Espresso'nun sonuçlari kadar iyidir (Savoj ve ark. 1989). (Brayton ve Somenzi 1989) bu çalismalarinda Quine-McCluskey metoduna benzer bir yöntem sunmuslardir. (Lin ve Somenzi 1990) sembolik iliskilerin minimumlastirilmasiyla ilgilenmislerdir. BDD metoduna dayali yeni bir yöntem gelistirmislerdir ve örtme probleminin lineer zamanda yapilabilecegini göstermislerdir.

Çarpim terimlerinin toplamindaki sadelestirme ikili (binary) sistem içerisinde önemli bir yer tutmustur (Tirumalai ve Butler 1991). Son zamanlarda sunulan çarpim terimlerinin toplami seklinde sadelestirme yapan algoritmalarin birçogu dogrudan örtme (direct cover) metodunu kullanmistir. Dogrudan örtme metodu üç adim halinde gerçeklestirilir (Tirumalai ve Butler 1991):

1) Mintermin seçilmesi,

2) Asal implikantlarin üretilmesi,

3) Esas asal implikantin seçilmesi ve örtme.

Pomper ve Armstrong 1981'de dogrudan örtme metodunu sunmuslardir. Bu metot da rasgele minterm seçilerek SOP ifadesi seklinde yakin sadelestirmeyi (near-minimal) bulur. 1986'da Besslich bir baska dogrudan örtme metodunu sunmustur (Besslich 1986). Bu metot ilk olarak, örtmek için en çok izole edilmis (most isolated) mintermleri arar.

Asagida yaygin olarak kullanilan dogrudan örtme yöntemleri gösterilmistir.

1) Rasgele minterm/rasgele implikant (Random-minterm/Random-implicant) yöntemi

2) Pomper and Armstrong yöntemi (Pomper ve Armstrong 1981)

3) Besslich yöntemi (Besslich 1986)

4) Dueck and Miller yöntemi (Dueck ve Miller 1987)

5) Gold yöntemi

Bu metotlar dogrudan örtme yaklasimini temel almistir. Bu yaklasimlarda; ilk önce minterm seçilir. Sonra bu mintermi örten AI seçilir. Bu AI fonksiyonun On-kümesinden çikarilir, bu islemler söz konusu kümede minterm kalmayincaya kadar devam edilir (Tirumalai ve Butler 1991).

Rasgele minterm/rasgele implikant (Random-minterm/random-implicant) metodunda; minterm ve implikant rasgele seçilir. Her bir örtme islemi için, mevcut bulunan mintermler arasindan rasgele bir tanesi seçilir. Implikant seçme islemi de rasgele yapilmaktadir. Diger algoritmalar da oldugu gibi, seçilen implikant fonksiyonun On-kümesinden çikartilir ve On-kümesindeki minterm durumuna göre islem tekrarlanir. Bu metotta minterm ve implikant seçimi için belirlenmis bir karakteristik yoktur (Tirumalai ve Butler 1991).

Pomper ve Armstrong yönteminde; minterm bir önceki metotta oldugu gibi rasgele olarak seçilirken, implikant farkli bir sekilde seçilir. En fazla mintermi örten implikant Esas Asal Implikant (EAI) olarak seçilir. Eger bu tür implikant birden fazla ise ilk önce üretilen implikant EAI olarak seçilir. Bu islemler fonksiyon tamamen örtülünceye kadar tekrarlanir. Bu yöntemde minimum bir sadelestirme gerçeklestirilememektedir (Pomper ve Armstrong 1981).

Besslich yönteminde; lojik fonksiyonlar için her bir mintermin agirlik degeri vardir ve AI seçim dönüsümleri yaklasimiyla dogrudan örtme sunulmustur (Besslich 1986). Besslich yaklasiminin temel fikri en çok izole edilmis mintermleri ilk önce örtmek ve minterm örtme basina düsük maliyeti olan AI kullanmaktir. Böylece; EAI seçimi Pomper ve Armstrong metoduyla ayni yolla yapilmis olur.

Dueck ve Miller yaklasiminda; en düsük deger ile bütün mintermler için yalitim faktörü (isolation factor) hesaplanir. En yüksek yalitim faktörüne sahip minterm seçilir. Bu mintermi örtecek bütün AI'lar üretilir. Her bir AI için göreceli

kesme sayisi (relative break count) hesaplanir. En düsük göreceli kesme sayisi degerine sahip olan AI, EAI olarak seçilir (Dueck ve Miller 1987).

Gold yönteminde, sadelestirilmek istenen fonksiyona Pomper-Armstrong, Besslich ve Dueck-Miller algoritmalari uygulanir. Uygulama sonucunda bu algoritmalarin buldugu en iyi sonuç seçilir. Bütün fonksiyonlar için sürekli daha iyi olan tek bir algoritma yoktur (Tirumalai ve Butler 1991).

1.1. Lojik Minimumlastirma Problemlerinin Bugünkü Durumu

Kesin (exact) ve bulussal (heuristic) çarpimlarin toplami (sum of product-SOP) minimumlastirma bilgisayar destekli tasarim (computer aided design-CAD) alaninda çok iyi arastirilan problemlerden bir tanesidir (Mishchenco ve Sasao 2003). SOP minimumlastirma; PLA optimizasyonunda, çok seviyeli lojik sentezde (muti-level logic synthesis), durum sifrelemede, güç kestirimde, test üretmede ve diger alanlarda kullanilir (Mishchenco ve Sasao 2003). Kesin SOP minimumlastirma probleminin üssel dogasindan dolayi modern algoritmalar, (Brayton ve ark. 1984, Coudert ve Madre 1993, Coudert 1994, McGeer ve ark. 1993) minimum SOP kümesinde yüzlerce çarpim terimi oluncaya kadar sadelestirilmek istenen fonksiyonu isleyebilir. Bu arada pratik uygulamalarin ve CAD araçlarinin çogu bulussal minimumlastirmaya dayanir (Brayton ve ark. 1984, Ciesielski ve ark. 1989, Rudell ve Sangiovanni-Vincentelli 1987).

Bulussal algoritmalarin karmasikligi çarpimlarin sayisinda yaklasik olarak kareseldir (Mishchenco ve Sasao 2003). Bu algoritmalar kesin (exact) olanlardan fark edilebilecek kadar hizlidir fakat çok çarpimli fonksiyonlar için yavas olabilir (Mishchenco ve Sasao 2003).

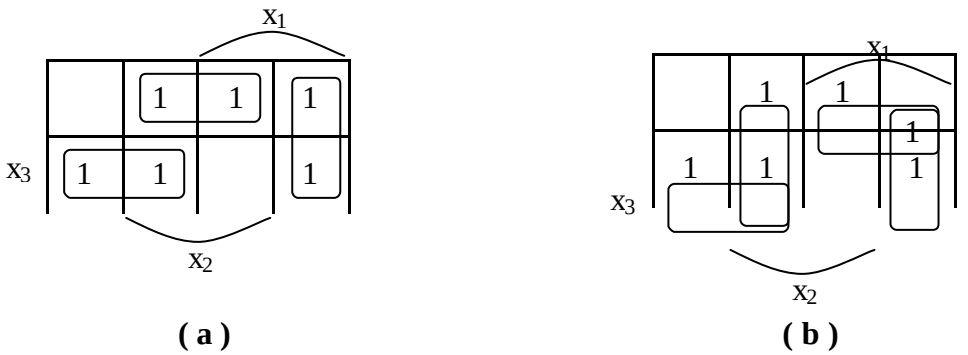
Bulussal SOP minimumlastirmayi hizlandirmak için çesitli yaklasimlar önerilmistir. Örnegin, off-kümesinin (Sasao 1985) hesaplamasi minimum SOP'da az sayida çarpimli fonksiyonlar için bile zaman tüketici olabildigi gözlenmistir (Mishchenco ve Sasao 2003). Bu yüzden sadelesmis off-kümesinin hesaplanmasi önerilmistir (Malik ve ark. 1991). Lojik sentez araçlari için optimizasyonda genisçe kullanilan baska hizlandirma sekli, bulussal minimumlastirmanin sadece bir döngüde

gerçeklestirilmesidir (Sentovich ve ark. 1992). Bu tür kisa yollarin bedeli çalisma zamani problemi hala dururken, daha düsük minimumlastirma kalitesidir. Bir çok benchmark için optimizasyon programlari bulussal SOP minimumlastirmanin uzun çalisma zamanindan dolayi sona ermez. (Mishchenco ve Sasao 2003).

Baska hizli bulussal SOP minimumlastirma algoritmalari BDD gösterimini kullanir (Minato 1992). Bu algoritma, sonuç kalitesinin kritik olmadigi durumlarda dikkat çekecek derecede iyi çalisir (Jacob ve Mishchenko 2001). Ancak (Sasao ve Butler 2001) da gösterilen bu algoritma (Minato 1992) minimum SOP lardan daha fazla çarpim içeren artiksiz (irredundant) SOP lar üretir (Mishchenco ve Sasao 2003). Bu yüzden birçok pratik problemler için uygun degildir.

Iki seviyeli lojik minimumlastirma lojik sentezin temel problemidir (Sasao ve Butler 2001). Genis fonksiyon kümeleri için kesin minimum SOP ifadeleri elde edecek algoritmalar olmasina ragmen (Coudert 1994, Dagenais ve ark. 1986), pratik sistemlerin çogunlugu bulussal lojik minimumlastirma algoritmalarini kullanir. Bu algoritmalar mutlak minimum olmayan gerekli SOP ifadelerini üretirler. Örnegin, PRESTO (Brown 1981, Svoboda ve White 1979), MINI (Hong ve ark. 1974), ESPRESSO (Brayton ve ark. 1984) ve digerler yöntemler (Dietmeyer 1978, Nguyen ve ark. 1987) minimum olmayan gerekli SOP ifadeleri üretirler.

Gerekli SOP, asal implikantlarin OR islemi ile baglanmis halidir. Öyle ki herhangi bir asal implikantin silinmesi fonksiyonu degistirir. Örnegin, Sekil 1.1.a ve Sekil 1.1.b de gösterilen $x_1\bar{x}_2 + x_2\bar{x}_3 + \bar{x}_1 x_3$ ve $x_1\bar{x}_2 + x_1\bar{x}_3 + \bar{x}_1 x_3 + \bar{x}_1 x_2$ ifadeleri ayni fonksiyon için gerekli SOP ifadeleridir. Bu ifadelerden birincisi minimum SOP ve ikincisi en kötü SOP dur.



(a) **(b)**

Sekil 1.1. Minimum SOP ve en kötü SOP ifadeleri için Karnaugh Haritasi

Sasao ve Butler (2001) fonksiyonlarin siniflarini, degisken sayisinin sinirsiz oldugu durumlarda en kötü SOP boyutunun minimum SOP boyutuna oraninin büyük oldugunu göstermislerdir. Sasao ve Butler (2001) verilen fonksiyon için bütün gerekli SOP ifadelerini üreten algoritmayi göstermislerdir.

1.2. Tezin Amaci ve Önemi

Bilgisayar devreleri ve programlarinin mümkün oldugu kadar basit ve etkili kilinmasi yolunda en etkin olan araçlardan biri lojik fonksiyonlarinin minimumlastirilmasidir. Halen çoklu miktarda minimumlastirma yöntemleri mevcuttur. Fakat bunlarin ürettikleri aralik sonuçlarinin sayisi, degisken sayisina göre üssel bir fonksiyonla belirlenir. Bu durumda mesela, 20 degiskenli fonksiyonlarin minimumlastirilmasi sirasinda meydana çikabilecek aralik sonuçlarinin sayisi bugün mevcut olan bilgisayarlarin bellek kapasitesini çok fazla asmaktadir. Pratikte 40'a kadar degisken degeri olan fonksiyonlarin minimumlastirilmasi ihtiyaci göz önüne alininca, mümkün oldugu kadar az sayida aralik sonuçlari üreten bir sadelestirme algoritmasinin elde edilmesine ihtiyaç oldugu süphesizdir. Bu tezde böyle bir algoritmanin meydana çikarilmasi hedeflenmis ve gerçeklestirilmistir.

Bu tezde gelistirilen algoritma sayesinde daha az lojik elemanlar kullanilarak yapilamayan programlanabilir lojik dizileri (PLA) kolaylikla tasarlanabilecek ve bu sayede büyük sayisal sistemlerin tasarlanmasinda donanim ve zaman kaybi büyük ölçüde önlenecektir.

Tez çalismasi yedi bölümden olusmustur.

Birinci bölümde; konunun tarihsel gelisimi anlatilarak, minimumlastirma problemlerinin bugünkü durumuna deginilmistir. Çalismanin amaci ve önemi açiklanmistir. Kaynak arastirmasina yer verilmistir.

Ikinci bölümde; Boole fonksiyonlari minimumlastirma metotlari özet seklinde anlatilmistir.

Üçüncü bölümde; gelistirilen algoritmalarda matematik araç olarak kullanilan küp cebri anlatilmistir. Bu bölümde ayni zamanda anahtarlama fonksiyonlari için

yerel basitlestirme algoritmalari için gelistirilen Yakin Minimum Örtme Algoritmasi (Near-Minimal Cover Algortihm) ve Kesin-Minimum Örtme Algoritmasi (Exact-Minimal Cover Algorithm) anlatilmistir. Gelistirilen algoritmalar örneklerle açiklanmistir. Matematik araç olarak kullanilan küp cebri islemlerinin standart bilgisayar islemleri üzerinden gerçeklestirilmesi gösterilmistir.

Dördüncü bölümde; karmasiklik degerlendirilmesi yapilmistir. Quine McCluskey Metodu ile Yakin Minimum Örtme algoritmasi karmasiklik yönünden karsilastirilmistir. Gelistirilen yöntemler ESPRESSO ile karsilastirilmis ve sonuçlari bu bölümde verilmistir.

Besinci bölümde; bu çalismadan elde edilen sonuçlara deginilerek ileride bu konuda çalismak isteyenler için bazi önerilerde bulunulmustur.

Altinci bölümde; bu çalismada yararlanilan kaynaklar verilmistir.

Yedinci bölümde; bazi algoritmalarla ilgili örnekler verilmistir. Gelistirilen algoritmalarin program kodlari verilmistir.

1.3. Materyal ve Metot

Çalismada mantik fonksiyonlarinin ifade biçimleri, sadelestirme yöntemleri, algoritmalari ve programlari kullanilmaktadir. Bu yolda elde edilmis son teorik sonuçlara dayanarak ve minterm yöntemiyle küp cebri yöntemleri bir arada kullanilarak daha etkin olan yeni bir yöntem meydana çikarilmaktadir. Gelistirilen algoritmalarin programlanmasi için C programlama dili kullanilmaktadir.

1.4. Lojik Minimumluk

Bir lojik fonksiyonun, birden fazla degisik ifadesi bulunabilir. Tüm olasi ifadeler arasindan minimum ifade bulunmaya çalisilir. Buradaki minimumluk, bir en iyilik ölçütüne göre tanimlanabilir (Çirpan 1992). Bu en iyilik ölçütü;

a) En az sayida lojik kapi gereksinimi,
b) Çarpim Terimlerinin Toplami (sum of product-SOP) biçiminde en az terim,
c) Toplam Terimlerinin Çarpimi (product of sum-POS) biçiminde en az terim,
d) Giris ile çikis arasindaki katman sayisinin minimumlastirilmasi ve dolayisiyla gecikme zamanini en aza indirebilmeyi saglamaktir.

Çarpim terimlerinin toplami biçimindeki bir fonksiyon, mantiksal degeri degistirilmeden hiçbir teriminin çikartilamayacagi biçimde ise, indirgenemezlik özelligine sahiptir. Genelde indirgenemezlik ve minimumluk birbirlerini içermez ya da gerektirmezler. Sonuç olarak her minimum fonksiyon indirgenemezdir. Fakat her indirgenemez fonksiyon, minimum fonksiyon degildir (Çirpan 1992).

1.5. Kaynak Arastirmasi

Akers S.B. (1978), ilk kez "ikili karar diyagrami" terimini benimsemistir. Akers ayni zamanda ilk kez BDD indirgeme kurallari kümesini sunmustur.

Allahverdi N.M. ve Kahramanli S.S. (1995), Küp cebri elemanlari ve uygulama biçimlerini belirtmis ve küp cebri islemlerini göstermislerdir.

Bartlett K.A. ve ark. (1988), çok seviyeli lojik devrelerin minimumlastirilmasi için yeni yaklasimlara deginmislerdir.

Beckert B. ve ark. (1997), çok seviyeli lojik devrelerin minimumlastirilmasi için yeni yaklasimlara deginilmistir

Bernasconi A. ve ark. (2001), minimumlastirma problemlerine karsin çesitli algoritmalara deginmislerdir.

Bryton R.K. ve ark. (1984), ESPRESSO lojik minimumlastirma algoritmasi ile ilgili algoritmalari sunmuslardir.

Bryant R.E. (1986), BDD için etkin operatörler kümesini sunmustur. OBDD'yi indirgemek için dönüsüm kuralini kullanmistir.

Çelikag M. (1989), çesitli minimumlastirma algoritmalari incelenmistir. Bu algoritmalar birbirleri ile karsilastirilmis ve degerlendirme yapilmistir.

Çirpan H.A. (1992), minimumlastirma algoritmalarindan Quine McCluskey yöntemi ile Biswas yöntemi karsilastirilmistir.

Çölkesen R. (2002), karmasikligin (complexity) tanimini belirtmis ve çesitli gösterimlerini sunmustur.

Dagenais M.R. ve ark. (1986), çok çikisli fonksiyonlarin tam minimumlastirilmasi için gelistirilen yeni prosedüre deginilmistir. Gelistirilen prosedür McBoole olarak adlandirilmistir.

Dietmeyer D.L. (1979), küp cebrini anahtarlama fonksiyonlarinin ilk terimlerini (local prime implicants) bulmak için kullanilmistir. Daha sonra lojik fonksiyonlarin minimumlastirilmasi üzerinde kullanilmistir.

Fišer P. ve Hlavicka J. (2003), iki seviyeli Boole minimumlastirma (BOOM) için yeni bir algoritma gelistirmistir.

Gurunath B. ve Biswas N.N. (1989), çok çikisli lojik fonksiyonlarin minimumlastirilmasina deginmislerdir. Minimumlastirma islemi gerçeklestirilirken esas asal implikantlari belirlerken bütün asal implikantlari üretmeden belirlenmistir.

Johnsonbaugh R. ve Schaefer M.(2004), algoritmalarla ilgili bilgiler verilmistir. Karmasiklik degerlendirmesi için bilgi verilmistir.

Kahramanli S.S. ve Allahverdi N.M. (1993), çok degiskenli Boole fonksiyonlar için yeni bir minimumlastirma algoritmasi sunulmustur.

Karnaugh, M. (1953), birlesik lojik devrelerin sentezi için harita metodunu sunmustur.

Mano M. M. (1984), lojik devreler ve lojik fonksiyonlar ile ilgili bilgiler vermistir. Fonksiyonlari minimumlastirirken elde edilen aralik sonuçlarinin sayisini bulmak için gerekli olan formüller verilmistir.

McCluskey, E.J. (1956), Boole fonksiyonlari sadelestirmek için Quine tarafindan baslatilan metodu gelistirmis ve sunmustur.

McGeer P.C. ve ark. (1986), çok çikisli fonksiyonlarin tam minimumlastirilmasi için gelistirilen yeni prosedüre deginilmistir. Gelistirilen prosedür ESPRESSO-SIGNATURE olarak adlandirilmistir.

Miller R.E. (1965), anahtarlama teorileri ile ilgili bilgiler vermistir.

Nadjafov E.M ve Kahramanli S.S. (1973), küp cebrini anahtarlama fonksiyonlarina uyarlamislardir. Daha sonra lojik fonksiyonlarin minimumlastirilmasi üzerinde kullanilmistir.

Perkins S.R. ve Rhyne T. (1988), Boole fonksiyonlarinin çoklu çikislari için Asal Implikantlari belirleme ve seçme algoritmalarini sunmuslardir.

Rujipattanapong, S. (2001), çesitli minimumlastirma algoritmalarini incelemistir. Incelenen algoritmalar için yeni yöntem gelistirmistir.

Sasao ve Butler (2001) ve Mishchenco ve Sasao (2003), minimumlastirma problemlerinin bugünkü durumlari hakkinda açiklama yapmislardir.

Tirumalai P.P. ve Butler J.T. (1991), son zamanlarda sunulan toplam terimlerin çarpimi seklinde sadelestirme yapan algoritmalarin birçogu dogrudan örtme (direct cover) metodunu kullanmistir. Bu makalede çesitli dogrudan örtme metotlari açiklanmistir.

Uçar O. (1996), lojik devre tasarimlari için çesitli algoritmalari incelemis ve bu algoritmalardan yeni bir yöntem gelistirmeye çalismistir.

2. BOOLE FONKSIYONLARINI MINIMUMLASTIRMA METOTLARININ ÖZETI

2.1. Fonksiyon Tanimlari

Boole fonksiyonlarinda, fonksiyonun degisken sayisina göre sahip oldugu çikis durumlari degismektedir. *n* sayida degiskene sahip olan fonksiyon 2^n sayida mintermle iliskide olur. Bu iliskinin karakterine göre söz konusu mintermler asagidaki gibi çesitli gruplara bölünür (Kahramanli ve Özcan 2002) :

1) Fonksiyonun degerinin 1'e esit oldugu mintermler,
2) Fonksiyonun degerinin 0'a esit oldugu mintermler,
3) Fonksiyonun degerinin belirsiz oldugu mintermler.

Bu gruplara uygun olarak L (gerekli), Q (yasak), D (belirsiz) kümeleri olusturulur. Yalniz 1. ve 2. grup mintermlerle iliskili olan fonksiyonlar ***Tam Belirlenmis Fonksiyonlar*** olarak, bunlarin hepsi ile iliskili olan fonksiyonlar ise ***Tam Belirlenmemis Fonksiyonlar*** olarak tanimlanirlar. L, Q ve D kümelerinin ölçüsü W(L), W(Q) ve W(D) olarak gösterilirse, L, Q ve D kümeleri ile onlara bagli olan F fonksiyonu arasinda asagidaki deger iliskilerinin oldugu görülebilir (Kahramanli ve Özcan 2002).

- $W(L)= 2^n$. Bu durumda mintermlerin tümünde fonksiyonun degeri 1 oldugu için aslinda fonksiyon degil bir sabit (lojik 1) söz konusudur,
- $W(Q)= 2^n$. Bu durumda mintermlerin tümünde fonksiyonun degeri 0 oldugu için aslinda fonksiyon degil bir sabit (lojik 0) söz konusudur,
- $W(L) < 2^n$, $W(Q) < 2^n$, $W(D) = 0$; $W(L) + W(Q) = 2^n$. Bu durumda tam belirlenmis olan bir fonksiyon söz konusudur,
- $W(L) < 2^n$, $W(Q) < 2^n$, $W(D) < 2^n$; $W(L) + W(Q) + W(D) = 2^n$. Bu durumda tam belirlenmemis olan bir fonksiyon söz konusudur.

2.2. Karnaugh Haritasi Metodu

Her fonksiyonun dogruluk tablosu gösterimi tektir; ancak, cebirsel olarak ifade edildiginde degisik sekillerde verilebilir (Mano 2002). Boole fonksiyonlari, cebirsel yollarla sadelestirilebilirler. Fakat bu minimumlastirma yönteminin, sistematik kurallari olmadigindan kullanisli degildir. Harita metodu Boole fonksiyonlarinin minimizasyonuna yarayan en basit ve görsel bir yöntemdir. Bu yöntem dogruluk tablosunun sekillendirilmis bir biçimi veya Venn diyagramlarinin gelismis bir sekli olarak da görülebilir. Önce Veitch tarafindan ortaya atilip sonradan Karnaugh tarafindan gelistirilen bu metot "Veitch Diyagrami" veya "Karnaugh Haritasi - KH" adiyla bilinir. KH (Karnaugh Haritasi) metodu en çok 4 veya 5 degiskenli fonksiyonlarin minimumlastirilmasi için kullanilir ve temel olarak,

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

kuralina dayanir. Degisken sayisi n olan bir fonksiyon için düzenlenen KH 2^n tane hücreden olusur. KH metodu, aslinda bir fonksiyonun standart formda ifade edilebilecegi tüm sekilleri sunan görsel bir yöntemdir. KH'de her bir hücreye karsilik gelen mintermlerin yazilmasi yerine, onun varligini bildiren bir isaret konur. Hücreleri isaretleme yöntemi (Mano 2002, Kahramanli ve Özcan 2002, Karnaugh 1953, Veitch 1952) kaynaklarinda açiklanmistir.

2.2.1. KH Metodunun uygulama durumlari

Degisken sayisinin dört veya besi geçmedigi durumlar için KH metoduyla minimumlastirma uygun bir yöntem olabilir. Degisken sayisi arttikça, çok sayidaki hücre, uygun komsu hücre seçimini zorlastirir. KH metoduyla minimumlastirma kullanicinin belirli kaliplari görebilme yetenegine dayandigindan, aslinda bir deneme yanilma yöntemidir. Bu durum, KH metodunun en belirgin dezavantajidir. Ayrica bes veya alti degiskenli fonksiyonlar için, en uygun seçimin yapilmis oldugundan

emin olmak bir hayli zordur. Bu metodu, bilgisayar programlarina uyarlamak oldukça güçtür.

2.3. Asal Implikantlara Dayanan Minimumlastirma Yöntemleri

2.3.1. Quine Mc Cluskey metodu

Quine McCluskey Metodu (QMM), bir fonksiyonun minimum sayida SOP seklinde ifade edilmesini saglar. Bu algoritma iki asamada gerçeklestirilir (Mano 2002, McCluskey 1956, Rujipattanapong 2001, Coudert 1994, Quine W.V.O. 1952) :

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

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

Bu asamalar asagida açiklanmistir.

2.3.1.1. Asal implikantlarin bulunmasi

Verilen Boole fonksiyonun AI'larinin bulunmasi süreci, söz konusu fonksiyonun minterm listesinin düzenlenmesi ile baslanir. Mintermler, içerdikleri 1'lerin sayisina göre gruplara ayrilir. Bu gruplar, mintermlerin içerdikleri 1'lerin sayisina göre küçükten büyüge dogru siralanir. Bu yöntemle olusturulabilecek maksimum grup sayisi *m*, degiskenlerin sayisi *n*'den bir fazla olabilir. Örnegin, dört degiskenli bir fonksiyon için en çok bes grup olusturulabilir (Rujipattanapong 2001, Quine 1955).

(2.1) kurali kullanilarak, *i.* grubun her bir mintermi ile *(i+1).* grubun her bir mintermi arasinda yeni terimlerin elde edilip edilemeyecegine bakilir. Eger komsu grup mintermleri arasinda sadece bir bitlik farklilik varsa, bu farklilik gösteren bit elde edilecek olan çarpim teriminde tire (-) isareti ile gösterilir. Gruplar arasindaki

karsilastirma süreci *(m-1)* ve *m* çiftine kadar tekrarlanir. Çarpim terimlerinde *k* tane degiskeni eksik olan terimler yani *k* tane (-) isareti olanlar *k-küp* olarak adlandirilir. Bu tanima göre mintermler *0-küp* olarak adlandirilir (Mano 2002, McCluskey 1956, Rujipattanapong 2001, Çelikag 1989).

0-küp sütunundaki bütün komsu grup mintermlerin karsilastirilmasiyla 1-küp sütunu olusturulur. Ayni islemler 1-küp sütununa uygulanir ve buradan 2-küp sütunu olusturulur. Ayni islemler sütunlar arasinda birlesme yapilamayacak duruma gelinceye kadar tekrarlanir. Bu *k-küp* sütunlarin sonunda isaretlenmemis çarpim terimler, AI'lardir (Mano 2002, McCluskey 1956, McCluskey 1986, Rujipattanapong 2001, Çelikag 1989).

2.3.1.2. Minimum AI kümesinin seçilmesi

Esas Asal Implikantlar kümesinin bulunmasi: Minimum AI kümesi, minimum sayida esas ve ikincil esas AI (essential and secondary essential prime implicant, EAI, IEAI) kümesinden olusur. EAI'lar, AI'lardan seçilir. Eger fonksiyonun bütün mintermleri, EAI'lar tarafindan örtülmüyorsa, IEAI'larin seçilmesi gerekmektedir. Burada örtülmek, fonksiyonu olusturan bütün mintermlerin, minimum sayida AI'lar tarafindan kapsanmasi demektir. Üstünlük (dominance) ve denklik (equivalent) kurallari, AI'larin fazla olanlarini eleyerek, IEAI'lari bulmak için kullanilir (Mano 2002, McCluskey 1956, McCluskey 1986, Rujipattanapong 2001, Çelikag 1989).

AI'larin minimum kümesini bulmayi kolaylastirmak için Asal Implikantlar Tablosu (prime implicant table - AIT) kullanilir (Mano 2002, McCluskey 1956, McCluskey 1986, Rujipattanapong 2001, Çelikag 1989). AIT'de, AI'lar satirlara, mintermler de sütunlara yerlestirilir. Fonksiyonun minimum seklini olusturacak AI'lari belirlemek için önce EAI'lar seçilir. Eger bir minterm sadece bir AI tarafindan örtülüyorsa, bu AI, EAI'dir ve SOP kümesine dahil edilir. Çünkü bu mintermi örtecek baska bir AI yoktur. Bütün EAI'lar seçildikten sonra, bütün mintermler örtüldüyse minimum SOP kümesi olusturulmus demektir. Eger hala bazi örtülmeyen mintermler varsa, bu mintermleri örtecek olan AI'larin diger AI'lardan

seçilmesi gerekir. Bu yolla seçilecek olan her bir AI, ikincil esas asal implikant (secondary essential prime implicant - IEAI) olarak adlandirilir.

IEAI kümesinin bulunmasi: IEAI'lar, sadelestirilmis AIT'dan (Reduced AIT-SAIT) seçilir. SAIT'da önceden seçilmis EAI'lar ve örtülmüs mintermler bulunmaz (McCluskey 1956, Rujipattanapong 2001, Çelikag 1989, Rudell 1989, Quine 1955).

Satir üstünlügü (row dominance) kurali:

Tanim 1. AIT'da bulunan herhangi bir *i* ve *j* satirlari için, *j* satirinda bulunan "x" isaretlerinin tümü *i* satirinda da bulunuyorsa, bu iki satir birbirine esittir.

Tanim 2. AIT'da bulunan *i* ve *j* satirlari için, *j* satirinda bulunan bütün "x" isaretleri *i* satirinda da varsa ve *i* satirinda en az bir tane fazla "x" isareti varsa, *i* satiri *j* satirini kapsar denir.

Tanim 3. AI'nin maliyeti, çarpim terimindeki literal sayisi ile belirlenir. Çarpim teriminde daha fazla literali olan daha fazla maliyete sahiptir.

Yukaridaki tanima göre *i* satiri kapsayan satir, *j* satiri kapsanan satirdir. Kapsanan satir SAIT'den çikarilabilir. Eger iki satir birbirine esitse bu satirlardan maliyeti fazla olan satir çikarilir (Basçiftçi ve ark. 2003).

Sütun üstünlügü (column dominance) kurali:

Tanim 4. AIT'da bulunan *i* ve *j* sütunlari için, *i* sütununda bulunan "x" isaretleri *j* sütununda da bulunuyorsa, bu iki sütun birbirine esittir.

Tanim 5. AIT'da bulunan *i* ve *j* sütunlari için, *i* sütununda bulunan bütün "x" isaretleri *j* sütununda da varsa ve *i* sütununda en az bir tane fazla "x" isareti varsa, *i* sütunu *j* sütununu kapsar denir.

Yukaridaki tanima göre *i* sütunu kapsayan sütun, *j* sütunu kapsanan sütundur. Kapsanan sütun SAIT'den çikarilir. Eger iki sütun birbirine esitse bu sütunlardan maliyeti fazla olan sütun çikarilir (Basçiftçi ve ark. 2003).

Bütün sadelestirme kurallari uyguladiktan sonra AIT'de birden fazla minterm kalabilir. Bu tür tablolara *periyodik tablo (cyclic table)* denir. Periyodik problemler Dallandirma Metoduyla veya Petrick Fonksiyonuyla çözülebilir (Rujipattanapong 2001, Çelikag 1989, Rudell 1989).

2.3.1.3. Periyodik AIT

Dallandirma metodu: Problemin çözümü, periyodik AIT'den herhangi bir minterm sütunu seçilerek devam ettirilir. Bu durumda en az AI'a ait olan minterm sütununun seçilmesi tercih edilir. Eger böyle bir sütundaki AI'larin sayisi *M* ise uygun mintermi örtmek için *M* tane alternatif var demektir. Buna ragmen hangi seçimin minimum sonuç verecegi bilinmez. Bundan dolayi *M* tane farkli seçim denenir ve her biri farkli SOP üretir. *M* tane SOP çözümünden minimum çözüm seçilir (Rujipattanapong 2001, Hong ve ark. 1974).

Petrick fonksiyonu: Petrick Fonksiyonu (PF, p-fonksiyon) metodunda, periyodik tabloda bulunan her AI; A,B,C,... olarak harflendirilir. Bir minterm sütununda *L* tane AI varsa bu mintermi örtmek için *L* tane farkli AI var demektir. Bu mintermi örtmek için olasi AI'larin toplami *L* tanedir. Periyodik tablodaki *N* tane sütun (veya minterm) *N* tane toplam terimi üretir. AI fonksiyonu veya p-fonksiyon *N* tane POS ile tarif edilir. Her sütun için AI'lar toplanir ve diger sütunlarin AI'larinin lojik toplami ile lojik çarpilir. Çarpimlar sonra toplam olarak düzenlenir. Düzenleme yapildiktan sonra en az literale sahip olan bilesen veya bilesenler minimum ifadeyi olusturur (Rujipattanapong 2001, Çelikag 1989). Minimum ifade tek olabilecegi gibi birden fazla da olabilir.

2.3.1.4. QMM uygulama durumlari

KH metodunda, bes veya alti degiskenli fonksiyonlar için, en uygun seçimin yapilmis oldugundan emin olmak ve bu metodu, bilgisayar programlarina uyarlamak da bir hayli zordur. Bu zorluklara QMM çözüm getirir. Bu metot, adim adim uygulanarak fonksiyon için minimumlastirilmis ifadeyi standart bir biçimde elde eder. Bu metot, çok degiskenli fonksiyonlara uygulanabilir ve bilgisayarda programlamaya uygundur. Ancak, rutin ve monoton islemlerinden dolayi kullanimi oldukça sikicidir ve hata yapma olasiligi yüksektir. QMM çok girisli - çok çikisli fonksiyonlar için genisletilebilir. Pratik uygulamalarda, çok çikisli problemlerde

AI'larin sayisi çok fazladir. Bundan dolayi, bu metot çok fazla hafizaya gereksinim duyar (Chai 2000). Giris degiskeni sayisi fazla olursa, minterm sayisi fazla olacagindan, üretilen AI fazla olacaktir ve bu AI'larin depolanmasi için çok fazla hafizaya ihtiyaç duyacaktir (Chai 2000). Bundan dolayi bu metot, çok degiskenli problemler için uygun degildir. Bununla birlikte, giris degiskeni sayisi az olursa diger metotlara göre daha hizli olabilir.

2.3.2. ESPRESSO-II algoritmasi

ESPRESSO-II, *f* fonksiyonunun 1-kümesini, belirsizler kümesini ve 0-kümesini giris olarak alir (Brayton ve ark. 1984). Bu algoritma çikis olarak sadelestirilmis bir örtü verir. ESPRESSO-II optimala yakin çözümü bulmaktadir ve asagida verilen 3 sayiyi azaltmaya çalismaktadir (Brayton ve ark. 1984, McGeer ve ark. 1986, Brayton ve ark. 1993, McGeer ve ark. 1993, Sanghavi 1996).

1. NPT: örtüdeki çarpim terimlerinin sayisi.

2. NLI: örtünün giris kismindaki terimlerinin sayisi.

3. NLO: örtünün çikis kismindaki terimlerinin sayisi.

ESPRESSO-II *F = (NPT, NLI, NLO)* vektörünü kullanarak sadelestirme süresince F'nin bilesenlerini azaltmaya çalismaktadir (Brayton ve ark. 1984, McGeer ve ark. 1986). Bu isleme, son döngü sirasinda, bilesenlerin hiçbirisi degismediginde son verilir (Brayton ve ark. 1993, McGeer ve ark. 1993).

Sadelestirme islemine geçmeden önce sadelestirilecek olan fonksiyonlara UNWRAP (dagitma, açma) prosedürü uygulanir. Bu prosedür *k* tane fonksiyon tarafindan paylasilan bir küpü, her biri sadece bir fonksiyon tarafindan paylasilan *k* tane küp ile yer degistirir (Brayton ve ark. 1984, Uçar 1996). Her ne kadar bu sekilde optimaldan daha uzaklasilsa da böyle bir islem sonucunda sadelestirme islemi girise daha az bagimli olur ve EXPAND prosedüründe küplerin daha yararli bir sekilde hangi fonksiyon tarafindan paylasilacagi bulunabilir (Brayton ve ark. 1984). Bu sekilde *F* (on-set), *D* (don't care set) ve *R* (off-set) örtüleri elde edildikten sonra F vektörü hesaplanir. Bu vektörün bilesenlerinde bir azalma görülemeyinceye kadar EXPAND, IRREDUNDANT_COVER ve REDUCE prosedürleri çalistirilir. F'nin

bilesenlerinde azalma görülmediginde LAST_GASP prosedürü çagrilir. Eger F'nin bilesenlerinde azalma görülürse tekrar REDUCE prosedürü çagrilir. ESPRESSO-II sadelestirme algoritmasi 6 tane temel prosedürden olusur. Bunlar COMPLEMENT, EXPAND, ESSENTIAL_PRIMES, IRREDUNDANT_COVER, REDUCE, LAST_GASP'dir. Bunlara ek olarak yukaridaki 6 algoritmanin pek çogu önemli bir sekilde TAUTOLOGY algoritmasina dayanir. Bu algoritma ile elemanlari küpler olan bir kümenin, bir küpü örtüp örtmedigi belirlenir (Brayton ve ark. 1984).

COMPLEMENT Prosedürü: Bu kisimda birden çok fonksiyon için tümleyen alma yöntemi verilmistir. Bu yöntemde monoton fonksiyonun özelliklerinden yararlanilarak kendisini çagiran (recursive) bir prosedür ile bir fonksiyonun tümleyeni bulunur ve bu islem her çikis için tekrarlanir. EXPAND prosedürü, ESPRESSO-II içinde tümleyen alma prosedürünü kullanan tek ana prosedürdür. Teker teker fonksiyonlarin tümleyenlerini alma islemi, bazi çarpim terimlerini tekrar kullanacagindan daha fazla bellek kullanir. Complement prosedürü, verilen F ve D örtüleri için R örtüsünü hesaplar. Bu prosedür EXPAND prosedüründe asal bilesen seçiminde kullanilir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

Monoton Fonksiyon: Bir f fonksiyonunun x_j girisinin degeri 0'dan 1'e degistirilmesi ile çikisi da 0 iken 1 (1 iken 0) oluyorsa, f fonksiyonu x_j degiskenine göre monoton artandir (azalandir) denir. Bir fonksiyon bütün degiskenlerine göre monoton artan veya azalan ise bu fonksiyona *monoton fonksiyon* denir. (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

EXPAND Prosedürü: Genisletme isleminin amaci F örtüsünden mümkün oldugu kadar çok sayida küpün atilmasidir. Bunun için F örtüsünün küpleri teker teker belirli bir sira ile ele alinir ve ele alinan küp ile F örtüsünde bulunan maksimum sayida küp örtülmeye çalisilir. Daha sonra genisletme islemi ile elde edilen asal küpler örtüye dahil edilir. Bu küplerin F örtüsündeki kapsadiklari küpler örtüden çikarilir. EXPAND algoritmasinin sonucu genisletilen küplerin ele alinma sirasina baglidir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986, Kruse 1987).

ESSENTIAL_PRIMES Prosedürü: Burada çözülmesi gereken problem, verilen $F=(c^1,...,c^k)$ örtüsü için her bir c^i , f'nin bir asal küpü olmak üzere, verilen bir c^i asal küpü f'nin bir temel asal bileseni olup olmadiginin belirlenmesidir. Temel asal bilesenler f'nin bütün asal örtülerinde bulunmalidir. Bu nedenle EXPAND, REDUCE

ve IRREDUNDANT_COVER prosedürleri yürütülürken temel asal bilesenleri örtüden elemek, hesaplama zamanini azaltir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

IRREDUNDANT_COVER Prosedürü: ESPRESSO-II'nin EXPAND prosedürünün uygulamasi ile, *F* asal örtüsü elde edilir. Bu örtüde hiçbir küp digerini kapsamaz. Bununla birlikte *F*'nin minimal örtü oldugu kesin degildir. IRREDUNDANT_COVER prosedürü verilen *F* ve *D* için, *F*'nin bazi küplerinden olusan minimale yakin F_2 örtüsünü belirler. Bu prosedür ile F_2 Í *F* olan ve mümkün oldugu kadar az küpe sahip F_2 örtüsü elde edilmeye çalisilir. Bu prosedürden sonra bir minimal örtü elde edilir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

REDUCE Prosedürü: IRREDUNDANT_COVER prosedürü ile elde edilen örtüdeki küpleri teker teker ele alir. Her *c* Î *F* küpü için, *c* küpünün *(F-{c})+D* örtüsü tarafindan kapsanmayan mintermlerden olusan en küçük küp $\underline{c}$'yi bulur. Daha sonra *F* örtüsünde *c* küpü ile $\underline{c}$ küpünü degistirir. Yani *F=(F-{c})+* $\underline{c}$ olur. Bu sekilde elde edilen örtü EXPAND prosedürü ile daha çok yönde genisletilebilir. Ayrica *F* örtüsü bu islemle daha küçük küplerden olusur ve genisletilen küpler tarafindan kapsanma olasiligi artar. Bu prosedürün sonucu küplerin ele alinma sirasina baglidir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

LAST_GASP prosedürü: Bu algoritma sadelestirilecek olan örtüden birkaç küp daha çikarabilmek için kullanilir. LAST_GASP, degistirilmis bir REDUCE ve degistirilmis EXPAND prosedürlerini içerir. En son sadelestirilmeye çalisilan küpler en az sadelestirme sansina sahiptir. Bunun sebebi daha önce sadelestirilerek kisaltilan küpler nedeniyle örtü zaten az sayida minterm içermektedir. Sadelestirilecek küplerin kabaca seçimi, EXPAND islemi sonunda örtüdeki küp sayisinin azalacagini garanti etmemektedir. LAST_GASP prosedüründe her bir küp maksimum sekilde sadelestirilir. Daha sonra sadelestirilen küpler üzerinde EXPAND islemi uygulanir. REDUCE prosedürü ayni islemi yapmaktadir fakat bu prosedürde küpler belirli bir sira ile ele alinmaz. Her küp bagimsiz olarak ele alinarak REDUCE prosedürü ile yapilan islem tekrarlanir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

TAUTOLOGY Prosedürü: Bir fonksiyonun sabit-1 olup olmadiginin belirlenmesi için ESPRESSO-II tarafindan kullanilan temel bir islemdir. Bu islem IRREDUNDANT_COVER, REDUCE, ESSENTIAL_PRIMES ve LAST_GASP

prosedürlerinin temel bölümünü olusturur. Bu nedenle etkili bir TAUTOLOGY algoritmasi ESPRESSO-II'nin hizi için önemlidir (Brayton ve ark. 1984, Uçar 1996, McGeer ve ark. 1986).

Procedure ESPRESSO-II(*F,D*)
Begin
F? UNWRAP(*F*)
R? COMPLEMENT(*F*,*D*)
F1? F2? F3? F4? COST(*F*)
LOOP1 **:** (F,*F*)? EXPAND(*F*,*R*)
if (First-Pass) (F,*F*,*D*,e)? ESSENTIAL_PRIMES(*F*,*D*)
if (F=F1) go to OUT
F1? F
(F,*F*)? IRREDUNDANT_COVER(*F*,*D*)
if (F=F2) go to OUT
F2? F
LOOP2 **:** (F,F)? REDUCE(F,D)
if (F=F3) go to OUT
F3? F
go to LOOP1
OUT **:** if (F=F4) go to QUIT
(F,*F*)? LAST_GASP(*F*,*D*,*R*)
if (F=F) go to QUIT
F1? F2? F3? F4? F
go to LOOP2
QUIT **:** ***F***? *FÈ* ?
D? *D*-e
(F)? (*F*,*D*,*R*)
return(F,*F*)
End

2.3.2.1. ESPRESSO-II programi ve dosya formati

ESPRESSO verilen fonksiyonu çarpim terimlerinin toplami seklinde sadelestiren, çok seçenegi olan bir programdir. ESPRESSO programinin kullanim formati asagidaki gibidir:

C:\>espresso [seçenekler] [dosya]

ESPRESSO programinin kullandigi dosya formati asagida gösterilmistir. Programin tanidigi anahtar kelimeler belirtilmistir. [d] desimal bir sayiyi belirtir. [s] bir string ifadeyi belirtir.

```
.i 3
.o 1
000 0
001 1
010 0
011 1
100 -
101 0
110 1
111 -
.e
```

Burada;

.i [d] giris degiskeninin sayisini belirtir.

.o [d] çikis degiskeninin sayisini belirtir.

.e dosyanin bittigini gösterir.

Verilen bu seçenekler her dosyada olmasi gereken durumlardir.

ESPRESSO programinda kullanilan seçeneklerden çok kullanilanlar asagida açiklanmistir.

-Dexact: Exact minimumlastirma algoritmasi (çarpim terimlerinin minimum sayida olmasini garanti eder ve bulussal (heuristic) olarak literallerin sayisini minimumlastirir). Genellikle pahali olabilecek sonuçlar üretir.

-Dsignature: Küp tabanli kesin (exact) minimumlastirma algoritmasi (çarpim terimlerinin minimum sayida olmasini garanti eder ve bulussal olarak literallerin sayisini minimumlastirir). –Dexact seçenegine göre daha hizlidir ve –Dexact seçeneginin takildigi problemleri çözer (Brayton ve ark. 1993, McGeer ve ark. 1993).

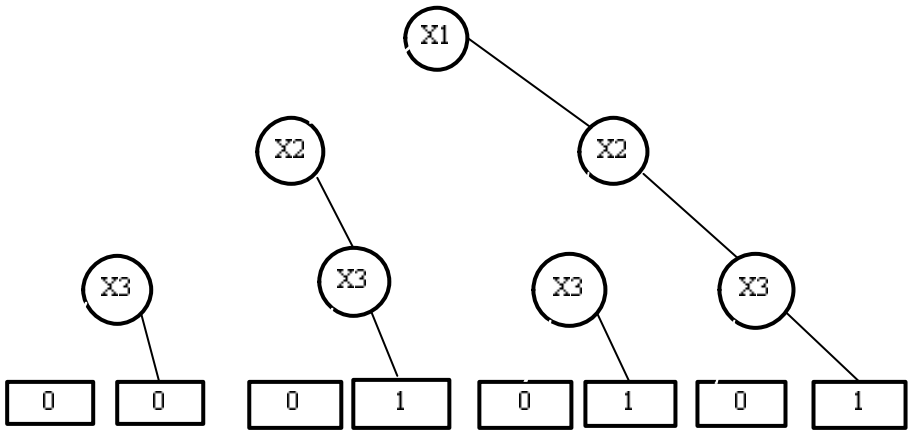
-Dso: Her fonksiyonu tek çikisli fonksiyon gibi minimumlastirir. Terimler fonksiyonlar arasinda paylastirilmaz.

2.4. Ikili Karar Diyagrami Kullanarak Minimumlastirma Algoritmasi

BDD bir Boole fonksiyonunu, bir köklü yönlendirilmis çevrimsiz (döngüsüz) grafik olarak sunar. Tablo 2.1'de verilen dogruluk tablosunda tanimlanan $f(x_1,x_2,x_3)$ fonksiyonunun agaç yapisi olarak sunumu Sekil 2.1'de gösterilmistir. Her bir dügüm *v* (nonterminal) bir degisken *var (v)* tarafindan etiketlendirilir ve iki çocuga dogru yönlendirilmis baglantilara sahiptir : *lo(v)* (kesikli çizgi ile gösterilen) degiskenin 0 degerine atandigi duruma karsilik gelir ve *hi(v)* (sürekli çizgi ile gösterilen) degiskenin 1 degerine atandigi duruma karsilik gelir. Her bir son çikis (terminal) 0 ya da 1 olarak etiketlendirilir. Fonksiyonun degeri son çikisin belirttigi deger olur. Sekil 2.1'deki son çikislarin degerleri soldan saga dogru okunur ve bunlara dogruluk tablosunda karsilik gelen degerler yukardan asagiya dogrudur (Kebschull ve ark. 1992, Drechster ve ark. 1994).

Tablo 2.1. $f(x_1,x_2,x_3)$ fonksiyonunun dogruluk tablosu

x_1	x_2	x_3	f
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**



2.4.1. BDD siralama ve sadelestirme

Bir BDD tanimlanirken sirali BDD (Ordered BDD-OBDD) ve bir lojik fonksiyonun degiskenleri arasindaki siralama iliskisini sunmak için ikili agaç yapisi kullanilir. Bir *f* fonksiyonu için OBDD de su özellikler bulunmaktadir (Liaw ve Lin 1992, Lafferty ve Vardy 1999, Malik ve ark. 1988, Rudell 1993, Reddy ve ark. 1995, Jain ve ark. 1995).

1) Her bir son çikis olmayan dügüm bir degiskenle birlestirilir. Bu kökten bir son çikisa dogru yoldaki degiskenler arasindaki siralama iliskisini saglar.

2) Her bir son çikis olmayan dügüm sag ve sol olmak üzere iki çocuga sahiptir ve her biri iliskili (birlestirilmis) degiskene göre 0 ya da 1 degerine karsilik gelir.

3) Her bir son çikis bir sabite karsilik gelir (0 ya da 1).

4) Her bir degisken, kökten son çikisa dogru herhangi bir yolda kesinlikle sadece bir kere görülebilir. Bundan dolayi, kökten son çikisa giden bir yol, *f* lojik fonksiyonunu degiskenlerinin örnek (özel) bir dogruluk atamasi için hesaplar.

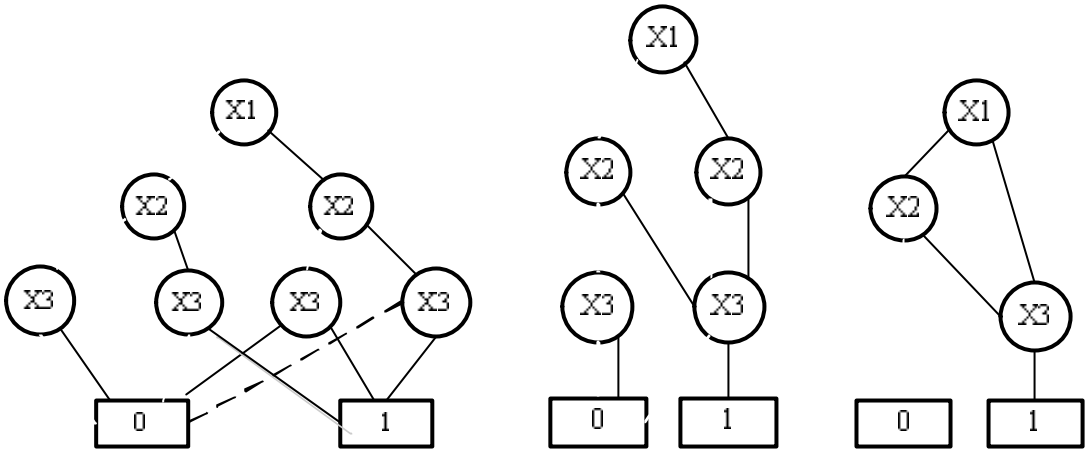
Sekil 2.1'de *f* fonksiyonu için degiskenlerin $x_1<x_2<x_3$ olacak sekilde siralandirilmis bir karar agaci gösterilmektedir. Prensip olarak degisken siralamasi rasgele seçilebilir (algoritmalar herhangi bir siralama için dogru olarak islem görecektir).

OBDD'yi sadelestirmek için asagidaki 3 tane dönüsüm kuralini kullanabiliriz (Bryant 1992, Liaw ve Lin 1992, Lafferty ve Vardy 1999, Malik ve ark. 1988, Rudell 1993, Bern ve ark. 1995, Gergov ve Meinel 1994, Sieling ve Wegener 1995).

a) Es terminallerin kaldirilmasi: Birbiri ile ayni olan terminallerden birisi hariç digerleri silinir ve silinen terminallere yöneltilen baglantilar kalan terminale yönlendirilir. Sekil 2.1'e bu kural uygulandiktan sonra Sekil 2.2 (a)'daki gibi olur.

b) Es dügümlerin kaldirilmasi: Eger dügüm *u* ve *v var(u)=var(v), lo(u)=lo(v)* ve *hi(u)=hi(v)* denkliklerine sahipse, iki dügümden birisi yok edilir ve tüm baglantilar diger dügüme yönlendirilir. Bu kural Sekil 2.2.(b)'de gösterilmistir.

c) Fazla dügümlerin kaldirilmasi: Eger dügüm *v lo(v)=hi(v)* denkligine sahipse, *v*'yi kaldirilir buraya gelen tüm baglantilar *lo(v)*'ye yönlendirilir. Bu kural Sekil 2.2.(c)'de gösterilmistir.


X1
X2
X2
X3
X3
X3
X3
0
1
X1
X2
X2
X3
X3
0
1
X1
X2
X3
0
1

3. ANAHTARLAMA FONKSIYONLARI IÇIN YEREL BASITLESTIRME ALGORITMALARI

Boole ifadelerinin sadelestirmesi, mantik devrelerinin ve dolayli olarak bilgisayar programlarinin daha etkili olmasina yol açmaktadir. Minimumlastirma ifadeleri önemlidir. Çünkü elektrik devreleri, verilen Boole ifadelerinin her bir terim veya literallerinin uygulanmasi için bireysel bilesenler içerir. Bu tasarimcilarin daha az bilesen kullanmasini ve böylece de belirli sistemlerin maliyetlerinin düsmesini saglamis olur. Tek çikisli veya çok çikisli Boole minimumlastirma teknikleri (Miller 1965, Mano 1984) anlatilmistir. Bu tekniklerin birçogu iki adimda çalisir. Ilk adimda bütün asal implikantlari (prime implicant-AI) belirler ve ikinci adimda da verilen Boole ifadesini örtecek (kapsayacak) AI'larin altkümesini seçer (Perkins ve Rhyne 1988).

Bütün AI'larin belirlenmesi sürecinde son sonucun tam olarak belirlenmesi için ayri durumlarda hesaplama yapilabilir. Özellikle, eger her bir asal implikant tam olarak *k* tane 0, *k* tane 1 ve *k* tane belirsiz terim içeriyorsa, AI'nin tamamlanmis kümesinin gücü $M=(3k)!/(k!)^3$ dür (Miller 1965, Gurunath ve Biswas 1989). Örnegin k=1,2,3,4 için sirasiyla M=6, 90, 1680 ve 34650 dir. *n* degiskenli bir fonksiyon için AI'larin sayisi $3^n/n$ kadar büyük olabilir (Perkins ve Rhyne 1988, Gurunath ve Biswas 1989). Sonuç olarak, AI belirleme adimi degisken sayisi *n* arttikça elverissiz bir duruma gelebilir (Perkins ve Rhyne 1988). Açikça görülmektedir ki ister iki seviyeli veya isterse çok seviyeli Boole ifadelerini sadelestirme prosedürlerinin hepsi tüm durumlarda $O(2^n)$ karmasikligina sahiptir (Bartlett ve ark. 1988, Allahverdi ve ark. 2000, Bernasconi ve ark. 2001, Beckert ve ark. 1997, Kahramanli ve Basçiftçi 2003, Kahramanli ve ark. 2005). Burada, tam belirlenmis Boole fonksiyonunun ON mintermlerini örten AI'larin yerel belirlenmesinin metodu önerilmistir. *n* degiskenli Boole ifadelerinin bu tür mintermleri maksimum *n* tek boyutlu küplere dahil edilebilir. Geçici sonuç küpleri kümesinin gücü *n* degerini geçmeyebilir (Allahverdi ve ark. 2000). Böylece, AI'larin minimum kümesini bulmak için $O(2^n)$ karmasikligi yerine $O(n)$ karmasikligi metodu kullanilabilir (Kahramanli ve Basçiftçi 2003, Kahramanli ve ark. 2005).

Bu çalismada, Off küme tabanli dogrudan örtme minimumlastirma metodu (direct cover minimization method) tek çikisli fonksiyonlar için çarpim terimlerinin toplami formunda sunulmustur. Var olan dogrudan örtme metotlarinda verilen On-küpü içeren yeterli asal implikantlar kümesini bulmak için, bu küp her defasinda bir koordinat için genisletilir. Her genislemenin dogrulugu, $k < 2^n$ Off-küplerin hepsi ile genisletilen küp kesistirilerek kontrol edilir. Bir küpün genislemesinin polinominal karmasikliga sahip oldugu dikkate alindiginda, bu yaklasimin toplam karmasikligi $O(n^p)O(2^n)$ seklinde olmaktadir. Bu polinominal ve üssel (exponansiyel) karmasikligin çarpimidir. Verilen On-küpü içeren asal implikantlarin tam kümesini elde etmek için önerilen metot, bu On-küp tarafindan genisletilen Off-küpleri kullanir. Bu islemin karmasikligi, yaklasik olarak bir koordinat için bir On-küpün genisletilme karmasikligina esdegerdir. Bundan dolayi, verilen On-küpü içeren asal implikantlarin tam kümesinin hesaplama isleminin karmasikligi yaklasik olarak $O(n^p)$ kadar azaltilmis olur. Pratik olarak bu yaklasim bir defada islenecek olan asal implikant sayisini yüzlerce ve binlerce defa azaltmaktadir. Bu ise halen problem olan bellek kapasitesi darbogazini kolaylikla asma imkani saglamaktadir.

Sunulan metot çesitli problemler üzerinde test edilmis ve standart MCNC bencmarklari kullanilarak ESPRESSO ile karsilastirilmistir. Bu karsilastirmalar sonucunda gelistirilmis olan yöntemlerin ESPRESSO'ya göre önemli bir ölçüde hizli oldugu ve az bellek kapasitesi gerektirdigi görülmüstür. Ayrica sadelestirme islemleri sonucunda karsilastirilan algoritmaya göre çarpim terimlerinin toplami seklinde daha iyi sonuç bulduklari belirlenmistir.

3.1. Isaretler Gösterimi

n girisli ve *m* çikisli bir çoklu çikisa sahip Boole fonksiyonu asagidaki gibi tanimlanir (Dagenais ve ark. 1986, Kahramanli ve Basçiftçi 2003):

Giris : B={0,1},
Çikis : Y={0,1,d},
Fonksiyon f : $B^n \rightarrow Y^m$.

Burada, çikista gösterilen *d* degeri (belirsiz terim) tam belirlenmemis deger manasindadir ve fonksiyonun istenildigi yerinde 0 veya 1 olarak kabul edilebilir. Böyle bir fonksiyon AI'larin listesiyle temsil edilebilir. Her bir AI giris ve çikis kisimlarini içerir (Kahramanli ve Basçiftçi 2003, Kahramanli ve ark. 2005).

Giris kismi: n sabitler {0,1,x} olabilir;

Çikis kismi: m sabitler {0,1,d} olabilir.

Giris kismi küpe uygulanacak giris uzayini belirler. Giris kismindaki x degeri bu degisken için 0 veya 1 degeri olabilir.

Bu tezde, tek çikisli Boole fonksiyonlari için yeni bir sadelestirme metodu gelistirilmistir. Boole fonksiyonu asagidaki gibi tanimlanir;

Giris : B={0,1},

Çikis : Y={0,1,d},

Fonksiyon f : $B^n \rightarrow Y$.

S_{ON} : Fonksiyonun degerini 1 yapan ON mintermlerinin kümesi,

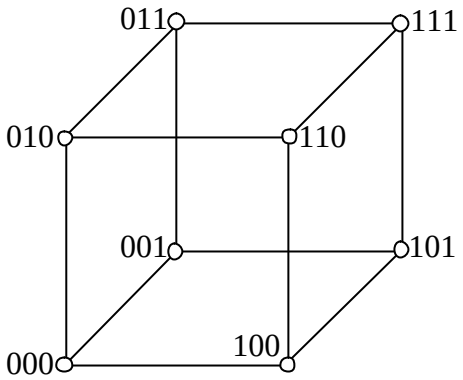
S_{OFF} : Fonksiyonun degerini 0 yapan OFF mintermlerinin kümesi,

S_{DC} : Belirsiz terim mintermlerinin kümesi.

Bu tezde sunulan algoritmada S_{ON} kümesi ve S_{OFF} kümesi tamamen kullanilmistir. S_{DC} kümesi ise kullanilmamistir.

3.2. Küp Cebri'nin Lojik Minimumlastirma Amaci ile Uygulanmasi

Lojik cebirdeki minimum terimler, küp cebrinin temelini olusturmaktadir. Ancak küp cebrinde degisken sayisi en az üçtür. Üç degisken bir küpü tanimlamaktadir. Küp cebri ile geometrik olarak; bir minterm ile bir nokta, iki nokta ile bir hat, dört hattin birlesmesi ile bir yüz, alti yüzün birlesmesi ile bir küp tanimlanir. Sekil 3.1'de 3 boyutlu bir küp görülmektedir (Miller 1965, Nadjafov ve Kahramanov 1973, Günes 2000). Bu küpün her bir koordinati, 3 degiskenli bir Boole fonksiyonunun bir degiskenidir.

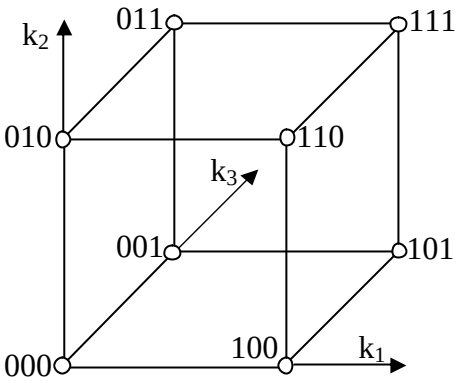


Sekil 3.1. Üç boyutlu küp

Küp cebri islemleri, önce anahtarlama fonksiyonlarinin (Switching Functions-SFs) en son durumunu bulmak için gelistirilmis ve uygulanmistir (Roth 1956, Nadjafov ve Kahramanov 1973). Yine bu islem SF'nin ilk terimlerini (local prime implicants) bulmak içinde kullanilmistir. Daha sonra lojik fonksiyonlarin minimumlastirilmasi üzerinde kullanilmistir (Nadjafov ve Kahramanov 1973, Dietmeyer 1979).

3.2.1. Küp cebrinin elemanlari ve uygulama biçimi

n-boyutlu bir küpün her bir tepe noktasi ikili kodlarla belirtilir. Bu küp $k_1,k_2,...,k_n$ koordinatlarina sahiptir. Dogal olarak k_i koordinati (0,1)'lerle belirtilir ve i=1,2,...,n'dir. Bu yüzden ayni zamanda, belirli bir tepe noktasinin kodu, bu tepe noktasinin cebirsel ifadesini gösterir. Tepe noktalarina komsu olan diger tepe noktalari da n bitlik kodlarla belirlenir. n bitlik kodlar, birbirinden sadece 1 bitlik farka sahipse bunlar komsu olarak adlandirilir. Örnegin 0110 kodu ile 0100 kodu komsudur (Nadjafov ve Kahramanov 1973, Günes 2000, Allahverdi 1999).



Sekil 3.2. Üç boyutlu kodlanmis küp

Sekil 3.2'de bulunan üç boyutlu küpün koordinat eksenleri k_1, k_2, k_3 ve $\tilde{k}_1, \tilde{k}_2, \tilde{k}_3$ olup $k_i \in \{0,1\}$ dir. Burada $\tilde{k}_i$, k_i koordinat ekseni üzerindeki bir degerdir. Sekil 3.2'de görüldügü gibi, küpün herhangi bir tepe noktasinin tespiti için, bu tepe noktasinin koduna ait bütün bitlerin {0,1} kümesi içinde olmasi gerekir. Diger bir deyisle, herhangi bir tepe noktasini tespit etmek için, bu tepe noktasinin kodunun tam olarak bilinmesi gerekir. Fakat küpün belirli bir kenarinin tespiti için iki koordinatinin bilinmesi yeterlidir. 000 ve 001 düsey kenarlarin gösterilmesi için, koordinatin $\tilde{k}_1 = 0$, $\tilde{k}_2 = 0$ yani $\tilde{k}_1 \tilde{k}_2 = 00$ degerinin verilmesi yeterlidir. 000 ve 010 ile belirtilen dogru için $\tilde{k}_1 = 0$, $\tilde{k}_3 = 0$ yani $\tilde{k}_1 \tilde{k}_3 = 00$ koordinatlarinin verilmesi yeterlidir. 000 ve 100 dogrusu için $\tilde{k}_2 = 0$, $\tilde{k}_3 = 0$ yani $\tilde{k}_2 \tilde{k}_3 = 00$ degerinin verilmesi yeterlidir. Bu örneklerin ilkinde k_3 koordinati, ikincisinde k_2, üçüncüsünde k_1 koordinati tam olarak gösterilmez. Çünkü bunlara kendi koordinatlarinin bilinmesi için gerek duyulmaz. Küpün elemanlari; *tepe, dogru, yüz, küp, hiperküp* seklinde adlandirilir. Bu elemanlar üzerinde bir islem yaparken gelebilecek belirsiz durumlarin olusmamasi için, bütün koordinatlari ve bu koordinatlarin arasindaki dogrularin kullanilmasi gerekir. Bu amaç için, {0,1} kümesine ait olmayan geçersiz koordinatlarin konumlari " * " ile belirtilmistir. Bu sembol, belirsiz, önemsiz (don't care) ya da keyfi deger olarak adlandirilir. Örnegin, $\tilde{k}_1 \tilde{k}_2 = 00$, $\tilde{k}_1 \tilde{k}_3 = 00$ ve $\tilde{k}_2 \tilde{k}_3 = 00$ dogrulari sirasiyla, 00*, 0*0 ve *00 olarak gösterilirler. Buna göre

$k_i \in \{0,1,*\}$, i=1,2,..,n olacaktir. m belirsiz koordinatlarin degerini içeren gösterim, m-küp olarak adlandirilir. Buna göre sadece 0 veya 1'ler ile gösterilen tepe noktasi 0-küp olusturur. Bir dogrunun gösterimi için bir tane belirsiz koordinat olmalidir. Bu durumda dogru, n-1 koordinattan olustugu için bu dogru (küpün kenari) 1-küp olarak adlandirilir. Bir yüzeyin gösteriminde iki tane belirsiz koordinat oldugu için 2-küp olarak adlandirilir. Bu belirsiz degerlerin artisina paralel olarak m-küp degeri de buna bagimli olarak degismektedir (Allahverdi ve Kahramanli 1995).

3.2.2. Küp cebrinin islemleri

Küp cebri, lojik 0 ve lojik 1 ile yapilan bütün islemlerin disinda dört islemi daha içermektedir. Bunlar;

a) Koordinatli çarpma (coordinate product, star product or ☆-operation),

b) Koordinatli çikarma (coordinate subraction, sharp product or #-opration),

c) Koordinatli kesisme (coordinate intersection or ∩-operation)

d) Dönüsümlü yutma islemi (commutative absorption operation - ? operation)

islemleridir (Miller 1965, Allahverdi ve Kahramanli 1995).

3.2.2.1. Koordinatli çarpma islemi (☆ islemi)

Koordinatli çarpma islemi, ayni boyuta sahip iki küp arasinda uygulanir. Fakat çarpimi yapilacak olan altküpler, ayni boyutta olmak sarti ile degisik degerde olabilirler. Koordinatli çarpma islemi iki asamada gerçeklestirilir. Ilk asamada bir v_i bileseninin belirlenmesi için çarpim vektörü (vector of product-VP) olusturulur. Ikinci asamada, olusturulan VP'nin koordinat degerlerine göre A ve B küpleri koordinatli çarpima tabii tutulur (Dietmeyer 1979, Allahverdi ve Kahramanli 1995, Allahverdi 1999).

A ve B küpleri ayni boyuta sahip iki küp olsun;

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

Bu iki küp arasinda koordinatli çarpma islemi uygulansin. Ilk asamada v_i bileseninin belirlenmesi için asagidaki islemler sonucunda çarpim vektörü VP olusturulur;

$VP = A \otimes B = v_1 v_2 ... v_i ... v_n$

olmak üzere, v_i bileseninde $i \in \{1,2,...,n\}$ dir. v_i bileseni;

Eger $a_i = b_i$	ise	$v_i = a_i = b_i$ olur.
Eger $a_i = *$ ve $b_i \neq *$	ise	$v_i = b_i$ olur.
Eger $a_i \neq *$ ve $b_i = *$	ise	$v_i = a_i$ olur.
Eger $a_i, b_i \in \{0,1\}$ ve $a_i = \bar{b}_i$	ise	$v_i = y$ olur.

VP'nin koordinat degerlerine göre A ve B küplerinin koordinatli çarpiminin sonuçlari asagidaki gibi olmaktadir;

a) Eger herhangi bir $v_i = y$ bulunmazsa, A ve B'nin çarpimi sonucu, A ve B'nin alt küpü olan VP olmaktadir (yani $A \otimes B = VP$).

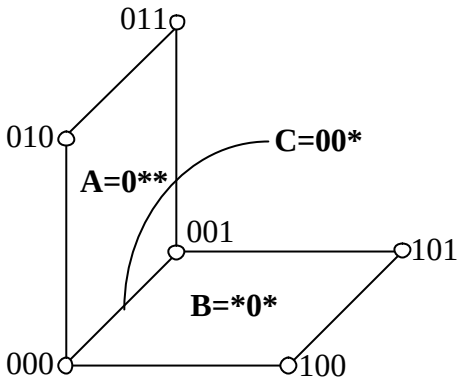
b) Eger sadece bir tane i degeri için $v_i = y$ bulunuyorsa ve diger degerler için $v_j = a_j = b_j$ ise (burada $j \in \{1, 2, ..., i-1, i+1, ..., n\}$ dir) A ve B küplerinin çarpimi sonucu, VP'de v_i yerine * sembolü konularak bulunan bir C küpüdür.

c) Eger sadece bir tane $v_i = y$ olusuyor ve $a_j = *$ veya $b_j = *$ için $v_j \neq *$ ise A ve B küplerinin çarpimlari sonucu, VP'de v_i yerine * sembolü konularak bulunan bir C küpüdür. C küpünün bir bölümü A tarafindan, diger bölümü B tarafindan örtülür. Bu durum C küpünün A ve B küpleri ile iliskisi oldugunu gösterir.

d) Eger en az iki tane v_i ve v_j bileseni için, $v_i = v_j = y$ olan A ve B küplerinin koordinatli çarpimi $C = \varnothing$ dir. Burda A ve B küpleri arasinda dogrudan bir baglanti yoktur.

a, b, c, d maddelerine göre A^m (m-küp) ile B^r (r-küp) küplerinin çarpilmasi sonucu asagidaki durumlar olusmaktadir.

i: Her iki A ve B küpüne giren C küpü (Sekil 3.3).

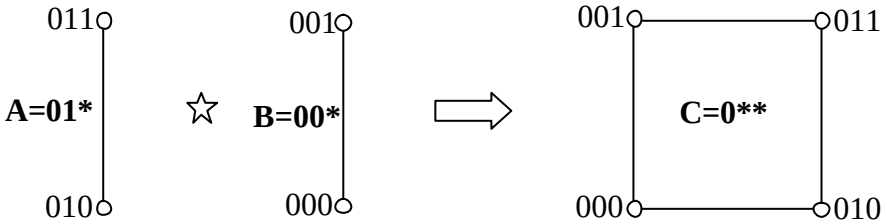


$VP = A \circledast B = 0** \circledast *0* = 00*$

$C = A \star B = VP = 00*$

Sekil 3.3. **C küpünün ortak olmasi durumu**

ii: (m+1) kenara sahip olan (m+1-küp) ve A ve B küpünün birlesmesinden olusan C küpü (Sekil 3.4).

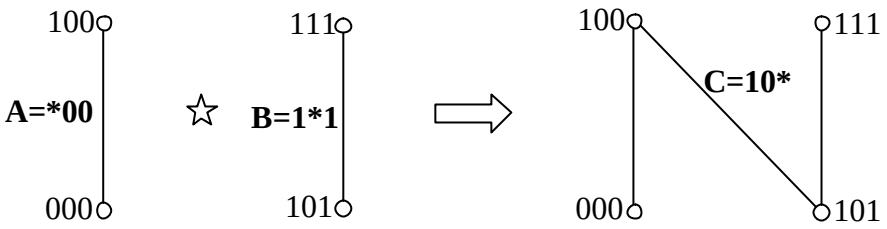


$VP = A \circledast B = 01* \circledast 00* = 0y*$

$C = A \star B = VP \xrightarrow{VP_2 = y} 0y* \xrightarrow{VP_2 = *} 0**$

Sekil 3.4. **C küpünün birlesim olusturdugu durum**

iii: A ve B küpleri arasinda birlestirilmis bir köprü olan C küpü (Sekil 3.5).

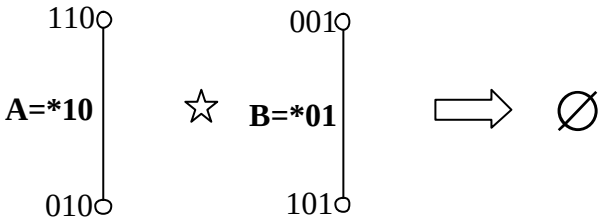


$$VP = A \circledast B = 00* \circledast 1*1 = 10y$$

$$C = A \star B = VP \xrightarrow{VP_3 = y} 10y \xrightarrow{VP_3 = *} 10*$$

Sekil 3.5. **C küpünün köprü olusturdugu durum**

iv: A ve B küpleri arasinda dogrudan bir baglanti olmadigini gösteren C = ∅ küpü (Sekil 3.6).



$$VP = A \circledast B = *10 \circledast *01 = *yy$$

$$C = \varnothing$$

Sekil 3.6. **C küpünün bos küme olmasi durumu**

Koordinatli çarpim asagidaki özelliklere sahiptir;

1) $A^m \star B^r = B^r \star A^m$ (Degisme özelligi vardir).

2) $A^m \star (B^r \star C^k) = (A^m \star B^r) \star C^k$ (Birlesme özelligi vardir).

Koordinatli çarpma ile asagidaki islemler gerçeklestirilebilir.

1) VP içindeki y bileseninin sayisina esit olan küpler arasinda Hamming uzakliginin tanimlanmasi.

2) Küplerin ortak parçalarinin (alt küp) tanimlanmasi amaci ile iki veya daha fazla küpün kesismesi.

3) Iki tane m-küp olan A ve B'yi (m+1)-küp olan C içinde birlestirmek.

3.2.2.2. Koordinatli çikarma islemi (# islemi)

Koordinatli çarpmada oldugu gibi, koordinatli çikarmada da ayni boyuta sahip iki küp kullanilir. Çikarma islemi, küplerin ayni taraflarinda (nokta, kenar, yüzeylerinde) veya farkli taraflarinda yapilabilir. Koordinatli çikarma islemi ilk olarak, tepe noktasi adreslenmemis SFs nin sinirini hesaplamada kullanilmistir. Bununla birlikte, SFs nin yerel asal implikantlarinin (local prime implicants) bulunmasina uygulanmistir (Allahverdi ve Kahramanli 1995, Kahramanli ve Allahverdi 1993, Brown ve Vranesic 2003, Kahramanli ve Allahverdi 1996, Günes ve ark. 2003, Allahverdi 1999).

A ve B küpleri ayni boyuta sahip iki küp olsun;

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

seklinde gösterilen iki küp arasindaki koordinatli çikarma islemi iki asamada gerçeklestirilir. Birinci asamada A ve B küplerine göre, kurallara uygun olarak,

$VS = A \textcircled{\#} B = v_1 v_2 ... v_i ... v_n$

seklinde çikarma vektörü (vector of subtraction-VS) olusturulur. Ikinci asamada VS'ye göre; A küpü, C sonuç küpüne dönüstürülür. C sonuç küpü bir küp veya bir kaç küpü içeren bir küme olabilir. v_i vektörünün durumuna göre asagidaki dönüsümler gerçeklestirilir.

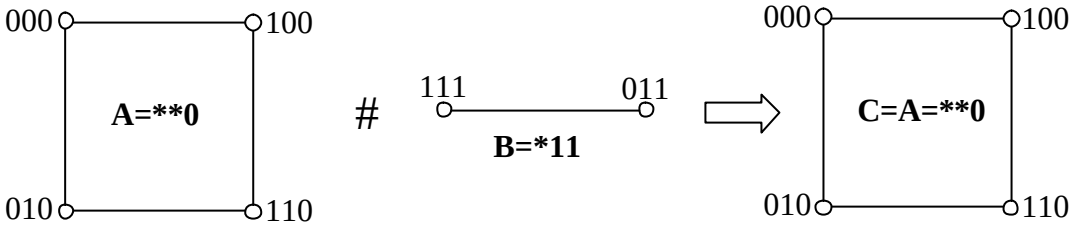
Eger $b_i = *$ veya $b_i = a_i$ ise $v_i = z$ dir.

Eger $a_i = *$ ve $b_i \neq *$ ise $v_i = \bar{b}_i$ dir.

Eger $a_i, b_i \in \{0,1\}$ ve $a_i = \bar{b}_i$ ise $v_i = y$ dir.

VS degerine göre, ikinci asamada, koordinatli çikartmanin sonucu asagidaki durumlarda tanimlanmaktadir:

a) Eger en az bir tane $v_i = y$ olusursa, çikarma islemi mümkün olmayacaktir. Bu demektir ki, C = A # B = A (Sekil 3.7) dir.

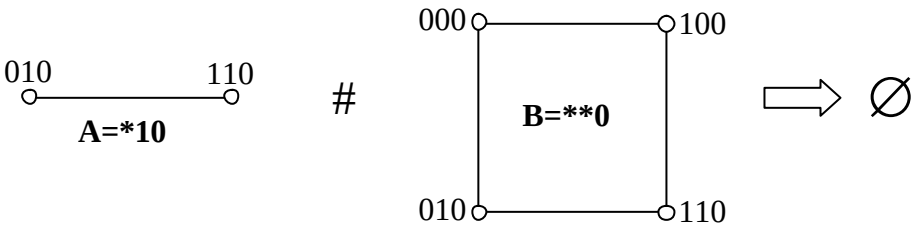


$VS = A \textcircled{\#} B = **0 \textcircled{\#} *11 = z0y$

$C = A \xrightarrow{VS} = **0 \xrightarrow{z0y} **0$

Sekil 3.7. **C kümesinin çikarilan küme olmasi durumu**

b) Eger bütün i = 1, 2, ..., i, ..., n degerleri için $v_i = z$ ise, çikarma isleminin sonucu bos küme olacaktir. Yani, C = A # B = ∅ (Sekil 3.8) dir.

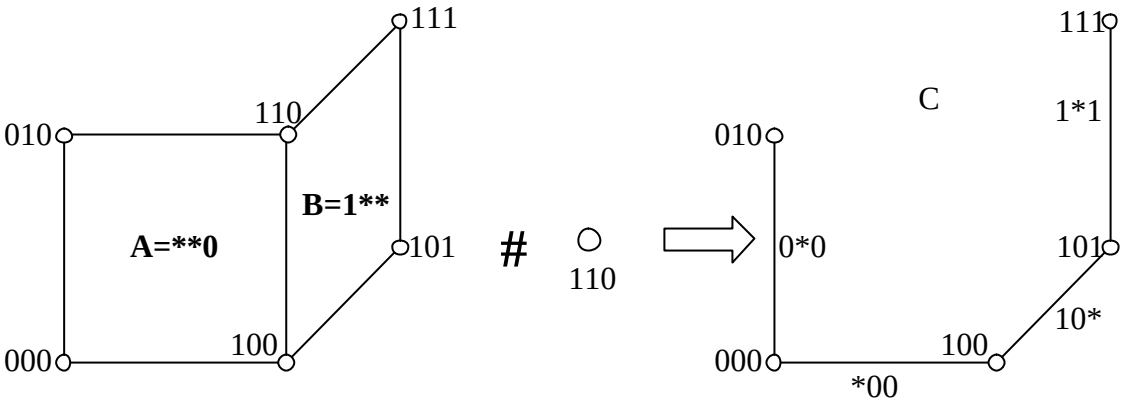


$VS = A \textcircled{\#} B = *10 \textcircled{\#} **0 = zzz$

$C = A \xrightarrow{VS} = *10 \xrightarrow{zzz} \varnothing$

Sekil 3.8. **C'nin bos küme olmasi durumu**

c) Eger bir tane bile $v_i = y$ olusmuyorsa ve $v_j, v_k, ..., v_m \in \{0,1\}$ ise, çikarma isleminin sonucu, $\{ a_1a_2...a_{j-1}v_ja_{j+1}...a_n\ ,\ a_1a_2...a_{k-1}v_ka_{k+1}...a_n\ ,\ a_1a_2...a_{m-1}v_ma_{m+1}...a_n\ \}$ kümesi olacaktir (Sekil 3.9).



$VS = \{A,B\} \textcircled{\#} 110 = \{**0, 1**\} \textcircled{\#} 110$

$VS_1 = **0 \textcircled{\#} 110 = 00z,\ C_1 = **0 \xrightarrow{VS_1} = **0 \xrightarrow{00z} = \{0*0, *00\}$

$VS_2 = 1** \textcircled{\#} 110 = z01,\ C_2 = 1** \xrightarrow{VS_2} = 1** \xrightarrow{z01} = \{10*, 1*1\}$

$C = C_1 \cup C_2 = \{0*0, *00, 10*, 1*1\}$

Sekil 3.9. **C'nin degisik durumlari**

Eger i ($i \in \{1,2,...,n\}$) koordinati üzerinde A küpünden B küpünün çikarilmasi sonucunda $C_i \in C$ küpü olusturuluyorsa bu C_i küpüne i koordinati üzerindeki fark küp denir. Eger **0 # 110 = {0*0, *00} ise o zaman 0*0 ve *00 küpleri sira ile koordinat 1 ve koordinat 2 üzerindeki fark küpleridir. Bu örnekde koordinat 3 üzerinde herhangi bir fark küp yoktur.

Yukarida gösterildigi gibi, $A = a_1, a_2, ..., a_n$ ve $B = b_1, b_2, ..., b_n$ seklinde gösterilen iki küp arasindaki koordinatli çikarma islemi iki asamada gerçeklestirilir. Birinci asamada çikarma vektörü VS asagidaki kurallar kullanilarak $VS=A\otimes B=v_1v_2...v_i...v_n$ seklinde olusturulur (Kahramanli ve Basçiftçi 2003).

- Eger $b_i=x$ veya $b_i=a_i$, ise $v_i=Z$
- Eger $a_i=x$ ve $b_i?x$, ise $v_i=\bar{b}_i$
- Eger $a_i=\bar{b}_i$, ise $v_i=Y$

Tablo 3.1. **Koordinatli çikarma islemi**

a_i \ b_i	**x**	**1**	**0**
x	Z	0	1
1	Z	Z	Y
0	Z	Y	Z

Ikinci asamada, VS degerine göre, koordinatli çikartmanin sonucu asagidaki durumlarda tanimlanmaktadir (Kahramanli ve Basçiftçi 2003):

- Eger $\exists i$, $v_i=y$ ise, çikarma islemi mümkün olmayacaktir. Örnegin, C=A#B=A.
- Eger hiç $\exists i$, $v_i=y$ olusmuyorsa ve v_j, v_k, ..., $v_m \in \{0,1\}$ ise, çikarma isleminin sonucu, $\{ a_1a_2...a_{j-1}v_ja_{j+1}...a_n$, $a_1a_2...a_{k-1}v_ka_{k+1}...a_n$, $a_1a_2...a_{m-1}v_ma_{m+1}...a_n \}$ kümesi olacaktir.
- Eger $\forall i$, $v_i=Z$ ise, çikarma isleminin sonucu bos küme olacaktir. Yani, C=A#B=$\varnothing$.

Koordinatli çikarma islemi asagidaki özelliklere sahiptir;

1) $A \# B \neq B \# A$ (Degisme özelligi yoktur).

2) $(A \# B) \# C \neq A \# (B \# C)$ (Birlesme özelligi yoktur).

3) $(A \cup B) \# C = (A \# C) \cup (B \# C)$ (Birlesme özelligi üzerinde dagilma özelligi vardir).

4) $(A \cap B) \# C = (A \# C) \cap (B \# C)$ (Kesisme özelligi üzerinde dagilma özelligi vardir).

5) $A \# \{B,C\} = (A \# B) \# C = (A \# C) \# B$ (Çikarmada simetrilik vardir).

3.2.2.3. Koordinatli kesisme islemi ($\cap$ islemi)

Koordinatli kesisme islemi, iki küp arasinda mevcut olabilecek bir ortak alt küpün belirlenmesini amaçlamaktadir. Kesisme islemi, SFs nin bazi asal implikantlarindan diger asal implikantlara benzeyen terimlerin belirlenmesi için uygulanmistir (Nadjafov ve Kahramanov 1973, Kahramanli ve Allahverdi 1993, Allahverdi 1999). Ayrica bu islem asal implikantlardan hangisinin en büyük oldugunun bilinmesini saglar. A ve B küpleri arasinda yapilan kesisme islemi sonucunda kesisme vektörü (vector intersection-VI) olusur.

$VI = v_1 v_2 \ldots\ldots v_i \ldots\ldots v_n$

seklindedir. v_i bileseninin degerleri asagida verilmistir;

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

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

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

Eger $a_i, b_i \in \{0,1\}$ ve $a_i = \bar{b}_i$ ise $v_i = y$ dir.

Eger en az bir tane $v_i = y$ bulunuyorsa $A \cap B = \varnothing$ olur. Eger $v_i = y$ bulunmuyorsa $A \cap B = VI$ olur. Bu kurala göre kesisme islemi, Sekil 3.9'da görülen $C = \{0*0, *00, 10*, 1*1\}$ kümesi üzerinde gerçeklestirilecektir. Kolaylik olmasi bakimindan bu kümenin elemanlari 1, 2, 3, 4 ile gösterilirse, kesisme islemleri asagidaki gibi olacaktir;

$VI_{1,2} = 0*0 \otimes *00 = 000,$

$VI_{1,3} = 0*0 \otimes 10* = y00,$

$VI_{1,4} = 0*0 \otimes 1*1 = y*y,$

$VI_{2,3} = *00 \otimes 10* = 100,$

$VI_{2,4} = *00 \otimes 1*1 = 10y,$

$VI_{3,4} = 10* \otimes 1*1 = 101,$

Burada;

$0*0 \cap *00 = 000,$

$0*0 \cap 10* = \varnothing,$

$0*0 \cap 1*1 = \varnothing,$

$*00 \cap 10* = 100,$

$*00 \cap 1*1 = \varnothing,$

$10* \cap 1*1 = 101,$

Bu sonuçlar, Sekil 3.9'un sag tarafinda bulunan yapiyi olusturur. VI ve VP olusturmanin kurallari karsilastirildiginda VI ve VP'nin ayni oldugu görülebilir. Bu yüzden A ve B küplerinin kesismesi, VI vektörünün yorumuna uygun olarak, küplerin çarpimi olarak da anlasilabilir.

Yukarida açiklanan islemler dogrultusunda Yakin Minimum Örtme ve Kesin Minimum Örtme Algoritmalarinda kullanilacak dönüsümlü yutma islemi asagida açiklanmistir.

3.2.2.4. Dönüsümlü yutma islemi (Commutative absorption operation - ?)

$A = a_1, a_2, ..., a_n$ ve $B = b_1, b_2, ..., b_n$ seklinde gösterilen iki küp arasindaki dönüsümlü yutma islemi iki asamada gerçeklestirilir. Birinci asamada yutma vektörü (absorption vector-AV) asagidaki kurallar kullanilarak $AV=A\nabla B=v_1v_2...v_i...v_n$ seklinde olusturulur (Kahramanli ve Basçiftçi 2003).

- Eger $a_i=b_i$ ise $v_i=Z$
- Eger $a_i=x$ ve $b_i?x$ ise $v_i=G$
- Eger $a_i=\bar{b}_i$ ise $v_i=Y$
- Eger $a_i?x$ ve $b_i=x$ ise $v_i=L$

Tablo 3.2. **Dönüsümlü yutma islemi**

a_i \ b_i	**x**	**1**	**0**
x	Z	G	G
1	L	Z	Y
0	L	Y	Z

Ikinci asamada, AV degerine göre, dönüsümlü yutma isleminin sonucu asagidaki durumlarda tanimlanmaktadir (Kahramanli ve Basçiftçi 2003):

- Eger $\exists i$, v_i=y ise yutma islemi mümkün olmayacaktir. Örnegin, C=A **?** B={A,B}
- Eger $\forall i$, v_i=Z ise A=B dir. Örnegin C=A **?** B=A
- Eger $\exists i$, v_i=G ve hiç $\exists i$, v_i=L, ise C= A **?** B=A
- Eger $\exists i$, v_i=L ve hiç $\exists i$, v_i=G, ise C=A **?** B=B
- Eger $\exists i$, v_i=G ve $\exists i$, v_i=L, ise yutma islemi mümkün olmayacaktir. Örnegin, C=A**?**B={A,B}

Örnek 3.1.

1) A=x11x; B=x0x1 ise AV=x11x ∇ x0x1=ZYLG; C={x11x, x0x1}

2) A=x11x; B=x11x ise AV=x11x ∇ x11x=ZZZZ; C=x11x

3) A=x1xx; B=x1x1 ise AV=x1xx ∇ x1x1=ZZZG; C=x1xx

4) A=x011; B=xx1x ise AV=x011 ∇ xx1x=ZLZL; C=xx1x

5) A=xx0x; B=1xxx ise AV=xx0x ∇ 1xxx=GZLZ; C={xx0x, 1xxx}

3.3. Küp Cebri Islemlerinin Temel Bilgisayar Islemleri Üzerinden Gerçeklestirilmesi

Ileriki bölümlerde anlatilan Yakin-Minimum Örtme ve Kesin Minimum Örtme Algoritmalarinda bir önceki bölümlerde anlatilan küp cebri islemleri kullanilmistir. Ayni zamanda bir sonraki bölümde açiklanan asal implikantlarin yerel sekilde belirlenmesi teoremi ispatinin sonucunda bulunan kurallar kullanilarak $B_i = b_1^i b_2^i ... b_j^i ... b_n^i \in S_{OFF}$ mintermi $Q_i = q_1^i q_2^i ... q_j^i ... q_n^i$ küpüne dönüstürülmektedir. Bu islemelerin hepsi küp cebri islemleri kullanilarak seri bir sekilde yapilmaktadir. Seri bir sekilde bu islemleri gerçeklestirirken algoritmalarin çözüme ulasma zamanlari artmaktadir. Bu yüzden, bu islemleri temel bilgisayar islemleri yardimi ile paralel bir sekilde yaparak algoritmalarin çözüm zamanlari azaltilmaya çalisilmistir. Bu islemler sayesinde algoritmalarin daha hizli olmasi saglanmistir. Çünkü, bu islemler temel bilgisayar islemleri kullanilarak paralel bir sekilde yürütülmesi saglanmistir. Bu bölümde bu islemlerin nasil gerçeklestirildigi açiklanmaya çalisilacaktir.

Temel bilgisayar islemlerinin gerçeklestirilmesi sirasinda asagida gösterilen islemler kullanilacaktir.

1) Küplerin koordinat degerlerinin gösterilmesi

Bir küpün her bir koordinat degeri asagida gösterildigi gibi iki bit ile temsil edilmistir.

Esas koordinatin lojik 0 degeri : **01**
Esas koordinatin lojik 1 degeri : **10**
x (“*” veya “-”) terimi ile gösterilen esas olmayan koordinatin degeri : **11**

ile temsil edilmektedir. Burada 00 bit çifti için herhangi bir deger belirlenmemistir. Burada belirtilen bit çiftleri ile küpün koordinat degerlerinin temsil edilmesi saglanmistir. Çünkü küpün koordinat degerleri arasinda gösterilecek esas olmayan koordinat degerleri için “x” sembolü kullanilmistir. Bu isareti bir ve sifir cinsinden

ifade edebilmek için bu sekilde bir gösterim kullanilmistir. Örnek olarak A=10x0x1 küpü için asagida küpün her bir koordinat degeri iki bit ile gösterilmistir.

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

Küp Degeri	1	0	x	0	x	1
Her bir koordinatin iki bit ile gösterimi	10	01	11	01	11	10

2) Bir küpün gösterilmesi

Her bir küp $K = k_{n-1}\ k_{n-2} \ldots k_i \ldots k_0$ bir bit dizisi çifti ile temsil edilmistir.

$K_L = l_{n-1}\ l_{n-2} \ldots l_i \ldots l_0$

$K_R = r_{n-1}\ r_{n-2} \ldots r_i \ldots r_0$

Burada ifade edilen l_i ve r_i degerleri her bir bit dizisi çiftindeki sol ve sag bitlerin degerleridir. Örnegin, küp K=01x0x1 ise bu küpün bit dizisi çifti olarak gösterimi asagidaki gibi olacaktir.

Tablo 3.4. Bit dizisi olarak bir küpün gösterimi

K	0	1	x	0	x	1	Temsil edilen küp
K_L	0	1	1	0	1	1	Bit dizisi çifti olarak küpün gösterimi
K_R	1	0	1	1	1	0	

Burada verilen bir K küpü bit dizisi çifti olarak gösterilmistir. Elde edilen sonuçlara göre K=01x0x1 küpümüz K_L = 011011 ve K_R = 101110 bit dizisi

çiftlerine dönüstürülmüstür. Burada K_L bit dizisi çiftlerindeki sol bitlerin degerlerine, K_R ise sag bitlerin degerlerine sahip olan bit dizileridir.

Yukarida gösterilen islemler Off kümesinde bulunan mintermleri genisletmek için, küplerin kesisim degerlerinin bulunmasi gereken durumlarda ve küplerin birbirlerini kapsayip kapsamadiklarini veya bir küpün mintermi örtüp örtmediginin belirlenmesi gereken islemlerde kullanilmistir. Bu islemler asagida açiklanmistir.

a) Off kümesindeki mintermleri genisletmek için temel bilgisayar islemleri

Off-küp kümesinde bulunan $B_i = b_1^i b_2^i ... b_j^i ... b_n^i \in S_{OFF}$ mintermleri On-küp kümesinde bulunan $A=a_{n-1}\ a_{n-2}\ \dots\ a_i\ \dots\ a_0$ minterm ile genisleterek $Q_i = q_1^i q_2^i ... q_j^i ... q_n^i$ küpüne asagidaki kurallar kullanilarak dönüstürülebilir. Bu kurallar bir sonraki bölümde açiklanan asal implikantlarin yerel sekilde belirlenmesi teoreminin ispatindan elde edilmistir.

Eger $b_j^i = x$ ise $q_j^i = x$,

Eger $a_j = b_j^i$ ise $q_j^i = x$,

Eger $b_j^i = \bar{a}_j$ ise $q_j^i = b_j^i$.

Bu islemleri gerçeklestirirken Off kümesindeki her bir mintermi On kümesindeki minterm ile bit bit karsilastirma yapmak suretiyle Q_i küpünü elde ederiz. Bu da zaman açisindan büyük kayiplara ugramamiza neden olur. Bu kurallari temel bilgisayar islemleri ile asagidaki gibi gerçeklestirebiliriz.

$A=a_{n-1}\ a_{n-2}\ \dots\ a_i\ \dots\ a_0$ On kümesindeki minterm

$B=b_{n-1}\ b_{n-2}\ \dots\ b_i\ \dots\ b_0$ Off kümesindeki minterm

On kümesindeki mintermi (A), Off kümesindeki mintermle genisleterek C küpü elde edilir. Bu genisletme islemi için On kümesindeki minterm ile Off kümesindeki minterm bit dizisi çiftleri haline dönüstürülürler. Bu islemler asagida gösterilmistir.

$C_L = \overline{A_L} \vee B_L$

$C_R = \overline{A_R} \vee B_R$

$C = C_L \cup C_R$

Örnek 3.2.

A = 0 1 0 0 1 0 On kümesi mintermi

B = 1 0 0 1 1 0 Off kümesi mintermi

On kümesi minterminin (A) ve Off kümesi minterminin her bir biti, bit dizisi çifti olarak islenir.

A_L = 0 1 0 0 1 0 B_L = 1 0 0 1 1 0

A_R = 1 0 1 1 0 1 B_R = 0 1 1 0 0 1

$C_L = \overline{A_L} \vee B_L$ = 1 0 1 1 0 1 $\vee$ 1 0 0 1 1 0 = 1 0 1 1 1 1

$C_R = \overline{A_R} \vee B_R$ = 0 1 0 0 1 0 $\vee$ 0 1 1 0 0 1 = 0 1 1 0 1 1

Sonuç olarak bit dizisi çiftleri (C_L, C_R) asagidaki gibi birlestirilerek C küpü elde edilir.

Tablo 3.5. Örnek 3.2 için C küpünün belirlenmesi

C_L	1	0	1	1	1	1
C_R	0	1	1	0	1	1
C	**1**	**0**	**x**	**1**	**x**	**x**

b) Iki küpün (A ve B) kesisimini temel bilgisayar islemleri ile gerçeklestirmek

$A = a_1\ a_2 \ldots a_i \ldots a_n$ ve $B = b_1\ b_2 \ldots b_i \ldots b_n$ kesisim islemine uygulanacak iki küp olsun. Asagida gösterilen C küpü A ve B küplerinin kesisimi sonucunda elde edilen küptür.

$C = A \cap B$

Bu kesisim islemi sonucunda elde edilen C küpünün bit dizisi çiftleri C_L ve C_R bit dizileri A ve B küplerinin bit dizilerinin "ve" islemine tabi tutulmasi ile bulunur. Burada,

$C_L = A_L \wedge B_L$
$C_R = A_R \wedge B_R$ dir.

Elde edilen C_L ve C_R bit dizileri sonucunda, A ve B küplerinin kesisim degeri belirlenmeye çalisilir. C küpünün degerinin belirlenmesi için asagidaki islemler gerçeklestirilir.

i) Sonucun bos küme olup olmadiginin kontrol edilmesi

Bulunan C_L ve C_R bit dizileri ve C küpü degeri ile, A ve B küplerinin kesisim degerinin herhangi bir degere veya bos kümeye esit olup olmadigi asagidaki islemlerle kontrol edilir.

$D = (C_L \vee C_R) \oplus 11\ldots11$
Eger $D \neq 0$ ise $C = \varnothing$

Bu islemler sonucunda elde edilen D küpü sifira esit degilse C küpü bos kümedir. Yani, A ve B küplerinin kesisiminden bir deger elde edilmemistir. D küpü sifira esitse A ve B küplerinin kesisiminden bir deger elde edilmis olacaktir.

ii) A ve B küplerinin C sonuç küpü ile karsilastirilmasi

A ve B küplerinin kesisimi sonucunda elde edilen C küpü herhangi bir degere sahipse, bu degerin A veya B küplerinden hangisine ait oldugunu asagidaki islemler dogrultusunda bulabiliriz. Bu islemler sonucunda kesisim degerinin hangi küpe ait oldugunu bulmakla beraber küpler arasindaki kapsama durumlari da bulunmus olmaktadir.

- Eger $C_L = A_L$ ve $C_R = A_R$ ise C = A dir.
- Eger C = A ise A küpünü çikar degilse A küpünü tut.
- Eger C = B ise B küpünü çikar degilse B küpünü tut.

Ayni sonuçlar dogrultusunda kapsama durumlari da belirlenmektedir.

- Eger sadece $C \neq A$ ise A küpü B küpünü kapsar ($A \supset B$)
- Eger sadece $C \neq B$ ise B küpü A küpünü kapsar ($B \supset A$)
- Eger hem $C \neq A$ ve hem de $C \neq B$ ise her iki küpte birbirini kapsamaz.

Örnek 3.3.

A = 1 0 x 0 x 1 ve B = 1 0 x 0 0 1 iki küp olsun. Bu küpler arasindaki kesisme durumunu ve birbirlerini kapsama durumunu temel bilgisayar islemleri ile gerçeklestirirsek;

A ve B küplerinden $A_L - A_R$ ve $B_L - B_R$ bit dizilerini asagidaki gibi olustururuz.

A_L = 1 0 1 0 1 1 B_L = 1 0 1 0 0 1

A_R = 0 1 1 1 1 0 B_R = 0 1 1 1 1 0

Elde edilen bit dizilerinden C_L ve C_R bit dizileri dolayisiyla da C küpü elde edilir.

$C_L = A_L \wedge B_L$ = 1 0 1 0 1 1 $\wedge$ 1 0 1 0 0 1 = 1 0 1 0 0 1

$C_R = A_R \wedge B_R$ = 0 1 1 1 1 0 $\wedge$ 0 1 1 1 1 0 = 0 1 1 1 1 0

$C (C_L, C_R) = A \cap B$ = 1 0 x 0 0 1

i) Sonucun bos küme olup olmadiginin kontrol edilmesi

$D = (C_L \vee C_R) \oplus 11\ldots11 = (1\,0\,1\,0\,0\,1 \vee 0\,1\,1\,1\,1\,0) \oplus 1\,1\,1\,1\,1\,1 = 0$

$D = 0$ oldugu için $C \neq \varnothing$ dir.

Yani bu islemler sonucunda A ve B küplerinin kesisiminden bir deger elde edilmistir.

ii) A ve B küplerinin C sonuç küpü ile karsilastirilmasi

$C_L = B_L = 1\,0\,1\,0\,0\,1$ ve

$C_R = B_R = 0\,1\,1\,1\,1\,0$ oldugu için

$C = B$ dir.

C = B oldugu için kesisim isleminin sonucu B küpüdür.

$C = A \cap B = B$

Küpler arasindaki kapsama durumuna bakilirsa,

$C \neq A$ oldugu için A küpü B küpünü kapsar ($A \supset B$)

Örnek 3.4.

A = 1 0 x x 0 0 ve B = 1 x 1 1 0 x iki küp olsun. Bu küpler arasindaki kesisme durumunu ve birbirlerini kapsama durumunu temel bilgisayar islemleri ile gerçeklestirirsek;

A ve B küplerinden $A_L - A_R$ ve $B_L - B_R$ bit dizilerini asagidaki gibi olustururuz.

$A_L = 1\,0\,1\,1\,0\,0$ $B_L = 1\,1\,1\,1\,0\,1$

$A_R = 0\,1\,1\,1\,1\,1$ $B_R = 0\,1\,0\,0\,1\,1$

Elde edilen bit dizilerinden C_L ve C_R bit dizileri dolayisiyla da C küpü elde edilir.

$C_L = A_L \wedge B_L = 1\,0\,1\,1\,0\,0 \wedge 1\,1\,1\,1\,0\,1 = 1\,0\,1\,1\,0\,0$

$C_R = A_R \wedge B_R = 0\,1\,1\,1\,1\,1 \wedge 0\,1\,0\,0\,1\,1 = 0\,1\,0\,0\,1\,1$

$C\,(C_L, C_R) = A \cap B = 1\,0\,1\,1\,0\,0$

i) Sonucun bos küme olup olmadiginin kontrol edilmesi

$D = (C_L \vee C_R) \oplus 11\ldots11 = (1\,0\,1\,1\,0\,0 \vee 0\,1\,0\,0\,1\,1) \oplus 1\,1\,1\,1\,1\,1 = 0$

$D = 0$ oldugu için $C \neq \varnothing$ dir.

Yani bu islemler sonucunda A ve B küplerinin kesisiminden bir deger elde edilmistir.

ii) A ve B küplerinin C sonuç küpü ile karsilastirilmasi

$C_L = A_L = 1\,0\,1\,1\,0\,0$ ve
$C_R = B_R = 0\,1\,1\,1\,1\,0$ oldugu için

$C \neq B$ ve $C \neq A$ dir.

$C \neq B$ ve $C \neq A$ oldugu için kesisim isleminin sonucu her iki küpte degildir. Fakat kesisim elemani olarak bir deger mevcuttur. Bulunan bu deger her iki küp tarafindan da kapsanmaktadir. Bulunan kesisim degeri;

$C\,(C_L, C_R) = A \cap B = 1\,0\,1\,1\,0\,0$ dir.

Küpler arasindaki kapsama durumuna bakilirsa,
Hem $C \neq A$ ve hem de $C \neq B$ oldugu için her iki küpte birbirini kapsamaz.

3.4. Asal Implikantlarin Yerel Sekilde Belirlenmesi

AI'larin belirlenmesi sürecinin karmasikligi S_{ON} kümesindeki her bir minterm ile S_{OFF} kümesindeki her bir mintermin ayrilmasiyla sadelestirilebilir (Nadjafov ve Kahramanov 1973, Kahramanli ve Allahverdi 1993). Var olan bu olasilik asagidaki teorem ile dogrulanmaktadir.

Teorem: Eger $A=a_1a_2...a_j...a_n$ S_{ON} kümesinden herhangi bir minterm ve $B_i= b_1^i b_2^i...b_j^i...b_n^i$ S_{OFF} kümesinden herhangi bir minterm ise $K_i = K_{i-1}$ # B_i, $i=1,2,...,m$, $K_0=xx...x$, prosedürü tarafindan örtülen mintermin AI'nin hesaplanmasi, sadece bütün böyle b_j^i degerlerinin mantiksal ters çevirerek a_j degerine karsilik gelmesi gereklidir ve yeterince korunmalidir.

Ispat: Herhangi bir j koordinati için $a_j=b_j^i$ oldugunu kabul edersek $v_j=x\#b_j^i=\bar{a}_j$ dir. Sonuç olarak, $a_j=b_j^i$ için farkli tanimlanmis küp, (a_j, b_j^i) çiftinde $\bar{a}_j$ koordinatini içerir. Bu da, bu küpün A mintermini içermedigini gösterir. Bu durumda, A mintermini örtmeyen küplerin üretilmesiyle önlenir. b_j^i degerlerinin x sembolüyle degistirilmesi gereklidir. a_j ? x ve b_j^i = x için j koordinati üzerindeki farkli küp biçimlenmemistir. Bundan dolayi, b_j^i nin degeri degismez. a_j = x ve b_j^i?x versiyonu A minterm oldugu için mümkün degildir. Eger $a_j=\bar{b}_j^i$ ise $v_i=x\# b_j^i=\bar{b}_j^i=a_j$ olur. Bundan dolayi, $a_j = \bar{b}_j^i$ durumunda, j koordinati üzerinde A mintermini kesin örten farkli küp biçimlendirilir.

Bu teoremi kullanarak, $B_i = b_1^i b_2^i...b_j^i...b_n^i \in S_{OFF}$ mintermi $Q_i= q_1^i q_2^i...q_j^i...q_n^i$ küpüne asagidaki kurallar kullanilarak dönüstürülebilir.

Eger $b_j^i=x$ ise $q_j^i=x$,

Eger $a_j=b_j^i$ ise $q_j^i=x$,

Eger $b_j^i=\bar{a}_j$ ise $q_j^i=b_j^i$.

3.4.1. Yakin-Minimum Örtme Algoritmasi (Near-Minimal Cover Algorithm)

Bu algoritma, 3.2.2.2 ve 3.2.2.4 de tanimlanan islemlere ve 3.4'de elde edilen kurallara dayanmaktadir. Bu algoritmanin sonucu, sadelestirilmis Boole fonksiyonunun olasi sonuçlarindan bir tanesidir. Sonuç çok sadelesmis veya az sadelesmis olabilir. Bu sonuç, bir veya bir kaç ekstra çarpim teriminin (AND kapisi) önemli olmadigi birçok pratik uygulamada yeterli olabilir. Genellikle, elde edilen sonucun, fonksiyonun en sadelesmis biçimi olmasi S_{ON} kümesindeki mintermlerin siralanmasina baglidir. Ancak, uygun siralamayi elde etmek, düzensiz AI kümesi elde etmekten daha zordur. Herhangi bir sirada verilmis herhangi bir Boole fonksiyon için yakin-minimum örtme algoritmasi asagidaki gibidir:

Algoritma Yakin-Minimum Örtme (S_{ON}, S_{OFF})

Begin

while (S_{ON} ? Æ) do

begin

$\lambda \leftarrow$ ilk_eleman (S_{ON})

$Q_0 \leftarrow$ S_{OFF} genislet (X)

$Q_1 \leftarrow$ dönüsümlü_yutma_islemi (Q_0)

$S_{AI}(x) \leftarrow (xx\ldots xx) \# Q_1$

if {EAI_seçme_islemi ($S_{AI}(x)$)} > 1 **then**

EAI(x) $\leftarrow$ {EAI_seçme_islemi ($S_{AI}(x)$)}nin herhangi bir elemani

else

EAI(x) $\leftarrow$ EAI_seçme_islemi ($S_{AI}(x)$)

$S_{ONi} \leftarrow S_{ON} \# EAI(x)$

$S_{EAI} \leftarrow S_{EAI} \cup EAI(x)$

end

End

Örnek 3.5.

$f(x_3,x_2,x_1,x_0)=\Sigma(0,1,4,5,6,8,10,14)$ Boole fonksiyonunu Yakin-Minimum Algoritmasini kullanarak minimumlastirin.

S_{ON1} = {0000,0001,0100,0101,0110,1000,1010,1110},

S_{OFF} = {0010,0011,0111,1001,1011,1100,1101,1111},

S_{ON} kümesinin ilk mintermi 0000, $\lambda 1$=0000,

1.1. Q0.1 ve Q1.1 kümelerinin belirlenmesi.

Tablo 3.6. 0000 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
0010	xx1x	Asal	xx1x
0011	xx11	xx1x tarafindan kapsandi	
0111	x111	xx1x tarafindan kapsandi	
1001	1xx1	Asal	1xx1
1011	1x11	xx1x tarafindan kapsandi	
1100	11xx	Asal	11xx
1101	11x1	11xx tarafindan kapsandi	
1111	1111	xx1x tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.1 = {xx1x,1xx1,11xx}.

1.2. 0000 minterminin örttügü küp kümesinin belirlenmesi.

S1 = xxxx # Q1 = {((xxxx # xx1x) # 1xx1) # 11xx}

= {(xx0x # 1xx1) # 11xx}

= {(0x0x,xx00) # 11xx}

= {0x0x,0x00,x000}

= {0x0x,x000},

S1 = {S1.1,S1.2} = {0x0x,x000}.

1.3. En büyük küpün belirlenmesi.

$P1.1 = S_{ON1} \# S1.1 = S_{ON1} \# 0x0x$

$= \{0000,0001,0100,0101,0110,1000,1010,1110\} \# 0x0x$

$= \{0110,1000,1010,1110\},$

$P1.2 = S_{ON1} \# S1.2 = S_{ON1} \# x000$

$= \{0000,0001,0100,0101,0110,1000,1010,1110\} \# x000$

$=\{0001,0100,0101,0110,1010,1110\},$

Yukaridaki islemlerde görüldügü gibi P1.1 kümesi (4 eleman) P1.2 kümesinde (6 eleman) daha güçlüdür. Sonuç olarak 0x0x küpü x000 küpünden daha büyüktür. Böylece,

$E1 = 0x0x, \quad SPI = \{0x0x\}.$

2.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

$S_{ON2} = P1.1 = \{0110,1000,1010,1110\},$

Buradan, yeni S_{ON2} kümesinin ilk mintermi 0110, $\lambda 2 = 0110,$

Tablo 3.7. 0110 mintermi için Q0.2 ve Q1.2 kümelerinin belirlenmesi

S_{OFF}	Q0.2	Küp Durumu	Q1.2
0010	x0xx	Asal	x0xx
0011	x0x1	x0xx tarafindan kapsandi	
0111	xxx1	Asal	xxx1
1001	1001	x0xx tarafindan kapsandi	
1011	10x1	x0xx tarafindan kapsandi	
1100	1x0x	Asal	1x0x
1101	1x01	xxx1 tarafindan kapsandi	
1111	1xx1	xxx1 tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.2 = {x0xx, xxx1,1x0x}.

2.2. 0110 minterminin örttügü küp kümesinin belirlenmesi.

S2 = xxxx # Q1 = {((xxxx # x0xx) # xxx1) # 1x0x}

= {(x1xx # xxx1) # 1x0x } = {(x1x0 # 1x0x)} = {01x0,x110},

S2 = {S2.1,S2.2} = {01x0,x110}.

2.3. En büyük küpün belirlenmesi.

P2.1 = S_{ON2} # S2.1 = S_{ON2} # 01x0 = {0110,1000,1010,1110} # 01x0

= {1000,1010,1110},

P2.2 = S_{ON2} # S2.2 = S_{ON2} # x110 = {0110,1000,1010,1110} # x110

= {1000,1010},

Yukaridaki islemlerde görüldügü gibi P2.2 kümesi (2 eleman) P2.1 kümesinde (3 eleman) daha güçlüdür. Sonuç olarak x110 küpü 01x0 küpünden daha büyüktür. Böylece, E2 = x110, SPI = {0x0x,x110}.

3.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

S_{ON3} = P2.2 = {1000,1010},

Buradan, yeni S_{ON3} kümesinin ilk mintermi 1000, $\lambda 3$ = 1000,

Tablo 3.8. 1000 mintermi için Q0.3 ve Q1.3 kümelerinin belirlenmesi

S_{OFF}	Q0.3	Küp Durumu	Q1.3
0010	0x1x	Asal	0x1x
0011	0x11	0x1x tarafindan kapsandi	
0111	0111	0x01 tarafindan kapsandi	
1001	xxx1	Asal	xxx1
1011	xx11	xxx1 tarafindan kapsandi	
1100	x1xx	Asal	x1xx
1101	x1x1	xxx1 tarafindan kapsandi	
1111	x111	xxx1 tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.3 = {0x1x,xxx1,x1xx}.

3.2. 1000 minterminin örttügü küp kümesinin belirlenmesi.

S3 = xxxx # Q1 = {((xxxx # 0x1x) # xxx1) # x1xx}

= {((1xxx,xx0x) # xxx1) # x1xx}

= {(1xx0,xx00) # x1xx}

= {10x0,x000},

S3 = {S3.1,S3.2} = {10x0,x000}.

3.3. En büyük küpün belirlenmesi.

P3.1 = S_{ON3} # 10x0 = {1000,1010} # 10x0 = ∅

P3.2 = S_{ON3} # x000 = {1000,1010} # x000 = {1010}

Yukaridaki islemlerde görüldügü gibi P3.1 kümesi (0 eleman) P3.2 kümesinde (1 eleman) daha güçlüdür. Sonuç olarak 10x0 küpü x000 küpünden daha büyüktür. Böylece, E3 = 10x0, SPI = {0x0x,x110,10x0}.

4.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

S_{ON4} = P3.1 = ∅.

Sonuç olarak, sadelestirme islemi tamamlanmistir. Çünkü S_{ON} kümesindeki bütün mintermler örtülmüstür. Fonksiyonun sadelestirilmis durumu;

SPI = {0x0x,x110,10x0}.

$$f = \bar{x}_3 \bar{x}_1 + x_2 x_1 \bar{x}_0 + x_3 \bar{x}_2 \bar{x}_0$$

Yakin-Minimum Örtme Algoritmasi ile yapilmis, minimumlastirma örnekleri EK A'da gösterilmistir.

3.4.2. Kesin-Minimum Örtme Algoritmasi (Exact-Minimal Cover Algorithm)

Iki seviyeli lojik minimumlastirma, lojik sentezin temel problemlerinden biridir (Sasao ve Butler 2001). Çarpimlarin toplami (SOP) biçimindeki kesin sekilde minimumlastirma probleminin üssel (exponansiyel) niteliginden dolayi, güncel algoritmalar minimum SOP'da yalniz yüze yakin çarpim terimine sahip olan fonksiyonlari isleyebilirler (Mishchenco ve Sasao 2003). Bu nedenle, mutlak (kesin) minimumlastirma metotlarinin sürekli olarak en minimum SOP üretmesine ragmen, pratik uygulamalarin çogu karmasikligi yaklasik olarak çarpim terimlerinin sayisinin karesine esit olan bulussal minimumlastirma (Mishchenco ve Sasao 2003, Tirumalai ve Butler 1991) yöntemlerini kullanirlar. Bu algoritmalar kesin minimumlastirma algoritmalarindan fark edilebilecek sekilde hizlidir fakat çok sayida çarpim terimi olan fonksiyonlar için yavas kalabilirler (Mishchenco ve Sasao 2003). Lojik sentezin optimumlastirma araçlarini hizlandirmak için bulussal algoritmalari tek bir döngüde gerçeklestirmege çalisirlar. Bunun bedeli ise genellikle düsük kaliteli çözüm olur (Mishchenco ve Sasao 2003). Ayrica bulussal algoritmalar ürettikleri sonuçlar açisindan çesitlilik gösterirler. Yani, bütün fonksiyonlar açisindan digerlerinden iyi olan bir bulussal algoritma mevcut degildir. Fakat belli bir bulussal algoritmanin digerlerinden daha iyi yaptigi fonksiyon siniflari vardir (Saad ve Schultz 1988).

Bulussal algoritmalarin gerçeklesme farkliliklari, isleme tabi tutulan mintermi içermekte olan iki ya da daha fazla asal implikant arasindan esas asal implikantin belirlenmesi için kullanilan yetersiz kurallardan dogmaktadir. Bu kurallar ve ilgili problemler asagida ele alinmaktadir. Açiklamalarin kisaligi için esas asal implikanti elde etmek için isleme tabi tutulmak üzere seçilen minterm *Hedef Minterm (Target Minterm-HM)* olarak adlandirilacaktir.

3.4.2.1. EAI seçme kurallari

Genel olarak; bulussal metotlar, minimumlastirilan fonksiyonun S_{ON} kümesi bosalana kadar (fonksiyon tamamiyla kapsanacak) asagida belirtilen adimlari tekrar eden Dogrudan Örtme Yöntemini (Direct Covering Principle-DÖY) kullanir:

1. Bir HM seçmek
2. S_{AI} (x)'i olusturmak
3. EAI seçmek
4. S_{ON}'nun en son durumundan EAI'yi çikarmak

Yaygin bir sekilde bilinen pek çok fonksiyon için minimumlastirma metotlarinin baslica zorlugu, HM'nin birden fazla AI tarafindan kapsandigi durumlarda yukarida belirtilen prosedürün 3. adiminda kullanilan bir karar verme kurali olan Dallandirma ve Sinirlandirma Tekniginde (Branch and Bound Technique-DST) yatmaktadir (Veitch 1952). EAI'nin belirlenmesine uygun olan S_{AI}(x) kümesindeki Kismi Düzenleme Operasyonuna (Partial Ordering Operation-KDO) dayanan DST, R.E.Miller tarafindan açiklanmistir (Miller 1965) ve daha sonra (Pomper ve Armstrong 1981, Besslich 1986, Dueck ve Miller 1987)'nin yazarlari tarafindan biraz gelistirilmistir.

DÖY'e dayali minimumlastirma metotlarinda ilk önce HM'yi içeren tüm AI'lar olusturulur ve ikinci olarak AI'larin bir tanesi KDO'yu kullanarak EAI (Tirumalai ve Butler 1991) olarak belirlenir.

Her bir AI için verimlilik faktörü; söz konusu AI'nin kapsadigi On-mintermlerin sayisinin, o AI'nin maliyetlerine bölünmesi yolu ile hesaplanir. En büyük verimlilik faktörüne sahip olan AI, EAI olarak seçilir. Eger birkaç tane böyle AI varsa ilk olusturulan AI, EAI olarak seçilir. Hatirlatalim ki PLA'larda tüm AI'lar ayni maliyetle gerçeklestirilir. Bu nedenle, hangi AI'nin seçilecegi sadece kapsanilan On-mintermlerin sayisi ile belirlenir.

3.4.2.2. Rasgele minterm seçme prensibi

DÖY'e dayali minimumlastirma metotlarinda KDO kullanmanin ana fikri, $S_{AI}(x)$ belirlendiginden hemen sonra EAI(x)'in seçilmesidir. Bu durumun yetersizligi (eksikligi) asagida belirtilmistir.

HM X için $S_{AI}(x) = \{AI(x)\}_{i=1}^{K}$ kümesinin olusturuldugunu farz edelim. Eger KDO izlenirse, tüm i?k için;

$$AI_k(x) > AI_i(x) \leftrightarrow |S_{ON} \cap AI_k(x)| > |S_{ON} \cap AI_i(x)| \qquad (3.1)$$

Çünkü On-mintermleri en çok sayida örten en büyük AI(x) tek degildir. Fakat bir tanesi, bütün diger AI(x)'ler tarafindan tamamiyla örtülen On-mintermlerin hepsini örtebilir. Kural (3.1) tüm durumlarda kesin sonuca varmayabilir. Asagida verilen örnek 3.6, yukarida bahsedilen kuralin (3.1) yetersizliginin nasil kötü (kesin olmayan) sonuca yol açacagini göstermektedir.

Örnek 3.6.

Kural (3.1) kullanarak rasgele minterm prensibiyle
S_{ON}={0011, 0100, 0101, 0111, 1001, 1101, 1110, 1111} ile verilen tamamiyla tanimlanmamis fonksiyonu minimumlastirmak.

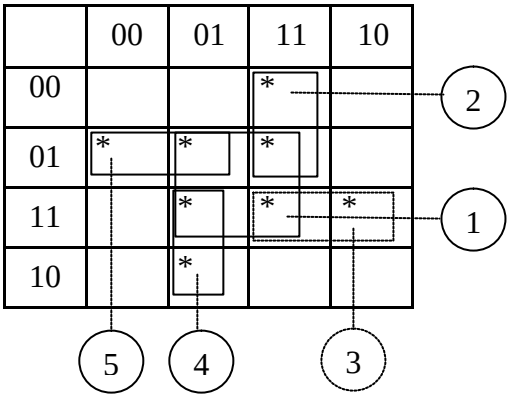
Eger birinci HM için 1101 On-mintermini seçersek, AI_1(1101)=x1x1 ve AI_2(1101)=1x01 asal implikantlari elde edilir. Bundan dolayi,

R(x1x1) = $S_{ON} \cap$ x1x1 = {0011, 0100, 0101, 0111, 1001, 1101, 1110, 1111} # x1x1
= {0101, 0111, 1101, 1111}
R(1x01) = $S_{ON} \cap$ 1x01 = {0011, 0100, 0101, 0111, 1001, 1101, 1110, 1111} # 1x01
= {1001, 1101}

$|R(x1x1|=4$ ve $|R(1x01)|=2$ oldugu için $AI_1(1101)=x1x1$ EAI_1'dir. Bundan dolayi:

$S_{EAI} = \{x1x1\};$

$S_{ON} = S_{ON} \# x1x1 = \{0011, 0100, 1001, 1110\}.$

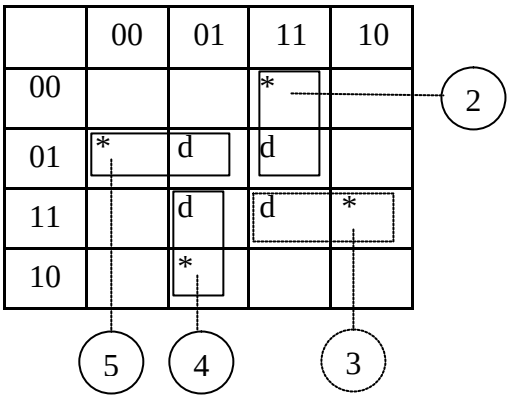


Sekil 3.10. Örnek 3.6 için baslangiç KH

EAI_1 tarafindan kapsanan 1 leri içeren mintermler $S_{DC}=\{0101,0111,1101,1111\}$ belirsiz mintermler kümesi olarak göründü. Sonuç olarak, minimumlastirilan fonksiyonun en son durumu $S_{ON}=\{0011, 0100, 1001, 1110\}$ ve $S_{DC}=\{0101, 0111, 1101, 1111\}$ olarak sunulan tamamiyla tanimlanmamis fonksiyondur. S_{DC} kümesinin elemanlari bir sonraki Karnaugh haritasinda (Sekil 3.11'de) d olarak isaretlenmistir.

Sekil 3.11.'de görüldügü gibi, tüm On-mintermler izole edilmistir ve bu yüzden tüm EAI'lar denetimle belirlenebilir. Buradan;

$EAI_2=0x11,$ $EAI_3=111x,$ $EAI_4=1x01,$ $EAI_5=010x$



Sekil 3.11. Minimumlastirilan fonksiyonun en son durumu için KH

Sonuç olarak;

S_{EAI} = {x1x1, 0x11, 111x, 1x01, 010x}

$S_{ON} = S_{ON}$ # {EAI_2, EAI_3, EAI_4, EAI_5}

S_{ON} = {0011, 0100, 1001, 1110} # {0x11, 111x, 1x01, 010x} = ∅

Hatirlatalim ki, eger ilk HM olarak 0101, 1101 ve 1111'den biri seçilseydi yine ayni sonuç olusacakti. Fakat Petrick dallandirma ve sinirlandirma fonksiyonu örtme Tablosu 3.9'da sekillendirilen (Sasao ve Butler 2001, Rujipattanapong 2001) ;

S_{EAI} = BE(A+E)(A+B)D(A+D)C(A+C)

S_{EAI} = BEDC = BCDE ⇒ {0x11, 111x, 1x01, 010x}

gösteriyor ki yukarida EAI_1 olarak belirtilen AI_1(1101) = x1x1 aslinda bir EAI degildir. Bu nedenle S_{EAI} = {0x11, 111x, 1x01, 010x} dir.

Tablo 3.9.
Örnek 1 için Petrick dallandirma ve sinirlandirma fonksiyonu örtme tablosu

		0011	**0100**	0101	0111	**1001**	1101	**1110**	1111
A	x1x1			+	+		+		+
B	0x11	**+**			+				
C	111x							**+**	+
D	1x01					**+**	+		
E	010x		**+**	+					

Bu örnegin analizi gösteriyor ki kesin sonuca, izole edilmis On-mintermlerden 0011, 0100, 1001 ve 1110 ilk olarak dört HM seçildiginde sadece 4!=24 durumda ulasilirken, en kötü sonuca neden olan siralamanin seçilmesi sonucu 4 x 4! = 96 durumda ulasilabilir. Diger bir ifadeyle, düsüs orani 96/(96+24)=80% dir.

3.4.2.3. Komsuluk faktörlerine dayali minterm siralama

Bölüm 3.4.2.2'de örnegi gösterilen muhtemel en kötü sonuçlardan kaçmak için $S_{ON}=\varnothing$ oluncaya kadar tekrarlanan iyilestirilmis DÖY (Umans 2001, Tirumalai ve Butler 1991) prosedürü asagida verilmistir.

1) Her bir S_{ON} elemani için komsularinin sayisina esit olacak agirligi hesaplamak
2) En düsük agirlikli X'i HM olarak seçmek ve $S_{AI}(x)$ olusturmak
3) AI(X)'lar arasindan bir EAI seçmek
4) S_{ON}'nun kalanindan (son durum) EAI'yi çikarmak

Örnek 3.7.

Baslangiç Karnaugh haritasi Örnek 3.6'dakiyle ayni olan fonksiyonu bitisiklik faktörlerine dayali minterm siralamayi kullanarak minimumlastirin.

Sekil 3.11'de görüldügü gibi;

N(0011) = N(0100) = N(1001) = N(1110) = 1,

N(0101) = N(0111) = N(1101) = N(1111) = 3.

Sonuç olarak düsük agirlikli mintermlerin kümesi S_F={0011,0100,1001,1110} dir.

1. Ilk HM'yi seçmek için minterm $1001 \in S_F$ olsun. HM 1001 örten EAI_1=1x01. Sonuç olarak, S_{EAI} = {1x01};

$S_{ON} = S_{ON}$ # 1x01 = {0011, 0100, 0101, 0111, 1001, 1101, 1110, 1111} # 1x01

S_{ON} = {0011, 0100, 0101, 0111, 1110, 1111}.

Bu örtmeden sonra Karnaugh haritasinin durumu Sekil 3.12 a'da gösterilmistir.

2. Ikinci HM'yi seçmek için minterm $0011 \in S_F$ olsun. HM 0011 örten EAI_2=0x11. Sonuç olarak, S_{EAI} = {0x11, 1x01};

$S_{ON} = S_{ON}$ # 0x11 = {0011, 0100, 0101, 0111, 1110, 1111} # 0x11

S_{ON} = {0100, 0101, 1110, 1111}.

Bu örtmeden sonra Karnaugh haritasinin durumu Sekil 3.12 b'de gösterilmistir.

3. Üçüncü HM'yi seçmek için minterm $0100 \in S_F$ olsun. HM 0100 örten EAI_3=010x. Sonuç olarak, S_{EAI} = {0x11, 1x01, 010x};

$S_{ON} = S_{ON}$ # 010x = {0011, 0100, 0101, {0100, 0101, 1110, 1111} # 010x

S_{ON} = {1110, 1111}.

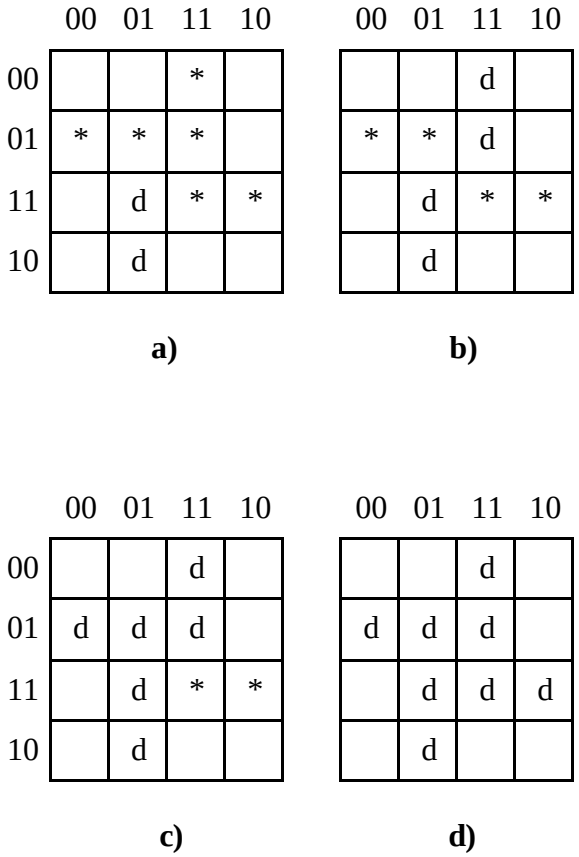
Bu örtmeden sonra Karnaugh haritasinin durumu Sekil 3.12 c'de gösterilmistir.

4. Dördüncü HM'yi zorunlu olarak seçilen minterm $1110 \in S_F$ olsun. HM 1110 örten EAI_4=111x. Sonuç olarak, S_{EAI} = {0x11, 1x01, 010x, 111x}.

$S_{ON} = S_{ON}$ # 111x = {1110, 1111} # 111x = $\varnothing$.

Bu örtmeden sonra Karnaugh haritasinin durumu Sekil 3.12 d'de gösterilmistir.

Minimumlastirma islemi tamamlanmistir. Çünkü S_{ON}'un son durumu bostur. Bu nedenle, S_{EAI} = {0x11,1x01,010x,111x}.



Sekil 3.12. Karnaugh haritasinin
(a) 10x1
(b) 0x11
(c) 010x
(d) 111x
EAI'lari tarafindan kapsandiktan sonraki durumu

Gösterilen metot'un 1. adimini gerçeklestirmek, büyük On-mintermler kümesi için çok uzun zaman alabilir. Çünkü k<1 iken $|S_{ON}|=N=k2^n$ ise

$$WC_1 = \sum_{i=1}^{N-1} i = \frac{1+N-1}{2} \times (N-1) = \frac{N(N-1)}{2} \cong \frac{N^2}{2}$$

$$WC_1 = \frac{(k \times 2^n)^2}{2} = \frac{k^2}{2} \times 2^{2n} \Rightarrow O(2^{2n})$$

kadar karsilastirma bu adimin uygulanmasi için gerek duyulur. Dolayisiyla bu adim genellikle istenmeyen üssel (exponansiyel) zaman karmasikligina sahiptir. Ek olarak, genellikle bu yaklasim rasgele minterm seçme ilkesinden daha iyi sonuçlara neden olabilir. Fakat her zaman kesin minimum sonuçlara ulastirmayabilir. Dolayisiyla ayni yeterlilik faktörüne sahip AI'lar arasindan bir EAI'nin belirlenmesinde belirsizlik vardir.

3.4.2.4. Yeni KDO kurali ve EAI'larin tam belirlenmesini saglamak için HM'lerin islemesini erteleme yöntemi

Eger bir HM'nin ele alinmasinin sonucunda birden fazla AI olusursa en büyük AI'i EAI olarak belirlemek için asagidaki KDO uygulanabilir.

$$AI_1(x) < AI_2(x) \leftrightarrow (S_{ON} \cap AI_1(x)) \subseteq (S_{ON} \cap AI_2(x)) \quad (3.2)$$

(3.1) ve (3.2) numarali kurallar arasindaki farklilik sudur. Kural (3.1) $R1(x)=S_{ON} \cap AI_1(x)$ ve $R2(x)=S_{ON} \cap AI_2(x)$ kümeleri kendi boyutlarinda karsilastirilirken, kural (3.2) bu kümeleri kendi içerikleriyle karsilastirir ve bu sebepten kesin sonuca neden olabilir. AI'larin rasgele seçilmis sayisi için (3.2)'nolu formül asagidaki gibi genisletilebilir.

$$AI_k(x) > \forall AI_{i \neq k}(x) \leftrightarrow (S_{ON} \cap AI_k(x)) \supset \forall (S_{ON} \cap AI_{i \neq k}(x)) \quad (3.3)$$

Bütün diger AI(x)'ler tarafindan sadece ve sadece tüm On-mintermleri tamamen kapsanan $AI_k(x)$ en büyüktür (EAI'dir). Bununla beraber, kosul (3.3)'de $AI_k(x)$'i memnun edici bir durum yoksa belirsizlik durumu vardir. Bu problemi çözmek için $AI_k(x)$ de kosul (3.3) memnun edici durum olusana kadar HM'nin isleme tabi tutulmasinin ertelenmesi önerilir.

Tanim 3.1. Isleme tabi tutulan HM aktif HM (AHM) olarak adlandirilir.

Tanim 3.2. Erteleneni, isleme tabi tutulan HM, ertelenmis HM (EHM) olarak adlandirilir.

Tanim 3.3. $SW(x)=\{X, S_{AI}(x)\}$ veri yapisi EHM X'in isleme tabi tutulmasini sürdürmesi HM X'in durum sözcügü olarak adlandirilir. $S_{AI}(x)=\{AI_2(x),\ldots,AI_m(x)\}$ ifadesi HM X içeren AI'larin kümesidir. Sonuç olarak:

$$SW(x) = \{X, S_{AI}(x)\} = \{X, \{AI_2(x),\ldots,AI_m(x)\}\} \quad (3.4)$$

HM'nin isleme tabi tutulmasi için $AI_k(x)$ kosul (3.3)'ü saglayan ertelenebilen durumlara sahip degildir.

1) AHM'nin isleme tabi tutulmasini durdurmak
2) Durum sözcügü SW(x)'i temsil etmek ve onu S_{SW} kümesine dahil etmek

Örnegin, $S_{ON}(Y) = \{001, 010, 011, 110\}$, X = 011 ve $S_{AI}(x) = \{01x, 0x1\}$ olsun

$R_1(x) = \{001, 010, 011, 110\} \# 01x = \{001, 110\};$

$R_2(x) = \{001, 010, 011, 110\} \# 0x1 = \{010, 110\}$

$R_1(x) \not\subset R_2(x)$ ve $R_2(x) \not\subset R_1(x)$ oldugu için HM X=011 asagidaki gibi ertelenir.

1) HM X = 011 isleme tabi tutulmasi durdurulur.
2) SW(011) = {011,{01x,0x1}} durum sözcügü sekillendirilir.
3) Sekillendirilen SW, S_{SW} kümesine dahil edilir, örnegin $S_{SW} \cup SW(011)$.

Her hangi bir HM'yi erteledikten sonra henüz kapsanmamis bir On-minterm yeni bir HM olarak seçilir.

3.4.2.5. On-Küme kapsama ve EHM'lerin yeniden baslama prosedürü

Prosedür $S_{AI}(x)$ kümesini_elde_etme

Begin

$Q_0 \leftarrow S_{OFF}$ kümesini genislet (X)

$Q_1 \leftarrow$ dönüsümlü_yutma_islemi (Q_0)

$S_{AI}(x) \leftarrow$ (xx…xx) # Q_1

if $|S_{AI}(x)|=1$ **then** iletisim_alani $\leftarrow$ EAI

SW olustur

$S_{SW} \leftarrow$ SW

Geri Dön

End

Prosedür EHM_SW_isleme

Begin

isleme_tabi_tutulmamis $\leftarrow$ bütün SW

bir : SW $\leftarrow$ ilk_isleme_tabi_tutulmamis_SW

EHM X # SW

if (EHM # EAI(x)) = $\varnothing$ **then** sil (aktif SW) ve goto iki

$S_{ONi} \cap$ aktif SW ($S_{AI}(Y)$)

if memnun_edici_durum (AI_k (x)) **then** goto son

isleme_tabi_tutulmamis SW $\leftarrow$ simdiki SW

iki : **if** isleme_tabi_tutulmamis SW **then** goto bir

iletisim_ara_bellegi $\leftarrow$ EAI(x) $\leftarrow$ AI_k (x)

sil (aktif SW)

son : **Geri Dön**

End

Prosedür Örtme

Begin

EAI ← iletisim_alani

bir : S_{ON} ← S_{ON} # EAI

if S_{SW} = ∅ **then** goto son

iletisim_alani ← ∅

Çagir EHM_SW_isleme_prosedürü

if iletisim_alani **?** ∅ **then** goto bir

son : Geri Dön

End

Algoritma Kesin-Minimum Örtme (S_{ON}, S_{OFF})

Begin

bas : if (S_{ON} = ∅) **then** goto dur

HM X ← bir_eleman (S_{ON})

Çagir $S_{AI}(x)$ kümesini_elde_etme_prosedürü

if $|S_{AI}(x)|$=1 **then** goto ilet

Çagir SW biçimlendirme_prosedürü

S_{ON} ← S_{ON} # X

ilet : iletisim_alani ← EAI

Çagir örtme_prosedürü

goto bas

dur : if S_{SW} = ∅ **then** goto son

Çagir Petric_prosedürü

son : Bitir

End

Prosedür Petrick

Begin

bas : S_{SW} (ilk SW)

seç (SW kismi ($S_{AI}(x)$))

if m=| $S_{AI}(x)$ | then AI $\leftarrow$ L_{i1}, L_{i2},…, L_{im}

olustur ($DEF_i = \bigcup_{i=1}^{m} L_{im}$)

sil (aktif SW)

if $S_{SW} \neq \varnothing$ **then** goto bas

genislet_olustur ($PF = \bigcap_{i=1}^{K} IF_i$)

seç (en az L_{im} literal içeren çarpim terimi)

olustur (EAI'lar SS_i alt kümesi)

olustur (minimum_örtme_kümesi-MÖK)

Geri Dön

End

Örnek 3.8.

Asagidaki S_{ON}, S_{DC} ve S_{OFF} kümeleri ile verilen tamamiyla tanimlanmis fonksiyonu minimumlastirin.

$S_{ON} = S_{ON0}$ = {00000, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110}

S_{DC} = {00101, 01011, 00111}

S_{OFF} = {00001, 00100, 00110, 01010, 01111, 10001, 10011, 10100, 10101, 10110, 10111, 11001, 11011, 11100, 11101, 11111}

Tablo 3.10. HM X = 00000 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=00000**
00001	**xxxx1**
001x0	**xx1xx**
01010	**x1x1x**
01111	x1111
100x1	1xxx1
1010x	1x1xx
1011x	1x11x
110x1	110x1
1110x	111xx
11111	11111

E1(00000) = xxxxx # {xxxx1, xx1xx, x1x1x} = xx0x0 # x1x1x = {x00x0, xx000}

$R_{01} = S_{ON0} \cap$ x00x0 = {00000, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ x00x0

R_{01} = {00000, 00010, 10000, 10010}

$R_{02} = S_{ON0} \cap$ xx000 = {00000, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ xx000

R_{02} = {00000, 01000, 10000, 11000}

$R_{01} \not\subset R_{02}$ ve $R_{02} \not\subset R_{01}$ oldugundan EAI için karar vermek mümkün degildir. Bundan dolayi, HM X = 00000 için durum sözcügü biçimi gereklidir.

SW_1 = {00000, {x00x0, xx000}};

$S_{SW} = \{SW_1\}$

S_{ON1} = {<u>00000</u>, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110}

Tablo 3.11. HM X = 00010 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=00010**
00001	**xxx01**
001x0	**xx1xx**
01010	**x1xxx**
01111	x11x1
100x1	**1xxx1**
1010x	1x10x
1011x	1x1xx
110x1	110x1
1110x	1110x
11111	111x1

E2(00010) = xxxxx # {xxx01, xx1xx, x1xxx, 1xxx1} = x00xx # {xxx01, 1xxx1}

E2(00010) = {x001x, x00x0} # 1xxx1 = {0001x, x0010, x00x0}

$R_{21} = S_{ON1} \cap 0001x$ = {<u>00000</u>, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ 0001x

R_{21} = {00010, 00011}

$R_{22} = S_{ON1} \cap x0010$ = {<u>00000</u>, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ x0010

R_{22} = {00010, 10010}

$R_{23} = S_{ON1} \cap x00x0$ = {<u>00000</u>, 00010, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ x00x0

R_{23} = {<u>00000</u>, 00010, 10000, 10010}

$R_{22} \subset R_{23}$ oldugu için AI_2 = x0010 çikarildi. Fakat $R_{21} \not\subset R_{23}$ ve $R_{23} \not\subset R_{21}$ oldugundan HM X = 00010 için durum sözcügü biçimi gereklidir.

SW_2 = {00010, {0001x, x00x0}; S_{SW} = {SW_1, SW_2}

S_{ON2} = {<u>00000</u>, <u>00010</u>, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110}

Tablo 3.12. HM X = 00011 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X**=00011
00001	**xxx0x**
001x0	**xx1x0**
01010	**x1xx0**
01111	**x11xx**
100x1	**1xxxx**
1010x	1x10x
1011x	1x1xx
110x1	110xx
1110x	1110x
11111	111xx

E3(00011) = xxxxx # {xxx0x,1xxxx,xx1x0,x1xx0,x11xx} = {0001x, 0x011, 00x11}

$R_{31} = S_{ON2} \cap 0001x$ = {<u>00000</u>, <u>00010</u>, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ 0001x

R_{31} = {<u>00010</u>, 00011}

$R_{32} = S_{ON2} \cap 0x011$ = {<u>00000</u>, <u>00010</u>, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ 0x011

R_{32} = {00011}

$R_{33} = S_{ON2} \cap 00x11$ = {<u>00000</u>, <u>00010</u>, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ 00x11

R_{33} = {00011}

$R_{32} \subset R_{31}$ ve $R_{33} \subset R_{31}$ oldugu için AI_1=0001x EAI_1 dir. Sonuç olarak, asagidaki gibi örtme prosedürünü gerçeklestirmek gereklidir

$S_{ON3} = S_{ON2}$ # 0001x = {<u>00000</u>, <u>00010</u>, 00011, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} # 0001x

S_{ON3} = {<u>00000</u>,01000,01001,01100,01101,01110,10000,10010,11000,11010,11110}

Sonuç olarak;

$S_{EAI} = S_{EAI} \cup EAI_1$ = {0001x};

S_{ON3} = {<u>00000</u>,01000,01001,01100,01101,01110,10000,10010,11000,11010,11110};

S_{SW} = {SW_1, SW_2}

Yeni bir EAI tanimlandigindan ertelenen HM'ler için SW'leri isleme tabi tutmak gerekir.

S_{SW} = {SW_1, SW_2}

a) SW_1 in islenmesi

SW_1 = {00000, {x00x0, xx000}}

HMP_0 # EAI_1 = 00000 # 0001x = 00000

EAI_1 = 0001x tarafindan HMP_0 örtülmedigi için verilen SW ün kisimlarini AI lar olarak ele almak gerekir.

$R_{01} = S_{ON3} \cap$ x00x0 = {<u>00000</u>, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ x00x0

R_{01} = {<u>00000</u>, 10000, 10010}

$R_{02} = S_{ON3} \cap$ xx000 = {<u>00000</u>, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110} $\cap$ xx000

R_{02} = {<u>00000</u>, 01000, 10000, 11000}

$R_{01} \not\subset R_{02}$ ve $R_{02} \not\subset R_{01}$ oldugundan EAI için karar vermek mümkün degildir. Bundan dolayi, SW_1 degismeden kalir.

b) SW_2 nin islenmesi.

SW_2 = {00010, {0001x, x00x0}

HMP_2 # EAI_1 = 00010 # 0001x = ∅

Görüldügü gibi, EHM_2 = 00010 EAI_1 = 0001x tarafindan örtülmüstür. Bu nedenle SW_2 silinmelidir. Sonuç olarak,

$S_{EAI} = S_{EAI} \cup EAI_1$ = {0001x};

S_{SW} ={SW_1};

S_{ON3} = {00000,01000,01001,01100,01101,01110,10000,10010,11000,11010,11110}

Tablo 3.13. HM X = 01000 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=01000**
00001	**x0xx1**
001x0	**x01xx**
01010	**xxx1x**
01111	xx111
100x1	10xx1
1010x	101xx
1011x	1011x
110x1	**1xxx1**
1110x	**1x1xx**
11111	1x111

E4(01000) = xxxxx # {x0xx1, x01xx, xxx1x, 1xxx1, 1x1xx} = {01x0x, xx000}

$R_{81} = S_{ON3} \cap 01x0x = \{\underline{00000}, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110\} \cap 01x0x$

$R_{81} = \{01000, 01001, 01100, 01101\}$

$R_{82} = S_{ON3} \cap xx000 = \{\underline{00000}, 01000, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110\} \cap xx000$

$R_{82} = \{\underline{00000}, 01000, 10000, 11000\}$

$R_{81} \not\subset R_{82}$ ve $R_{82} \not\subset R_{81}$ oldugundan EAI karar vermek mümkün degildir. Bundan dolayi, HM X = 01000 için durum sözcügü biçimi gereklidir.

$SW_3 = \{01000, \{01x0x, xx000\}\}$; $S_{SW} = \{SW_1, SW_3\}$

Tablo 3.14. HM X = 01001 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=01001**
00001	**x0xxx**
001x0	x01x0
01010	**xxx10**
01111	**xx11x**
100x1	10xxx
1010x	101xx
1011x	1011x
110x1	**1xxxx**
1110x	1x1xx
11111	1x11x

E5(01001) = xxxxx # {x0xxx, xxx10, xx11x, 1xxxx} = {01x0x, 010x1}

$R_{91} = S_{ON3} \cap 01x0x = \{\underline{00000}, \underline{01000}, 01001, 01100, 01101, 01110, 10000, 10010, 11000, 11010, 11110\} \cap 01x0x$

$R_{91} = \{\underline{01000}, 01001, 01100, 01101\}$

$R_{92} = S_{ON3} \cap 010x1 = \{\underline{00000}, \underline{01000}, 01001, 01100, 01101, 01110, 10000, 10010,$
$11000, 11010, 11110\} \cap 01x0x$

$R_{92} = \{01001\}$

$R_{92} \subset R_{91}$ için $AI_{91} = 01x0x$ EAI_2 dir. Bundan dolayi,

$S_{ON4} = S_{ON3} \# 01x0x = \{\underline{00000}, 01110, 10000, 10010, 11000, 11010, 11110\};$
$S_{EAI} = S_{EP2} \cup EAI_2 = \{0001x, 01x0x\};$
$S_{SW} = \{SW_1, SW_3\}.$

Yeni bir EAI tanimlandigindan ertelenen HM ler için SW leri isleme tabi utmak gerekir.

$S_{SW} = \{SW_1, SW_3\}$

a) SW_1 in islenmesi

$SW_1 = \{00000, \{x00x0, xx000\}\}$

$HMP_0 \# EAI_2 = 00000 \# 01x0x = 00000$

$EAI_2 = 01x0x$ tarafindan HMP_0 örtülmedigi için verilen SW ün kisimlarini AI lar olarak isleme tabi tutmak gerekir.

$R_{01} = S_{ON4} \cap x00x0 = \{\underline{00000}, 01110, 10000, 10010, 11000, 11010,11110\} \cap x00x0$
$R_{01} = \{\underline{00000}, 10000, 10010\}$
$R_{02} = S_{ON4} \cap xx000 = \{\underline{00000}, 01110, 10000, 10010, 11000, 11010,11110\} \cap xx000$
$R_{02} = \{\underline{00000}, 01000, 10000, 11000\}$

$R_{01} \not\subset R_{02}$ ve $R_{02} \not\subset R_{01}$ oldugundan EAI karar vermek mümkün degildir. Bundan dolayi, SW_1 degismeden kalir.

b) SW_3 ün islenmesi

SW_3 = {01000, {01x0x, xx000}}

EHM_8 # EAI_2 = 01000 # 01x0x = ∅, 01000 mintermi örtüldü. Bundan dolayi, SW_3 silinmelidir. Sonuç olarak,

S_{ON4} = {<u>00000</u>, 01110, 10000, 10010, 11000, 11010, 11110};
S_{EAI} = {0001x, 01x0x};
S_{SW} = {SW_1}.

Tablo 3.15. HM X = 01110 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=01110**
00001	x0001
001x0	**x0xxx**
01010	**xx0xx**
01111	**xxxx1**
100x1	100x1
1010x	10x0x
1011x	10xxx
110x1	1x0x1
1110x	**1xx0x**
11111	1xxx1

E6(01110) = xxxxx # {x0xxx, xx0xx, xxxx1, 1xx0x} = {011x0, x1110}

$R_{141} = S_{ON4} \cap 011x0$ = {<u>00000</u>, 01110, 10000, 10010, 11000,11010,11110} ∩ x11x0
R_{141} = {01110}
$R_{142} = S_{ON4} \cap x1110$ = {<u>00000</u>, 01110, 10000, 10010, 11000,11010,11110} ∩ x1110
R_{142} = {01110, 11110}

$R_{141} \subset R_{142}$ için AI_{142} = x1110 EAI_3 dür. Bundan dolayi,

$S_{ON5} = S_{ON4}$ # x1110 = {<u>00000</u>, 10000, 10010, 11000, 11010};
S_{EAI} = {0001x, 01x0x, x1110};
$S_{SW} = \{SW_1\}$.

Yeni bir EAI tanimlandigindan ertelenen HM ler için SW leri isleme tabi tutmak gerekir.

$S_{SW} = \{SW_1\}$

a) SW_1 in islenmesi

SW_1 = {00000, {x00x0, xx000}}

HMP_0 # EAI_3 = 00000 # x1110 = 00000

EAI_3 = x1110 tarafindan HMP_0 örtülmedigi için verilen SW ün kisimlarini AI lar olarak ele almak gerekir.

$R_{01} = S_{ON5} \cap$ x00x0 = {<u>00000</u>, 10000, 10010, 11000, 11010} $\cap$ x00x0
R_{01} = {<u>00000</u>, 10000 ,10010 }
$R_{02} = S_{ON5} \cap$ xx000 = {<u>00000</u>, 10000, 10010, 11000, 11010} $\cap$ xx000
R_{02} = {<u>00000</u>, 10000, 11000}

$R_{01} \not\subset R_{02}$ ve $R_{02} \not\subset R_{01}$ oldugundan EAI karar vermek mümkün degildir. Bundan dolayi, SW_1 degismeden kalir.

Tablo 3.16. HM X = 10000 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=**10000
00001	0xxx1
001x0	0x1xx
01010	**01x1x**
01111	01111
100x1	**xxxx1**
1010x	**xx1xx**
1011x	xx1xx
110x1	x1xx1
1110x	x11xx
11111	x1111

E7(10000) = xxxxx # {01x1x, xxxx1, xx1xx} = {1x0x0, x00x0, xx000}

$R_{161} = S_{ON5} \cap 1x0x0 = \{\underline{00000}, 10000, 10010, 11000, 11010\} \cap 1x0x0$

$R_{161} = \{10000, 10010, 11000, 11010\}$

$R_{162} = S_{ON5} \cap x00x0 = \{\underline{00000}, 10000, 10010, 11000, 11010\} \cap x00x0$

$R_{162} = \{\underline{00000}, 10000, 10010\}$

$R_{163} = S_{ON5} \cap xx000 = \{\underline{00000}, 10000, 10010, 11000, 11010\} \cap 1x0x0$

$R_{163} = \{\underline{00000}, 10000, 11000\}$

Burada herhangi bir R_i digerlerini içermediginden EAI için karar vermek mümkün degildir. Bundan dolayi, HM X = 10000 için durum sözcügü biçimi gereklidir.

$SW_4 = \{\{10000, \{1x0x0, x00x0, xx000\}\};$

$S_{SW} = \{SW_1, SW_4\};$

$S_{ON5} = \{\underline{00000}, \underline{10000}, 10010, 11000, 11010\}$

Tablo 3.17. HM X = 10010 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=10010**
00001	0xx01
001x0	0x1xx
01010	**01xxx**
01111	011x1
100x1	**xxxx1**
1010x	xx10x
1011x	**xx1xx**
110x1	x1xx1
1110x	x110x
11111	x11x1

E8(10010) = xxxxx # {01xxx, xxxx1, xx1xx} = {1x0x0, x00x0}

$R_{181} = S_{ON5} \cap$ 1x0x0 = {<u>00000</u>, <u>10000</u>, 10010, 11000, 11010} $\cap$ 1x0x0

R_{181} = {<u>10000</u>, 10010, 11000, 11010}

$R_{182} = S_{ON5} \cap$ x00x0 = {<u>00000</u>, <u>10000</u>, 10010, 11000, 11010} $\cap$ x00x0

R_{182} = {<u>00000</u>, <u>10000</u>, 10010}

Burada herhangi bir R_i digerlerini içermediginden EAI için karar vermek mümkün degildir. Bundan dolayi, HM X = 10010 için durum sözcügü biçimi gereklidir.

SW_5 = {{10010, {1x0x0, x00x0}};

S_{SW} = {SW_1, SW_4, SW_5};

S_{ON5} = {<u>00000</u>, <u>10000</u>, <u>10010</u>, 11000, 11010}

Tablo 3.18. HM X = 11000 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=11000**
00001	00xx1
001x0	001xx
01010	**0xx1x**
01111	0x111
100x1	x0xx1
1010x	x01xx
1011x	x01xx
110x1	**xxxx1**
1110x	**xx1xx**
11111	xx111

E9(11000) = xxxxx # {0xx1x, xxxx1, xx1xx} = {1x0x0, xx000}

$R_{241} = S_{ON5} \cap 1x0x0 = \{\underline{00000}, \underline{10000}, \underline{10010}, 11000, 11010\} \cap 1x0x0$

$R_{241} = \{\underline{10000}, \underline{10010}, 11000, 11010\}$

$R_{242} = S_{ON5} \cap xx000 = \{\underline{00000}, \underline{10000}, \underline{10010}, 11000, 11010\} \cap xx000$

$R_{242} = \{\underline{00000}, \underline{10000}, \underline{11000}\}$

Burada herhangi bir R_i digerlerini içermediginden EAI için karar vermek mümkün degildir. Bundan dolayi, HM X = 11000 için durum sözcügü biçimi gereklidir.

$SW_6 = \{\{11000, \{1x0x0, xx00\}\};$

$S_{SW} = \{SW_1, SW_4, SW_5, SW_6\};$

$S_{ON5} = \{\underline{00000}, \underline{10000}, \underline{10010}, \underline{11000}, 11010\}$

Tablo 3.19. HM X = 11010 içeren bütün AI'lari belirlemek

S_{OFF}	**HM X=11010**
00001	00x01
001x0	001xx
01010	**0xxxx**
01111	0x1x1
100x1	x0xx1
1010x	x010x
1011x	**x01xx**
110x1	**xxxx1**
1110x	**xx10x**
11111	xx1x1

E10(11010) = xxxxx # {0xxxx, x01xx, xxxx1, xx10x} = {1x0x0, 11x10}

$R_{261} = S_{ON5} \cap$ 1x0x0 = {<u>00000</u>, <u>10000,</u> <u>10010</u>, <u>11000</u>, 11010} $\cap$ 1x0x0

R_{261} = {<u>10000</u>, <u>10010</u>, <u>11000</u>, 11010}

$R_{262} = S_{ON5} \cap$ 11x10 = {<u>00000</u>, <u>10000</u>, <u>10010</u>, <u>11000</u>, 11010} $\cap$ 11x10

R_{262} = {11010}

$R_{262} \subset R_{261}$ için AI_1 = 1x0x0 EAI_4 dür. Bu nedenle,

$S_{ON6} = S_{ON5}$ # 1x0x0 = {<u>00000</u>, <u>10000</u>, <u>10010</u>, <u>11000</u>, 11010} # 1x0x0

S_{ON6} = {<u>00000</u>};

S_{EAI} = {0001x, 01x0x, x1110, 1x0x0};

$S_{SW} = \{SW_1, SW_4, SW_5, SW_6\}$.

Yeni bir EAI tanimlandigindan ertelenen HM ler için SW leri isleme tabi tutmak gerekir.

$S_{SW} = \{SW_1, SW_4, SW_5, SW_6\}$

a) SW_1 in islenmesi

SW_1 = {00000, {x00x0, xx000}}

HMP_0 # EAI_4 = 00000 # 1x0x0 = 00000

E1(00000) = xxxxx # {xxxx1, xx1xx, x1x1x} = xx0x0 # x1x1x = {x00x0, xx000}

$R_{01} = S_{ON6} \cap x00x0 = \{\underline{00000}\} \cap x00x0 = \{00000\}$

$R_{02} = S_{ON6} \cap xx000 = \{\underline{00000}\} \cap xx000 = \{00000\}$

$R_{01} = R_{02}$ ve $QC(x00x0) = QC(xx000)$ dir. EAI_5 olarak $AI_1 = x00x0$ veya AI_2=xx000 seçilebilir. Fakat yukaridaki algoritmayi korumak için SW_1 degistirilmemis olarak birakalim.

$SW_1 = \{00000, \{x00x0, xx000\}\}$

b) $SW_4 = \{\{10000, \{1x0x0, x00x0, xx000\}\};$

$HMP_{16} \# EAI_4 = 10000 \# 1x0x0 = \varnothing,$ oldugu için SW_4 silinmelidir.

c) $SW_5 = \{\{10010, \{1x0x0,x00x0\}\}$

$HMP_{18} \# EAI_4 = 10010 \# 1x0x0 = \varnothing,$ oldugu için SW_5 silinmelidir.

d) $SW_6 = \{\{11000, \{1x0x0,xx00\}\};$

$HMP_{24} \# EAI_4 = 11000 \# 1x0x0 = \varnothing,$ oldugu için SW_6 silinmelidir.

Sonuç olarak;

$S_{EAI} = \{0001x, 01x0x, x1110, 1x0x0\};$

$S_{ON7} = \{\underline{00000}\};$

$S_{SW} = \{SW_1\}.$

Görüldügü gibi S_{ON7} de islenme tabi tutulmamis minterm kalmamistir ve S_{SW} kümesi sadece SW_1 i içermektedir. Bu yüzden, SW_1 in kisimlarindaki AI lari (x00x0+xx000) Petrick fonksiyonu formunda tek dahil etme fonksiyonunu biçimlendirmek gereklidir. Çünkü, baska dahil etme fonksiyonu yoktur ve çoklu degildir. S_{EAI} kümesinin elemanlarinin çarpilmasi ile çoklu yapilir. Fakat bunu yapmak için S_{EAI} dan EAI lari 0001x, 01x0x, x1110, 1x0x0 ve SW_1 in kisimlarindaki x00x0, xx000 AI lari P_1, P_2, P_3, P_4, P_5, P_6 olarak gösterelim. Sonra,

$PF = P_1 P_2 P_3 P_4 (P_5 + P_6) = (P_1 P_2 P_3 P_4 P_5) + (P_1 P_2 P_3 P_4 P_6)$

Sonuç olarak, iki tane kesin minimum örtü elde edilir :

$S_{EAI1} = \{0001x, 01x0x, x1110, 1x0x0, x00x0\}$

$S_{EAI2} = \{0001x, 01x0x, x1110, 1x0x0, xx000\}$

4. SUNULAN YÖNTEMIN MEVCUT YÖNTEMLER ILE KARMASIKLIGA GÖRE KARSILASTIRILMASI

4.1. Karmasiklik (Complexity)

Karmasilik bir algoritmanin çok sayida parametre karsisinda maliyet davranisindaki degisikligi/artmayi gösteren kavramsal bir ifadedir (Çölkesen 2002).

Büyük O gösterimi matematiksel bir gösterim olup fonksiyonlarin asimptotik davranislarini tarif etmek için kullanilir. Daha açik sekilde anlatmak gerekirse, bir islevin büyümesinin asimptotik üst sinirini daha basit baska bir fonksiyon cinsinden tanimlanmasi demektir. Bu gösterim ilk olarak Alman sayilar kuramcisi Paul Bachmann tarafindan 1892 yilinda yazdigi *Analytische Zahlentheorie* kitabinda kullanilmistir. Gösterim bir baska Alman matematikçi olan Edmund Landau tarafindan yaygin kullanima sokulmustur. Bundan ötürü bazen Landau sembolü olarak da anilir. Büyük O, Ingiliz dilindeki "order of" yani bir seyin derecesi anlamina gelen söz öbegini hatirlatmak amaci ile kullanilmistir (Vikipedi, Stockmeyer 1990).

Karmasikligi ifade edebilmek için asimtotik ifadeler kullanilmaktadir; bu amaçla O(g(*n*)), Θ(g(*n*)), Ω(g(*n*)), o(g(*n*)) gibi tanimlara basvurulur. Genel olarak büyük O ve Θ notasyonlari daha çok kullanilmaktadir. Herbiri, sonuçta, Tablo 4.1'de büyük O notasyonuna dayanilarak verilen bazi temel fonksiyonlara yaklasir. Örnegin, bir siralama algoritmasinin karmasikligi O(n^2) ise, bunun anlami, n çok büyük degerlere giderken algoritmanin zaman maliyeti karesel olarak artar seklindedir. Örnegin karmasikligi $O(n \log_2 n)$ olan bir siralama algoritmasi O(n^2) olana göre daha iyidir; n çok büyük degerlere giderken karmasiklik dogrusal çarpanli logaritmik olarak artar (Çölkesen 2002, Bovet ve Crescenzi 1994, Stockmeyer 1990). O gösterimi, fonksiyonlarin artis oraninin üst sinirini belirler. O(f(*n*)), f(*n*) fonksiyonundan daha hizli artmayan fonksiyonlar kümesini gösterir. Büyüklük derecesini problemin boyutuna bagli olarak fonksiyonda en hizli artis gösteren terim belirler. Örnek olarak:

$$f(n) = n^4 + 100\,n^2 + 10\,n + 50 = O(n^4)$$

fonksiyonunda *n*'in derecesi n^4'tür. Yani *n*'in büyük degerleri için fonksiyonu en fazla n^4 etkiler. Peki, daha düsük dereceli deyimlere ne olmaktadir? *n*'in çok büyük degerleri için n^4, $100n^2$'den, *10n*'den ve *50*'den çok büyük olacagindan daha düsük dereceli terimler dikkate alinmayabilir. Bu diger terimlerin, islem süresini etkilemedikleri anlamina gelmez. Bu yaklasim yapildiginda *n*'in çok büyük degerlerinde önem tasimadiklari anlamina gelir (Ugur 2002).

Bir programin performansi genel olarak programin isletimi için gerekli olan bilgisayar zamani ve bellegidir. Bir programin zaman karmasikligi (time complexity) programin yürütülme süresidir. Bir programin yer karmasikligi (space complexity) programin yürütüldügü sürece bu program ve veriler için gerekli olan yer miktaridir. Bir problemin çözümünde, kullanilabilecek olan algoritmalardan en etkin olani seçilmelidir. En kisa sürede çözüme ulasan veya en az islem yapan algoritma tercih edilmelidir. Bazi durumlarda da en az bellek harcayan algoritmanin tercih edilmesi gerekebilir. Daha kisa sürede biten bir algoritma yazmak için daha çok kod yazmak veya daha çok bellek kullanmak gerekebilir (Ugur 2002).

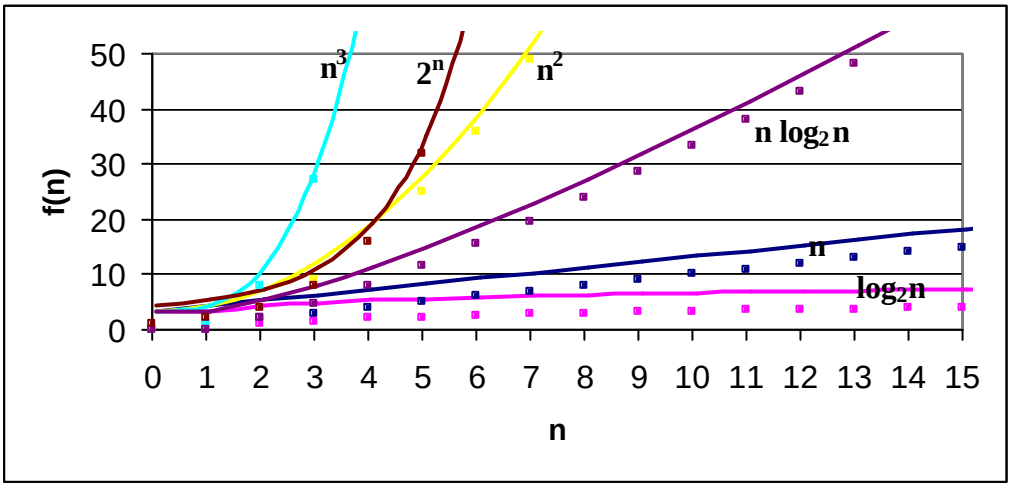
Tablo 4.1. Büyük O ifadeleri ve anlamlari

Büyük O	**Degisim Sekli**
O(1)	Sabit
O(*log n*)	Logaritmik
O(n)	Dogrusal
O(*n log n*)	Dogrusal çarpanli logaritmik
O(n^2)	Karesel
O(n^3)	Kübik
O(2^n)	Iki tabanindan üssel
O(10^n)	On tabanindan üssel
O(*n!*)	Faktöriyel

Tablo 4.1'de gösterilen ifadelerin alabilecekleri degerler Tablo 4.2'de gösterilmistir. Bu degerlerin grafik olarak gösterimi de Sekil 4.1'de verilmistir.

Tablo 4.2. Çesitli fonksiyonlarin aldiklari degerler

n	*O(n)*	*O(log n)*	*O(n log n)*	*O(n^2)*	*O(n^3)*	*O(2^n)*
1	1	0	0	1	1	2
2	2	1	2	4	8	4
3	3	1,6	4,8	9	27	8
4	4	2	8	16	64	16
5	5	2,3	11,5	25	125	32
6	6	2,6	15,6	36	216	64
7	7	2,8	19,6	49	343	128
8	8	3	24	64	512	256
9	9	3,2	28,8	81	729	512
10	10	3,3	33	100	1000	1024
11	11	3,5	38,5	121	1331	2048
12	12	3,6	43,2	144	1728	4096
13	13	3,7	48,1	169	2197	8192
14	14	3,8	53,2	196	2744	16384
15	15	3,9	58,5	225	3375	32768



Sekil 4.1. Tablo 4.2'deki fonksiyonlarin degerlerinden elde edilen grafik

4.2. Algoritmalarin Karmasiklik Degerlendirmesi

Bu bölümde Yakin Minimum Örtme Metodunun asal implikantlari olusturan kisminin karmasikligi karsilastirmali olarak hesaplanacaktir. Çünkü bunlarin esas asal implikant belirleme ve örtme kisimlari var olanlar ile aynidir.

Verilen On-küpü örtmek için asal implikantlarin yeterli bir kümesini olusturan belli bulussal metotlar dogrudan örtme prensibini kullanarak asagidaki algoritmaya göre çalisirlar (Fišer ve Hlavicka 2003, Umans 2001).

1. Yeterli asal implikantlar kümesini elde etmek için bir tane On küp seçmek
2. Bu On küpü örten implikantlari üretmek
3. Verilen implikanti, henüz dokunulmamis literallerden (koordinat degerleri) bir tanesini çikararak (x ile degistirerek) genisletmek
4. Sonucu, Off kümesi ile kesistirmek
5. Eger kesisme isleminin sonucu bos degilse çikarilan literali geri koymak
6. Eger isleme tabi tutulmayan bir literal varsa 3. adima geri dönmek, degilse devam etmek
7. Genisletilen implikanti, asal implikantlardan bir tanesi olarak kaydetmek
8. Eger bütün implikantlar genisletildiyse 11. adima gitmek degilse devam etmek
9. Henüz isleme tabi tutulmamis implikantlardan birisini yeni implikant olarak kabul etmek
10. 3. adima geri dönmek
11. Örtme problemini çözmek

Bu algoritma S_{ON} kümesi bosalincaya kadar tekrarlanir.

Bu algoritmaya dayanarak, verilen terimin literalleri (veya koordinatlari) birer birer etkisiz duruma getirilir. Bu islem söz konusu terim asal implikant oluncaya kadar sürdürülür. Bu islemler polinomal zamanda (Umans 2001) yapilabilir. Bu, matematiksel olarak asagidaki gibi formüle edilebilir.

On-küpü i esas ve n-i esas olmayan (removed) koordinatla ifade edebilmek amaciyla genisletebilmek için i tane incelemeye ihtiyaç vardir. Bundan dolayi,

genisletilen küpün koordinatlarinin hepsi için gereken incelemelerin toplam sayisi asagidaki gibi ifade edilebilir;

$$Q = \sum_{i=1}^{n} i = \frac{1+n}{2} \times n = \frac{n^2 + n}{2} \tag{4.1}$$

Fakat genellikle, incelenen bir küp $0 \leq m < n$ koordinatlari çikarildiktan sonra asal implikant olur. Bunu dikkate alarak, incelemelerin beklenen sayilarinin genel ifadesi (Tablo 4.3'ün üçüncü sütunu) Tablo 4.3'ün birinci ve ikinci sütunlarinda sunulan verilerin endüktif genellemesiyle (Johnsonbaugh ve Schaefer 2004) elde edilmistir. Yani, (4.1) deki formül yerine asagidaki formül kullanilacaktir.

$$Q_0 = n(m+1) - \frac{m^2 + m}{2} \tag{4.2}$$

Varsayalim ki, minimumlastirilan n degiskenli bir fonksiyon, On-kümesi için $K_1 \times 2^n$ boyutuyla ve Off-kümesi için $K_2 \times 2^n$ boyutuyla gösterilmis olsun.

Tablo 4.3. n ve m degerlerine bagli olarak genisletilen küpün incelemelerinin sayisi

Esas olmayan koordinatlarin sayisi (m)	Gereken incelemenin sayisi (Q_0)	Q_0'in genel ifadesi
0	n	$Q_0 = n \times (m+1) - \sum_{i=1}^{m} i = n \times (m+1) - \frac{m^2 + m}{2}$
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-1)+(n-2)+(n-3)+(n-4) = 5n-10	
….	….	
Herhangi bir m için	n+(n-1)+(n-2)+(n-3)+(n-4)+…+(n-m) $= n \times (m+1) - (1 + 2 + ... + m)$	

Incelenen On-küplerin maksimum muhtemel sayisi, $a \le K_1$ oldugu durumlarda $a \times 2^n$ dir ve genisletilmis küp ile bire bir karsilastirilan Off-küplerin sayisi $K_2 \times 2^n$ dir. Sonuç olarak, On-küplerin genisletilmesi metoduna dayanarak bir asal implikantin olusturulmasinin karmasikligi asagidaki formül ile ifade edilebilir.

$$Q_1 = a2^n Q_0 K_2 2^n = aK_2 2^{2n}(2n-m)(m+1) \tag{4.3}$$

Fakat genellikle, bir minterm (0-boyutlu küp) birden fazla asal implikant tarafindan örtülebilir. Bu nedenle bulussal (heuristic) metotlarin çogu *d*'nin baslangiç boyutunun her bir terimi için *n-d* adete kadar asal implikant üretir (Fišer ve Hlavicka 2003). Her bir izole edilmis minterm bir tek asal implikant tarafindan örtüldügünden, bir minterm için olusturulmus asal implikantlarin ortalama sayisi yaklasik olarak *(n-d)/2* ye esit varsayilabilir. Bütün mintermler için *d* sifir oldugundan, verilen mintermi örten yeterli sayida asal implikantlarin (Fišer ve Hlavicka 2003) olusturulmasinin toplam karmasikligi, asagidaki formül ile hesaplanabilir:

$$Q_2 = anK_2 2^{2n-1}(2n-m)(m+1) \tag{4.4}$$

Diger taraftan, burada önerilen metot asagidaki algoritmadaki gibi çalisir.

1. Asal implikantlar kümesinin tamamini elde etmek için gerekli bir tane On küp (minterm) seçmek. Bunu genisletici olarak kullanmak
2. Genisletilmesi gereken küp olarak birinci Off-küpü seçmek
3. Verilen Off-küpü genisletmek (genisletme kuralina göre)
4. Genisletilen küpü, önceden genisletilmis olanlarla bire bir karsilastirmak. Eger genisletilen küp digerlerini içeriyorsa, kapsanan küpleri silmek. Veya önceden üretilmis küplerden birisi bu küpü içeriyorsa bu küpü silmek
5. Eger bütün Off-küpler isleme tabi tutulduysa 8. adima gitmek degilse devam etmek
6. Henüz isleme tabi tutulmamis bir sonraki Off-küpü seçmek
7. 3. adima geri dönmek

8. n-küp den genisletilmis asal küpler kümesini çikartmak
9. Asal olmayan küpleri silmek
10. Örtme problemini çözmek.

Bu algoritma S_{ON} kümesi bosalincaya kadar tekrarlanir.

Bu algoritmaya göre, verilen On-küpü örten tüm asal implikantlarin kümesini üretmede Off-küplerin hepsinin genisletilmesi için On-küp ile bire bir karsilastirilmasi gereklidir. Bundan dolayi, her bir $a \times 2^n$ On-küp, genisletilen $K_2 \times 2^n$ Off-küp ile karsilastirilir. Bununla, yeni genisletilmis Off-küp önceden genisletilmis olanlarla asal olmayanlarin silinmesi için karsilastirilir. Yapilan uygulamalar göstermistir ki buna benzer karsilastirmalarin sayisi n/2'yi geçmemektedir. Sonuç olarak, sunulan metodun karmasikliginin en kötü zaman degeri:

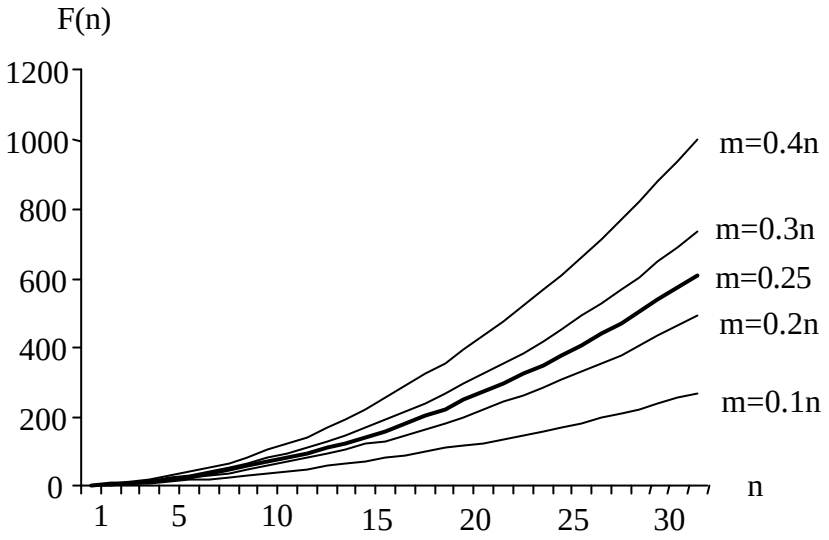
$$Q_3 = a2^n K_2 2^n (1+\frac{n}{2}) = a(2+n)K_2 2^{2n-1} \tag{4.5}$$

Bu yüzden, burada sunulan asal implikant üretme algoritmasinin var olan herhangi bir algoritmaya göre asagidaki deger kadar daha hizli gerçeklestirilebilir olmasini beklemek mümkündür.

$$F(n,m) = \frac{Q_2}{Q_3} = \frac{anK_2 2^{2n-1}(2n+m)(m+1)}{a(n+2)K_2 2^{2n-1}}$$

$$F(n,m) = \frac{Q_2}{Q_3} = \frac{n(2n+m)(m+1)}{n+2} \tag{4.6}$$

Daha fazla görsellik için, Sekil 4.2'de *m*'nin farkli degerlerine karsilik gelen *F(n)*'nin egrileri gösterilmistir. Yaklasik olarak *m*'nin ortalama degerinin *0.25n* oldugu (Fišer ve Hlavicka 2003)'nin gerçeklestirmis olduklari deneylerde gösterilmistir.



Sekil 4.2. n ve m degerleri artarken F(n,m) çok hizli artar

4.3. Yöntemlerin Karsilastirilmasi

Gelistirilmis olan Yakin Minimum Örtme Algoritmasi (YMÖA) ve Kesin Minimum Örtme Algoritmasi (KMÖA), ESPRESSO-II algoritmasi ile karsilastirilmistir. Karsilastirma kriteri olarak üç ana durum belirlenmistir. Bunlar;

- Algoritmalarin çözüm sonucunda bulduklari çarpim terimlerinin toplam ifadesinin sayisi (SOP Sayisi),
- Algoritmalarin çözüme ulasma süreleri,
- Algoritmalarin çözüme ulasirken kullandiklari bellek kapasitesi

Gerçeklestirilen bu karsilastirmalar asagidaki Tablo 4.4, Tablo 4.5 ve Tablo 4.6'da verilmistir.

Gerçeklestirilen algoritmalar C programlama dilinde kodlanmislardir. ESPRESSO-II algoritmasi da C programlama dilinde kodlanmistir. Bütün algoritmalar ayni dosya formatini kullanmistir. Yani YMÖA, KMÖA ve ESPRESSO algoritmalari için ayni benchmarklar kullanilmistir. Algoritmalari ayni sartlarda karsilastirabilmek için ESPRESSO algoritmasinin belirledigi durumlar dikkate alinmistir. Karsilastirmalar tek çikisli fonksiyonlar kullanilarak yapilmistir. Benchmarklarin karsilastirilmasi için tam tanimlanmamis ve tam tanimlanmis fonksiyonlar kullanilmistir. Fonksiyonlarin tam tanimlanmis veya tam tanimlanmamis fonksiyonlar oldugunu belirtmek için ESPRESSO-II algoritmasinin *.type* seçenegi kullanilmistir. Bu seçenekte *.type [fdr]* fonksiyonun durumunu belirlemektedir. Bu seçenekteki f on-kümesi için, r off-kümesi için ve d belirsizler kümesi için kullanilmaktadir.

Karsilastirmalarin gerçeklestirilmesini kolaylastirmak için Delphi programlama dilinde ara yüz programi yazilmistir.

Asagidaki tablolarda benchmarklara ait, giris degisken sayisi, S_{ON} sayisi, S_{OFF} sayisi, SOP sayisi, algoritmalarin sadelestirme zamanlari ve kullandiklari bellek kapasiteleri verilmistir.

Tablo 4.4. Standart MCNC benchmarklari için SOP sayisi

Benchmarklar	Degisken sayisi	S_{ON} sayisi	S_{OFF} sayisi	SOP Sayisi				
				YMÖA (S_Y)	KMÖA (S_K)	ESPRESSO (S_E)	$\frac{S_E}{S_Y}$	$\frac{S_E}{S_K}$
check	04	4	9	1	1	1	1	1
check1	04	4	8	1	1	1	1	1
check2	04	4	6	1	1	1	1	1
check3	04	8	5	2	2	2	1	1
wim	04	9	1	4	4	4	1	1
p82	05	11	13	4	4	4	1	1
squar	05	9	23	2	2	2	1	1
m	06	17	8	4	4	4	1	1
m5	06	27	5	4	4	4	1	1
new2	06	3	4	2	2	2	1	1
poperom	06	56	8	7	7	7	1	1
sqr	06	18	46	2	2	2	1	1
inc	07	12	22	6	6	6	1	1
linrom	07	65	63	24	23	24	1	1,04
max128	07	29	99	8	8	8	1	1
max3	07	12	116	7	7	7	1	1
sqn	07	48	48	8	8	8	1	1
z5xp1	07	25	103	3	3	3	1	1
dist	08	53	203	12	12	12	1	1
e	08	65	128	21	20	20	0,95	1
ex5	08	33	223	2	2	2	1	1
exp	08	18	52	3	3	4	1,33	1,33
exps	08	65	131	21	20	20	0,95	1
exp1	08	24	42	3	3	6	2	2
f51m	08	128	128	23	23	23	1	1

Tablo 4.4'ün devami

Benchmarklar	Degisken sayisi	S_{ON} sayisi	S_{OFF} sayisi	SOP Sayisi				
				YMÖA (S_Y)	KMÖA (S_K)	ESPRESSO (S_E)	$\frac{S_E}{S_Y}$	$\frac{S_E}{S_K}$
m3	08	98	30	16	14	14	0,88	1
m4	08	223	26	26	22	23	0,88	1,05
mlp4	08	32	224	9	9	9	1	1
root	08	15	241	4	4	4	1	1
rd84	08	120	136	84	84	84	1	1
apex4	09	4	434	4	4	4	1	1
max4	09	38	8	6	5	6	1	1,2
min	09	87	46	6	6	6	1	1
max512	09	267	245	10	10	10	1	1
prom1	09	488	14	23	19	19	0,83	1
prom2	09	142	145	7	7	7	1	1
max1024	10	516	508	4	4	4	1	1
t10	10	134	189	48	47	49	1,02	1,04
t11	10	266	371	83	76	76	0,92	1
t12	10	266	758	109	98	99	0,91	1,01
br11	12	29	5	3	3	3	1	1
br1	12	26	8	5	4	4	0,8	1
br2	12	29	6	2	2	2	1	1
t3	12	27	121	6	6	6	1	1
pdc	16	29	1886	4	4	4	1	1
spla	16	67	2036	31	29	29	0,93	1
den	18	24	3	4	4	4	1	1
bca	26	15	13	1	1	1	1	1

Performans ve sonuç kalitesini karsilastirmak için standart MCNC benchmarklari sunulan algoritmalar ve ESPRESSO tarafindan sadelestirilmistir. Karsilastirmalar AMD Sempron 2600 islemcili 1.83 GHz ve 512 MB RAM bellegi olan standart bir PC'de gerçeklestirilmistir. Tablo 4.4'de verilen kirk sekiz farkli tek-çikisli fonksiyon kullanilmistir. Fonksiyonlara ait olan degisken sayilari, S_{ON} sayilari ve S_{OFF} sayilari tablolarda verilmistir. Çarpim terimlerinin toplami ifadesi seklinde verilen sonuçlar (SOP sayisi) açisindan algoritmalar karsilastirildiginda Tablo 4.4'den elde edilen bilgiler söyledir. Yakin Minimum Örtme Algoritmasi ile ESPRESSO-II algoritmasi karsilastirildiginda; fonksiyonlarin %75'inde esit sayida SOP sayisina sahip olduklari görülmüstür. Bu algoritmalardan ESPRESSO, fonksiyonlarin %18,75'inde daha iyi sonuç bulurken YMÖA %6,25'inde daha iyi sonuç bulmustur. SOP sayilarinin ortalama degerlerine göre YMÖA ile ESPRESSO'yu karsilastirdigimizda ESPRESSO'nun daha iyi sonuç buldugu fonksiyonlarda ortalama %9,7 daha az SOP bulmustur. YMÖA'nin daha iyi sonuç buldugu fonksiyonlarda ortalama %45 daha az SOP bulmustur. Burada ESPRESSO algoritmasinin daha iyi sonuç buldugu görülse de YMÖA ile ESPRESSO algoritmalarinin çözüme ulasma yöntemleri farklidir. YMÖA olabilecek ihtimal sonuçlari bulurken ESPRESSO algoritmasi kesin olan sonuçlari bulmaya çalismaktadir. Bu sartlarda dahi YMÖA'nin daha iyi sonuç buldugu fonksiyonlarin olmasi bu algoritmanin güçlü ve gelistirilebilecek yönlerinin oldugunu göstermektedir.

Algoritmalar esit sartlarda karsilastirildiginda, yani ESPRESSO algoritmasi ile Kesin Minimum Örtme Algoritmasinin kesin sonuç (exact) bulma özelliklerine göre karsilastirma yapildiginda fonksiyonlarin %14,6'sinda KMÖA daha iyi sadelestirme yapmistir. Fonksiyonlarin %85,4'ünde ise algoritmalar esit sayida SOP sayisi bulmuslardir. Kesin sonuç bulma durumlarina göre bakildiginda ESPRESSO algoritmasinin daha iyi sonuç buldugu fonksiyon bulunmamaktadir. SOP sayilarinin ortalama degerlerine göre KMÖA ile ESPRESSO'yu karsilastirdigimizda KMÖA'nin ortalama %11 daha iyi sonuç buldugu görülmektedir.

Tablo 4.5. Standart MCNC benchmarklari için çalisma zamanlari

Benchmarklar	Degisken sayisi	S_{ON} sayisi	S_{OFF} sayisi	Çalisma Zamani (milisaniye)				
				YMÖA (T_Y)	KMÖA (T_K)	ESPRESSO (T_E)	$\frac{T_E}{T_Y}$	$\frac{T_E}{T_K}$
check	04	4	9	15,625	15,625	15,625	1	1
check1	04	4	8	15,625	15,625	15,625	1	1
check2	04	4	6	15,625	15,625	15,625	1	1
check3	04	8	5	15,625	15,625	15,625	1	1
wim	04	9	1	15,625	15,625	31,25	2	2
p82	05	11	13	15,625	15,625	15,625	1	1
squar	05	9	23	15,625	31,25	31,25	2	1
m	06	27	5	15,625	31,25	40,625	2,6	1,3
m5	06	27	5	15,625	31,25	40,625	2,6	1,3
new2	06	3	4	15,625	31,25	40,625	2,6	1,3
poperom	06	56	8	15,625	31,25	56,25	3,6	1,8
sqr	06	18	46	31,25	46,875	56,25	1,8	1,2
inc	07	12	22	15,625	31,25	56,25	3,6	1,8
linrom	07	65	63	15,625	31,25	40,625	2,6	1,3
max128	07	29	99	31,25	46,875	56,25	1,8	1,2
max3	07	12	116	15,625	31,25	56,25	3,6	1,8
sqn	07	48	48	31,25	31,25	40,625	1,3	1,3
z5xp1	07	25	103	15,625	15,625	40,625	2,6	2,6
dist	08	53	203	15,625	31,25	65,625	4,2	2,1
e	08	65	128	15,625	31,25	65,625	4,2	2,1
ex5	08	33	223	15,625	31,25	81,25	5,2	2,6
exp	08	18	52	31,25	46,875	65,625	2,1	1,4
exps	08	65	131	31,25	46,875	81,25	2,6	1,73
exp1	08	24	42	15,625	31,25	65,625	4,2	2,1
f51m	08	128	128	15,625	31,25	81,25	5,2	2,6

Tablo 4.5'in devami

Benchmarklar	Degisken sayisi	S_{ON} sayisi	S_{OFF} sayisi	Çalisma Zamani (milisaniye)				
				YMÖA (T_Y)	KMÖA (T_K)	ESPRESSO (T_E)	$\frac{T_E}{T_Y}$	$\frac{T_E}{T_K}$
m3	08	98	30	31,25	46,875	116,25	3,72	2,48
m4	08	223	26	31,25	593,75	337,5	10,8	0,57
mlp4	08	32	224	31,25	31,25	81,25	2,6	2,6
root	08	15	241	31,25	31,25	65,625	2,1	2,1
rd84	08	120	136	46,875	125	186,25	3,97	1,49
apex4	09	4	434	31,25	31,25	100,625	3,22	3,22
max4	09	38	8	31,25	265,625	287,5	9,2	1,08
min	09	87	46	15,625	31,25	81,25	5,2	2,6
max512	09	267	245	15,625	46,875	100,625	6,44	2,15
prom1	09	488	14	31,25	9187,5	3586,875	114,8	0,39
prom2	09	142	145	15,625	31,25	131,875	8,44	4,22
max1024	10	516	508	15,625	46,875	116,25	7,44	2,48
t10	10	113	201	46,875	1484,375	715,625	15,27	0,48
t11	10	266	371	46,875	1234,375	650	13,87	0,53
t12	10	266	758	62,5	218,75	256,25	4,1	1,17
br11	12	29	5	15,625	468,75	456,25	29,2	0,97
br1	12	26	8	15,625	109,375	156,25	10	1,43
br2	12	29	6	15,625	31,25	140,625	9	4,5
t3	12	27	121	15,625	31,25	140,625	9	4,5
pdc	16	29	1886	31,25	46,875	184,375	5,9	3,93
spla	16	67	2036	46,875	296,875	284,375	6,07	0,96
den	18	24	3	31,25	1328,125	537,5	17,2	0,4
bca	26	15	13	31,25	812,5	468,75	15	0,58

Tablo 4.5'de görüldügü gibi, bu benchmarklar sunulan algoritmalar (YMÖA-KMÖA) ve ESPRESSO tarafindan sadelestirilmistir. Sunulan algoritmalarin ve ESPRESSO'nun sadelestirme islemlerini yaparken ihtiyaç duyduklari zaman açisindan degerlendirilmesi Tablo 4.5'de gösterilmistir. Bu degerlendirmeye göre; Yakin Minimum Örtme Algoritmasi ile ESPRESSO Algoritmasini karsilastirildiginda YMÖA'sinin sadelestirme islemlerini çok daha hizli gerçeklestirdigi görülmektedir. Fonksiyonlarin %89,6'sinda YMÖA daha hizli bir sekilde sadelestirme yapip sonuca ulasmistir. %10,4'ünde ise algoritmalarin sonuca ulasma zamanlari esittir. Bu iki algoritma açisindan bakildiginda YMÖA'si ESPRESSO algoritmasina göre çok daha hizlidir. Ortalama olarak YMÖA ESPRESSO'ya göre 7,9 kat daha hizli sadelestirme yapmaktadir. Bu algoritmalar için sonuca ulasma süresine bakildigi zaman, genellikle, sunulan algoritmanin göreceli hizi $|S_{OFF}|/|S_{ON}|$ degerinin azaldigi durumlarda artmaktadir. Örnegin; 9 degiskene sahip olan *prom1* benchmarkinin $|S_{OFF}|/|S_{ON}|=14/488 = 0.03$ oldugu durumda sunulan algoritma ESPRESSO'dan 115 kat daha hizli sadelestirme yapmaktadir. Bunun yaninda, 10 degiskene sahip olan *t12* benchmarkinin $|S_{OFF}|/|S_{ON}| = 758/266 = 2,85$ oldugu durumda sunulan algoritma ESPRESSO'dan 4 kat daha hizli sadelestirme yapmistir.

Kesin Minimum Örtme Algoritmasi ile ESPRESSO Algoritmasi karsilastirildigi zaman, fonksiyonlarin %12,5'inde her iki algoritmada esit zamanda sadelestirme islemini gerçeklestirmistir. Fonksiyonlarin %16,7'sinde ESPRESSO algoritmasinin daha hizli sonuca ulastigi görülmüstür. Fonksiyonlarin büyük bir bölümünde ise KMÖA'sinin daha hizli oldugu görülmüstür. KMÖA fonksiyonlarin %70,8'inde daha hizli sadelestirme yapmistir. KMÖA'nin daha hizli sadelestirme yaptigi fonksiyonlarda ortalama olarak 2 kat daha hizli oldugu belirlenmistir. ESPRESSO'nun hizli oldugu fonksiyonlarda ise ortalama olarak 0,64 kat daha hizli oldugu görülmüstür. ESPRESSO algoritmasinin hizli oldugu fonksiyonlarin hepsinde SOP sayisi dikkate alinarak degerlendirme yapilirsa KMÖA ya daha iyi sonuç bulmus ya da esit sayida SOP ifadesi bulmustur.

Tablo 4.6. Standart MCNC benchmarklari için bellek kullanim durumlari

Benchmarklar	Degisken sayisi	S_{ON} sayisi	S_{OFF} sayisi	Bellek Kullanimi (kilobyte)				
				YMÖA (M_Y)	KMÖA (M_K)	ESPRESSO (M_E)	$\frac{M_E}{M_Y}$	$\frac{M_E}{M_K}$
check	04	4	9	2.064	2.457	2.482	1,20	1,01
check1	04	4	8	1.995	2.375	2.441	1,22	1,03
check2	04	4	6	2.318	2.310	2.449	1,06	1,06
check3	04	8	5	1.802	2.330	2.469	1,37	1,06
wim	04	9	1	1.946	2.306	2.478	1,27	1,07
p82	05	11	13	2.314	1.950	2.428	1,05	1,25
squar	05	9	23	2.290	2.314	2.437	1,06	1,05
m	06	27	5	1.954	1.921	2.449	1,25	1,27
m5	06	27	5	2.269	1.790	1.982	0,87	1,11
new2	06	3	4	2.310	2.322	2.670	1,16	1,15
poperom	06	56	8	2.249	2.322	2.666	1,19	1,15
sqr	06	18	46	2.281	1.942	2.461	1,08	1,27
inc	07	12	22	1.937	1.831	2.428	1,25	1,33
linrom	07	65	63	2.277	2.294	2.457	1,08	1,07
max128	07	29	99	2.285	2.281	2.154	0,94	0,94
max3	07	12	116	2.310	1.925	2.056	0,89	1,07
sqn	07	48	48	2.310	2.322	2.256	0,98	0,97
z5xp1	07	25	103	2.281	2.298	2.416	1,06	1,05
dist	08	53	203	1.945	2.314	2.404	1,24	1,04
e	08	65	128	1.908	2.318	2.379	1,25	1,03
ex5	08	33	223	2.298	1.864	2.408	1,05	1,29
exp	08	18	52	2.298	2.322	2.379	1,04	1,02
exps	08	65	131	2.298	2.322	2.007	0,87	0,86
exp1	08	24	42	1.966	2.326	2.416	1,23	1,04
f51m	08	128	128	2.318	2.294	2.437	1,05	1,06

Tablo 4.6'nin devami

Benchmarklar	Degisken sayisi	S_{ON} sayisi	S_{OFF} sayisi	Bellek Kullanimi (kilobyte)				
				YMÖA (M_Y)	KMÖA (M_K)	ESPRESSO (M_E)	$\frac{M_E}{M_Y}$	$\frac{M_E}{M_K}$
m3	08	98	30	2.285	2.310	2.387	1,04	1,03
m4	08	223	26	2.248	2.187	2.396	1,07	1,1
mlp4	08	32	224	1.941	2.326	2.469	1,27	1,06
root	08	15	241	2.040	2.326	2.404	1,18	1,03
rd84	08	120	136	2.019	1.970	2.142	1,06	1,09
apex4	09	4	434	2.298	2.298	2.379	1,04	1,04
max4	09	38	8	2.306	2.322	2.379	1,03	1,02
min	09	87	46	2.302	2.322	2.379	1,03	1,02
max512	09	267	245	2.278	2.334	2.347	1,03	1,01
prom1	09	488	14	1.872	2.183	2.887	1,54	1,32
prom2	09	142	145	2.302	1.781	2.289	0,99	1,29
max1024	10	516	508	2.232	1.970	2.301	1,03	1,17
t10	10	113	201	2.318	2.015	2.818	1,22	1,4
t11	10	266	371	2.244	2.183	2.408	1,07	1,1
t12	10	266	758	2.318	2.342	2.441	1,05	1,04
br11	12	29	5	1.937	2.310	2.338	1,21	1,01
br1	12	26	8	2.281	2.215	2.375	1,04	1,07
br2	12	29	6	2.244	2.277	1.957	0,87	0,86
t3	12	27	121	2.286	2.302	2.379	1,04	1,03
pdc	16	29	1886	2.338	1.868	2.289	0,98	1,23
spla	16	67	2036	2.220	2.224	2.428	1,09	1,09
den	18	24	3	2.273	2.277	2.396	1,05	1,05
bca	26	15	13	2.281	1.945	2.363	1,04	1,21

Algoritmalarin sadelestirme yaparken kullandiklari bellek alani bakimindan degerlendirilmesi yapildiginda, ESPRESSO'nun YMÖA'na göre %16,7 fonksiyonda daha iyi oldugu görülmesine ragmen %83,3 fonksiyonda YMÖA daha az bellek alani kullanmistir. Algoritmalarin daha az bellek alani kullandiklari fonksiyonlardaki durumlarina bakildiginda ise YMÖA %13,2 daha az bellek alani kullanirken ESPRESSO %9 daha az bellek alani kullanmistir. KMÖA'sinda ise ESPRESSO %8,3 fonksiyonda iyi olmasina karsin %91,7 fonksiyonda KMÖA daha az bellek alani kullanmistir. Algoritmalarin daha az bellek alani kullandiklari fonksiyonlardaki durumlarina bakildiginda ise KMÖA %11,1 daha az bellek alani kullanirken ESPRESSO %9 daha az bellek alani kullanmistir.

5. SONUÇ VE ÖNERILER

5.1. Sonuç

Tez çalismasinda anahtarlama fonksiyonlarini sadelestirmek için iki tane yeni yöntem sunulmustur. Bu yöntemler; Yakin-Minimum Örtme Algoritmasi (YMÖA) ve Kesin-Minimum Örtme Algoritmasi (KMÖA)'dir. Sunulan bu yöntemler ayri ayri kullanilabilecegi gibi, YMÖA, KMÖA'nin alt programi olarak da çalisabilmektedir.

Sunulan yöntemlerde küp cebri islemleri kullanilmaktadir. Sunulan algoritmalarda küp cebrinin koordinatli çikarma, koordinatli kesisim ve dönüsümlü yutma islemleri kullanilmistir. Bu islemlerin gerçeklestirilmesi seri bir sekilde yapilmaktadir. Seri gerçeklesen bu islemler çözüme ulasma süresini artirmaktadir. Bu islemlerden koordinatli kesisim ve dönüsümlü yutma islemleri temel bilgisayar islemleri üzerinden paralel bir sekilde gerçeklestirilmistir. Bu sayede algoritmalarin daha hizli bir sekilde çözüme ulasmalari saglanmistir. Çünkü küp cebri islemlerini gerçeklestirebilmek için yapilacak karsilastirmalarin hepsi bit bit yapilmaktadir. Temel bilgisayar isleri üzerinden gerçeklestirildiginde ise sayilarin karsilastirilmasi yapilmistir. Veya sonuçlarin elde edilmesinde Ve (And), Veya (Or), Degil (Not), Veya Degil (Exor) lojik islemleri kullanilmistir. Bu sayede bit bit karsilastirma yapmaktan kaçinilmistir.

Sunulan yöntemlerden YMÖA da verilen fonksiyonun ON kümesi mintermlerinden bir tanesini rasgele seçilmekte ve bu mintermi kapsayan asal implikantlar (AI) olusturulmaktadir. YMÖA "Büyük veya Daha Az" islemi kullanilarak esas asal implikantlar (EAI) bir bir seçilmektedir. Belirlenen asal implikant için esit sayida minterm örtülürse üretilmis AI'lardan bir tanesi seçilmektedir. Bu islemlerin yapilmasi ile fonksiyonun sadelesmis halini temsil edecek esas asal implikantlar belirlenmis olur. Sunulan YMÖA önemli bir sekilde var olan metotlardan hizli çalismaktadir ve daha az bellek kapasitesine ihtiyaç duymaktadir. Çünkü minimum sayida geçici sonuçlar üreterek isleme tabi tutmaktadir. Bu özellikler sunulan yönetimi özlü ve son derece verimli yapmaktadir.

KMÖA, YMÖA'ni alt program olarak kullanabilir. KMÖA da da verilen fonksiyonun ON kümesi mintermlerinden bir tanesini rasgele seçilmekte ve bu mintermi kapsayan asal implikantlar (AI) olusturulmaktadir. Bazi durumlarda isleme tabi tutulmus On-küpün birden fazla asal implikant tarafindan örtüldügü görülebilir. Bu durumda ek olarak, kismi düzenleme operasyonu (özel bir çesit dallandirma ve sinirlandirma teknigi) olarak adlandirilan esas asal implikant belirleme prosedürü kullanilir. Bu islemlerin yapilmasi ile fonksiyonun sadelesmis hali belirlenmis olur. Sunulan KMÖA önemli bir sekilde var olan metotlardan hizli çalismaktadir ve daha az bellek kapasitesine ihtiyaç duymaktadir. Çünkü minimum sayida geçici sonuçlar üreterek isleme tabi tutmaktadir. Ayni zamanda KMÖA sadelestirme sonucunda elde edilen SOP sayilari dikkate alindiginda da diger algoritmalara göre daha iyi sadelestirme yapmaktadir. Bu özellikler sunulan yönetimi özlü ve son derece verimli yapmaktadir. Birkaç tane paralel bit-islemleri yolu ile küp islemlerinin gerçeklestirilmesi sunulan yöntemi daha esnek ve hizli yapmaktadir. Bu yöntem, Off-kümesinin önemli derecede On-kümesinden daha büyük oldugu fonksiyonlarda nispeten daha yavas çalisabilmektedir.

Gelistirilen algoritmalardan Yakin Minimum Örtme Metodunun asal implikantlari olusturan kisminin karmasikligi karsilastirmali olarak hesaplanmistir. Çünkü bunlarin esas asal implikant belirleme ve örtme kisimlari var olanlar ile aynidir.

Sunulan algoritmalar C programlama dilinde kodlanmistir. Karsilastirmasi yapilacak olan ESPRESSO programi da C programlama dilinde kodlanmistir. Sunulan algoritmalarda ve karsilastirmasi yapilan ESPRESSO programinda ayni dosya yapisi kullanilmistir. Programlarin kullanimini kolaylastirmak için Delphi programlama dilinde ara yüz programi yazilmistir. Karsilastirmalarda tek çikisli fonksiyonlar kullanilmistir. Karsilastirmasi yapilan fonksiyonlar tam tanimlanmamis veya tam tanimlanmis fonksiyonlardir. Algoritmalarin karsilastirmasi üç duruma göre yapilmistir. Bunlar, algoritmalarin verilen fonksiyonlari sadelestirdikten sonra elde ettikleri çarpim terimlerinin toplami (SOP) sayisina göre, algoritmalarin sadelestirme zamanlari ve bellek kullanma durumlaridir.

Performans ve sonuç kalitesini karsilastirmak için kirk sekiz farkli tek-çikisli fonksiyon sunulan algoritmalar ve ESPRESSO tarafindan sadelestirilmistir.

Fonksiyonlara ait olan degisken sayilari, S_{ON} sayilari ve S_{OFF} sayilari tablolarda verilmistir.

Çarpim terimlerinin toplami ifadesi seklinde verilen sonuçlar (SOP sayisi) açisindan algoritmalar karsilastirildiginda YMÖA ile ESPRESSO algoritmasi sonuçlarinda; fonksiyonlarin %75'inde esit sayida SOP sayisina sahip olduklari, ESPRESSO programinin fonksiyonlarin %18,75'inde daha iyi SOP sayisi buldugu, YMÖA'ninda fonksiyonlarin %6,25'inde daha iyi SOP sayisi buldugu görülmüstür. Algoritmalarin bulduklari SOP sayilarina göre; ESPRESSO'nun daha iyi sonuç buldugu fonksiyonlarda ortalama %9,7 daha az SOP bulurken, YMÖA'nin daha iyi sonuç buldugu fonksiyonlarda ortalama %45 daha az SOP bulunmustur.

ESPRESSO algoritmasi ile KMÖA karsilastirildiginda fonksiyonlarin %14,6'sinda KMÖA daha iyi sadelestirme yaptigi görülmüstür. Fonksiyonlarin %85,4'ünde ise algoritmalar esit sayida SOP sayisi bulmuslardir. Kesin sonuç bulma durumlarina göre bakildiginda ESPRESSO algoritmasinin daha iyi sonuç buldugu fonksiyon bulunmamaktadir. SOP sayilarinin ortalama degerlerine göre KMÖA ile ESPRESSO'yu karsilastirdigimizda KMÖA'nin ortalama %11 daha iyi sonuç buldugu görülmektedir.

Sunulan algoritmalarin ve ESPRESSO'nun fonksiyonlari sadelestirme zamanlarina göre karsilastirildiginda; YMÖA'sinin sadelestirme islemlerini çok daha hizli gerçeklestirdigi görülmektedir. Fonksiyonlarin %89,6'sinda YMÖA daha hizli bir sekilde sadelestirme yapip sonuca ulasmistir. %10,4'ünde ise algoritmalarin sonuca ulasma zamanlari esittir. Ortalama olarak YMÖA ESPRESSO'ya göre 7,9 kat daha hizli sadelestirme yapmaktadir.

KMÖA ile ESPRESSO karsilastirildigi zaman, fonksiyonlarin %12,5'inde her iki algoritmada esit zamanda sadelestirme islemini gerçeklestirmistir. Fonksiyonlarin %16,7'sinde ESPRESSO algoritmasinin daha hizli sonuca ulastigi görülmüstür. Fonksiyonlarin %70,8'inde KMÖA tarafindan daha hizli sadelestirme yapilmistir. KMÖA'nin daha hizli sadelestirme yaptigi fonksiyonlarda ortalama olarak 2 kat daha hizli oldugu belirlenmistir. ESPRESSO'nun hizli oldugu fonksiyonlarda ise ortalama olarak 0,64 kat daha hizli oldugu görülmüstür.

Algoritmalarin kullandiklari bellek alani bakimindan karsilastirildiginda, ESPRESSO'nun YMÖA'na göre %16,7 fonksiyonda daha iyi oldugu görülmesine

ragmen %83,3 fonksiyonda YMÖA tarafindan daha az bellek alani kullanilmistir. Algoritmalarin daha az bellek alani kullandiklari fonksiyonlardaki durumlarina bakildiginda ise YMÖA %13,2 daha az bellek alani kullanirken ESPRESSO %9 daha az bellek alani kullanmistir. KMÖA'sinda ise ESPRESSO %8,3 fonksiyonda iyi olmasina karsin %91,7 fonksiyonda KMÖA daha az bellek alani kullanmistir. Algoritmalarin daha az bellek alani kullandiklari fonksiyonlardaki durumlarina bakildiginda ise KMÖA %11,1 daha az bellek alani kullanirken ESPRESSO %9 daha az bellek alani kullanmistir.

5.2. Öneriler

Bu tez çalismasinda gelistirilen algoritmalar tek çikisli (single output) fonksiyonlara uygulanmistir. Bu algoritmalar gelistirilerek çok çikisli (multiple valued) fonksiyonlara uygulanabilir. Bu çalismanin bir sonraki adimi olarak çok çikisli fonksiyonlar için bu algoritmalar gelistirilebilir. Bu algoritmalarin çok çikisli fonksiyonlar için gelistirilmesi ile çok çikisli diger algoritmalarla çözüme ulasma zamanlari, kullandiklari bellek alani ve SOP sayilari durumlarina göre karsilastirilabilirler. Tek çikisli fonksiyonlar ile çok çikisli fonksiyonlarin ortak noktalari arastirilarak bu algoritmalarin verimlilikleri incelenebilir.

KMÖA da ve YMÖA da verilen fonksiyonun S_{ON} kümesinden hedef mintermi seçerken rasgele seçim yapilarak bu mintermi kapsayacak asal implikantlar bulunmaktadir. KMÖA daki hedef mintermi seçme prosedürü gelistirilebilir. Hedef mintermi seçerken izole edilmis mintermler belirlenebilir ve daha sonra bunlarin arasindan bir tanesi seçilebilir. Hedef mintermi seçmek için baska bir prosedür olarak komsuluk faktörleri dikkate alinarak gelistirilebilir. Bu prosedür için önce bütün mintermler için komsuluk faktörleri hesaplanir. Daha sonra en düsük komsuluk faktörüne sahip olan minterm seçilir. Bu islem, sonucun kesin olmasi istenen durumlarda iyi sonuçlar vermesi beklenirken sadelestirme zamani açisindan da kötü sonuçlar ortaya koyabilir.

Bu tez çalismasinda sunulan algoritmalarda küp cebri islemleri kullanilmistir. Bu islemler seri bir sekilde gerçeklestirildigi için bunlardan koordinatli kesisim ve

dönüsümlü yutma islemleri temel bilgisayar isleri üzerinden gerçeklestirilerek paralel bir duruma getirilmistir. Bu paralel islemler sayesinde algoritmalar önemli bir sekilde hizlanmistir. Burada paralel duruma getirilmeyen koordinatli çikarma islemi üzerinde çalismalar yapilarak bu isleminde temel bilgisayar islemleri üzerinden gerçeklestirilmesi saglanabilir. Bu isleminde paralel bir sekilde yapilmasi sonucunda algoritmalarin hizlarinda nasil bir degisme oldugu incelenebilir.

6. KAYNAKLAR

Akers, S.B., 1978, Functional Testing with Binary Decision Diagrams. In Eighth Annual Conference on Fault –Tolerant Computing, pages 75-82.

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

Allahverdi N., 1999, Procedure For Definition Of Neighborhood in Faulty Hypercube Multiprocessor. Mathematical and Computational Applications, Vol. 4, No.1, pp. 17-23.

Allahverdi N.M., Kahramanli S.S., Erciyes K., 2000, A Fault Tolerant Routing Algorithm Based On Cube Algebra For Hypercube Systems. Journal of Systems Architecture 46, pages 201-205.

Arevalo Z. and Bredeson, J.G., 1978, A Method To Simplify A Boolean Function Into A Near Minimal Sum-Of-Products For PLA. IEEE Trans. On Comp., Vol.C-27, No.11, pp.1028-10239, November.

Bartlett K.A., Brayton R.K., Hachtel G.D., Jacoby R.M., Morrison C.R., Rudell R.L., Sangiovanni-Vincentelli A., Wang A.R., 1988, Multilevel Logic Minimization Using Implicit Don't Cares. IEEE Transactions On Computer Aided Design, Vol. 7, No:6, June.

Basçiftçi F., Kahramanli S., Tütüncü K., Saraçoglu R., 2003, Quine Mccluskey Lojik Fonksiyonlari Sadelestirme Metodu, II. Ulusal Meslek Yüksekokullari Sempozyumu, 15-17 Ekim, sayfalar:365-378, Ege Üniversitesi, Izmir.

Beckert B., Iiahnle R., Escalada-Imaz G., 1997, Simplification of Many-Valued Logic Formulas Using Anti-Links. http://citeseer.nj.nec.com/cachedpage /221052/1

Bern J., Meinel C., Slobodova A., 1995, Efficient OBDD based Boolean Manipulation in CAD beyond Current Limits. The Proceedings of the Design Automation Conference, sayfalar 408-413, Haziran.

Bernasconi A., Ciriani V., Luccio F., Pagli L., 2001, Fast Three-Level Logic Minimization Based on Autoymmetry. http://citeseer.nj.nec.com/cachedpage /462188/1

Besslich P.W., 1986, Heuristic minimization of MVL functions:A direct cover approach. IEEE Trans. Comput, pp. 134-144, Feb.

Bovet D.P., Crescenzi P., 1994, Introduction to the Theory of Complexity. Prentice-Hall, Englewood 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 Syst., 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, Hardbound, Kluwer Academic Publishers.

Brayton R.K., Rudell R., Sangiovanni-Vincentelli A., Wang A.R., 1987, MIS: A Multiple-Level Logic Optimization Systems. IEEE Trans. on Computer-Aided Design, CAD-6(6), pp.1062-1081.

Brayton R.K., Somenzi F., 1989, Minimization of Boolean Relations. IEEE Int. Symp. on Circuits and Systems, pp. 738-743.

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

Brown, D.W., 1981, A State-Machine Synthesizer – SMS. Proc. 18th Design Automation Conference, pp.301-304, Nashville, June.

Brown S., Vranesic Z., 2003, Fundamentals of Digital Logic with Verilog Design. McGraw Hall, Chapter 4.

Bryant, R.E., 1986, Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans. On Computer, 35 (8) : 677-691, August.

Bryant R.E., 1995, Binary :Decision Dyagrams and Beyond; Enabling Technologies for Formal Verification. The Proceedings International Conference on Computer Aided Design, sayfalar 236-243, Kasim.

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

Ciesielski M.J., Yang S., Perkowski M.A., 1989, Multiple-valued minimization based on graph coloring. Proc. ICCD '89, pp.262-265.

Coudert O., Madre J. C., 1993, Towards a symbolic logic minimization algorithm. Proc. VLSI Design, January.

Coudert O., 1994, Two-Level Logic Minimization: an overview. Integration, the VLSI Journal, 17-2, pp. 97-140, October.

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

Çirpan H.A., 1992, Lojik Fonksiyonlarin Bilgisayarla Basitlestirilmesi Için Algoritmalar. Yüksek Lisans Tezi, Fen Bilimleri Enstitüsü, Istanbul Üniversitesi.

Çölkesen R., 2002, VeriYapilari ve Algoritmalar. Papatya Yayincilik, Mayis 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-5, No:1, Jaunary.

Dietmeyer D.L., 1979, Logic Design of Digital Systems. Boston, MA:Ally and Bacon.

Drechster R., Sarabi A., Theobald M., Becker B., Perkowski M., 1994, Efficient Representation and Manipulation of Switching Functions based on Ordered Kronecker Functional Decision Diagrams. The Proceedings of the Design Automation Conference, sayfalar 415-419, Haziran.

Drechsler R., Theobald M., Becker B., 1996. Fast OFDD-Based Minimization of Fixed Polarity Reed-Muller Expressions. IEEE Transactions On Computers, Vol. 45, No. 11, Pages:1294-1299

Dueck G.W. and Miller D.M., 1987, A direct cover MVL minimization using the truncated sum. in Proc. 17th Int. Symp. Multiple-Valued Logic, May, pp. 221-227.

Fišer P., Hlavicka J., 2003, BOOM - A Heuristic Boolean Minimizer. Journal of Computing and Informatics, 22 (June), pp. 1001-1033.

Gergov J., Meinel C., 1994, Efficient Boolean Manipulation with OBDDs can be extended to FBDDs. IEEE Transactions on Computers, 43(10):1197-1209, Ocak.

Gurunath B., Biswas N.N., 1989, An Algorithm for Multiple Output Minimization. IEEE Transactions On Computer Aided Design, Vol. 8, No:9, September.

Günes S., 2000, Hiperküp Paralel Islem Sisteminde Arizaya Toleransli Veri Iletimi Yöntemlerinin Analizi Ve Simülasyonu. Doktora Tezi, Selçuk Üni. Fen Bilimleri Enstitüsü.

Günes S., Yilmaz N., Allahverdi N., 2003, A Fault-Tolerant Multicast Routing Algorithm Based on Cube Algebra for Hypercube Networks. The Arabian Journal for Science and Engineering(AJSE), 28(1B), pp. 95-103.

Hong, S.J., Cain, 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, September.

Jain J., Mukherjee R., Fujita M., 1995, Advanced Verification Techniquesbased on Leraning, The Proceedings of the Design Automation Conference, sayfalar 420-426, Haziran.

Jacob J. and Mishchenko A., 2001, Unate decomposition of Boolean functions. Proc. IWLS '01, pp. 66-71.

Johnsonbaugh R., Schaefer M., 2004, Algorithms. Pearson Prentice Hall.

Kahramanli 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.

Kahramanli S., Allahverdi N., 1996, An Algebraic Approach to Transformations on Hypercube System. Mathematical and Computational Applications, 4(1), pp. 50-59.

Kahramanli, S., Özcan, M., 2002, Lojik Tasarimin Temelleri ve Uygulamalari. Atlas Yayin Dagitim. Istanbul.

Kahramanli S., Basçiftçi F., 2003, Boolean Functions Simplification Algorithm Of O(n) Complexity. Mathematical & Computational Applications, Volume 8 Numbers 1-3, pages:271-278. ISSN:1300-686X

Kahramanli S., Basçiftçi F., Savran I., 2005, O(N) Karmasikliginda Anahtarlama Fonksiyonlarini Sadelestirme Algoritmasi. 4. Uluslararasi Ileri Teknolojiler Sempozyumu, Selçuk Üniversitesi, Konya, 28-30, Eylül, sayfalar: 214-219

Kang S. and VanCleemput W.M., 1981, Automatic PLA Synthesis From a DDL-P description. Proceeding of the 18th Design Automation Conf., pp.391-397, June.

Kannurao S., Falkowski B.J., 2003, Single Variable Symmetry Conditions in Boolean Functions Through Reed-Muller Transform. IEEE, pages:680-683.

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

Kebschull U., Schubert E., Rosentiel W., 1992, Multilevel Logic based on Functional Decision Diagram. The Proceedings European Design Automation Conference, sayfalar 43-47.

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

Lafferty J., Vardy A., 1999, Ordered Binary Decision Diagrams and Minimal Trellises. IEEE Transaction on Computers, Vol. 48, No. 9, pages:971-986.

Lee, C.Y., 1959, Representation of switching circuits by binary decision programs. Bell System Technical Journal, 38 (4) : 985-999, June.

Liaw H., Lin C., 1992, On The OBDD-Representation of General Boolean Functions. IEEE Transaction on Computers, Vol. 41, No. 6, pages:661-664.

Linn B., Somenzi F., 1990, Minimization of Symbolic Relations. IEEE Int. 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 Diyagrams in a Logic Synthesis Environment. The Proccedings International Conference on Computer-Aiaded Design, 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-10(4), pp. 413-426.

Mano M. M., 1984, Digital Design, Prentice-Hall International Editions.

Mano, M.M., 2002, Sayisal Tasarim. Literatür Yayincilik, Istanbul.

McCluskey, E.J., 1956, Minimization of Boolean Functions. Bell System Technical Journal, Vol. 35, No.5, pp.1417-1444.

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 Sanghavi, Robert Brayton, Alberto Sangiovanni Vincentelli, 1986, Espresso-Signature: A New Exact Minimizer for Logic Functions. University of California at Berkeley, Berkeley, CA 94720.

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

McGeer P., Sanghavi J., Brayton R., Sangiovanni-Vincentelli A., 1993, Espresso-Signature: A new exact minimizer for logic functions. Proc. DAC ’93, pp. 618-624.

Miller, R.E., 1965, Switching Theory. Vol. 1 Combination Circuits, New York; John Wiley and sons.

Minato, S. 1992, Fast generation of irredundant sum-of-products forms from binary decision diagrams. Proc. SASIMI'92, pp. 64-73.

Mishchenco A., Sasao T., 2003, Large-Scale SOP minimization Using Decomposition and Functional Properties. DAC, June 2-6, pp149-154.

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

Nguyen K., Perkowski M., Goldstein N., 1987, Palmini-Fats Boolean Minimizer for Personal Computers. Proc. Design Automation Conf., pp. 615-621, Aug.

Ochi H., Ishiura N., Yajima Breadth S., 1991,First Manipulation of SBDD of Boolean Functions for Vector Processing. The Proceedings of the Design Automation Conference, sayfalar 413-416, Haziran.

Ochi H., Yasuoka K., Yajima S., 1993, Breadth-First Manipulation of Very Large Binary Decision Diagrams. The Proceedings International Conference on Computer Aided Design, sayfalar 48-55, Kasim.

Perkins S.R., Rhyne T., 1988, An Algorithm for Identifying and Selecting The Prime Implicants of a Multiple-Output Boolean Function. IEEE Transactions On Computer Aided Design, Vol. 7, No:11, November.

Pomper G. and Armstrong J.A., 1981, Representation of multivalued functions using the direct cover method. IEEE Trans. Comput, pp. 674-679, Sept.

Quine W.L., 1952, The problem of Simplifying Truth Functions. American Mathematics Monthly, Vol. 59, pp.521-531.

Quine, W.L., 1955, A Way of Simplifying Truth Functions. Am. Math. Montly, Vol. 62, No. 9, pp. 627-631.

Reddy S.M., 1972, Easily Testable Realizations For Logic Function. IEEE Transactions on Computers, 21, pp:1183-1188.

Reddy S.M., Kunz W., Pradhan D., 1995, Novel Verification Framework Combining Structural and OBDD Methods in a Synthesis Environment. The Proceedings of the Design Automation Conference, sayfalar 414-419, Haziran.

Rhyne, V.T., Noe, P.S., McKinny, M.N., and Pooch, U.W., 1977, A New Technique For The Fast Minimization of Switching Function. IEEE Trans. On Comp., Vol. C-26, No.8, pp. 757-764, August.

Roth J.P.,1956, Algebric Topological Methods for the Synthesis of Switching Systems in n-variables. The Instute for Advanced Study, Princeton, New Jersey, ECP56-02.

Roth, J.P., 1980, Computer Logic, Testing and Verification. Computer Science Press.

Rujipattanapong, S., 2001, Minimization of Boolean Functions and Transient Analysis of Exclusive-OR Switching Circuits. PhD Dissertation, University of Massachusetts Lowell.

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, UCB/ERL M89/49.

Rudell R.L., 1993, Dynamic Variable Ordwering for Binary Decision Diagrams. The Proccedings International Conference on Computer-Aiaded Design, sayfalar 42-47, Kasim.

Saad Y. and Schultz M.H., 1988, Topological Properties of Hypercubes. IEEE Trans. on Comput., C- 37(7), pp. 867-872.

Sanghavi J.V., 1996, High Performance Verification Algorithms. Phd Thesis, University of California, Berkeley.

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., Besslich PH.W., 1990, On The Complexity of Mod-2 Sum PLA's. IEEE Transactions on Computers, 39, pp:262-266.

Sasao T., Debnath D., 1994, An Exact Minimization Algorithm for Generalized Reed-Muller Expression. IEEE ASPDAC, pp:460-465.

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

Svoboda A., White D.E, 1979, Advanced Logical Circuit Design Techniques. New York: Garland Pres.

Saul J., 1991, An Algorithm for the Multi-Level Minimization of Reed-Muller Representations. IEEE. pages:634-637.

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

Sentovich E., et al, 1992, SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERI, M92/41, ERL, Dept. of EECS, Univ. of California, Berkeley.

Shmoys D.B., Tardos E., 1989, Computational complexity of combinatorial problems. in Lovász L., R.L. Graham, and M. Groetschel, eds., Handbook of Combinatorics. North-Holland, Amsterdam.

Sieling D., Wegener I., 1995, Graph-driven OBDDs – A New Data Structure for Boolean Functions, Theoretical Computer Science.

Stockmeyer L.J., 1990, Complexity theory. in Coffman, Jr., E. G., J. K. Lenstra, and A. H. G. Rinnooy Kan, eds., Handbooks in Operations Research and Management Science; Volume 3: Computation, Chapter 8. North Holland, Amsterdam.

Tran, A., 1987, Graphical Method For The Conversion of Minterms to Reed-Muller Coefficients And The Minimization of Exclusive-OR Switching Functions. IEE Proceedings, PT E, Computers and Digital Techniques, pp:93-99.

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

Uçar O., 1996, Lojik Devre Tasarimi Algoritmalari, Istanbul Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Yüsek Lisans Tezi.

Ugur A, 2002, http://bornova.ege.edu.tr/~ugur, Ege Üniversitesi, Bilgisayar Mühendisligi Bölümü.

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

Veitch E.W., 1952, A Chart Method for Simplifying Truth Functions. Proc. of the ACM, pp. 127-133.

Villa T., Saldanha A., Brayton R.K., Sangiovanni-Vincentelli A., 1997, Symbolic Two-Level Minimization. IEEE Trans. on Computer-Aided Design, CAD-16(7), pp. 692-708.

http://tr.wikipedia.org/wiki/Big-O

EK A

YAKIN-MINIMUM ÖRTME ALGORITMASI ILE YAPILMIS MINIMUMLASTIRMA ÖRNEKLERI

Örnek 1. $f(x_3,x_2,x_1,x_0)=\Sigma(0,1,2,3,5,7,10,11,13,14,15)$ Boole fonksiyonunu Yakin-Minimum Örtme Algoritmasini kullanarak minimumlastirin.

S_{ON1} = {0000,0001,0010,0011,0101,0111,1010,1011,1101,1110,1111},

S_{OFF} = {0100,0110,1000,1001,1100},

S_{ON} kümesinin ilk mintermi 0000, $\lambda 1$=0000,

Tablo 1.1. 0000 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
0100	x1xx	Asal	x1xx
0110	x11x	x1xx tarafindan kapsandi	
1000	1xxx	Asal	1xxx
1001	1xx1	1xxx tarafindan kapsandi	
1100	11xx	x1xx tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.1 = {x1xx,1xxx}.

1.2. 0000 minterminin örttügü küp kümesinin belirlenmesi.

S1 = xxxx # Q1 = {((xxxx # x1xx) # 1xxx)}

= {x0xx # 1xxx} = {00xx}

S1 = {S1.1} = {00xx}.

1.3. En büyük küpün belirlenmesi.

$P1.1 = S_{ON1}$ # $S1.1 = S_{ON1}$ # 00xx

= {0000,0001,0010,0011,0101,0111,1010,1011,1101,1110,1111} # 00xx,

= {0101,0111,1010,1011,1101,1110,1111}.

Yukaridaki islemlerde görüldügü gibi S1 kümesi tek elemanlidir. Bundan dolayi P1.1 kümesi (7 eleman) dir. Böylece, E1 = 00xx, SPI = {00xx}.

2.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

$S_{ON2} = P1.1$ = {0101,0111,1010,1011,1101,1110,1111},

Buradan, yeni S_{ON2} kümesinin ilk mintermi 0101, $\lambda 2 = 0101$,

Tablo 1.2. 0101 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
0100	xxx0	Asal	xxx0
0110	xx10	xxx0 tarafindan kapsandi	
1000	10x0	xxx0 tarafindan kapsandi	
1001	10xx	Asal	10xx
1100	1xx0	xxx0 tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.2 = {xxx0,10xx}

2.2. 0101 minterminin örttügü küp kümesinin belirlenmesi.

S2 = xxxx # Q1 = {(xxxx # xxx0) # 10xx}

= {xxx1 # 10xx} = {0xx1,x1x1},

S2 = {S2.1,S2.2} = {0xx1,x1x1}.

2.3. En büyük küpün belirlenmesi.

$P2.1 = S_{ON2}$ # $S2.1 = S_{ON2}$ # 0xx1 = {0101,0111,1010,1011,1101,1110,1111} # 0xx1

= {1010,1011,1101,1110,1111},

$P2.2 = S_{ON2} \# S2.2 = S_{ON2} \# x1x1 = \{0101,0111,1010,1011,1101,1110,1111\} \# x1x1$
$= \{1010,1011,1110\}$,

Yukaridaki islemlerde görüldügü gibi P2.2 kümesi (3 eleman) P2.1 kümesinde (5 eleman) daha güçlüdür. Sonuç olarak x1x1 küpü 0xx1 küpünden daha büyüktür. Böylece, E2 = x1x1, SPI = {00xx,x1x1}.

3.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

$S_{ON3} = P2.2 = \{1010,1011,1110\}$,

Buradan, yeni S_{ON3} kümesinin ilk mintermi 1010, $\lambda 3 = 1010$,

Tablo 1.3. 1010 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
0100	010x	01xx tarafindan kapsandi	
0110	01xx	Asal	01xx
1000	xx0x	Asal	xx0x
1001	xx01	xx0x tarafindan kapsandi	
1100	x10x	xx0x tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.3 = {xx0x,01xx}.

3.2. 1010 minterminin örttügü küp kümesinin belirlenmesi.

$S3 = xxxx \# Q1 = \{(xxxx \# xx0x) \# 01xx\}$
$= \{xx1x \# 01xx\} = \{1x1x,x01x\}$
$S3 = \{S3.1,S3.2\} = \{1x1x,x01x\}$.

3.3. En büyük küpün belirlenmesi.

$P3.1 = S_{ON3} \# 10x0 = \{1010,1011,1110\} \# 1x1x = \varnothing$

$P3.2 = S_{ON3} \# x000 = \{1010,1011,1110\} \# x01x = \{1110\}$

Yukaridaki islemlerde görüldügü gibi P3.1 kümesi (0 eleman) P3.2 kümesinde (1 eleman) daha güçlüdür. Sonuç olarak 1x1x küpü x01x küpünden daha büyüktür. Böylece, E3 = 1x1x, SPI = {00xx,x1x1,1x1x}.

4.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

S_{ON4} = P3.1 = ∅.

Sonuç olarak, sadelestirme islemi tamamlanmistir. Çünkü, S_{ON} kümesindeki bütün mintermler örtülmüstür. Fonksiyonun sadelestirilmis durumu;

SPI = {00xx,x1x1,1x1x}.

$$f = \bar{x}_3\,\bar{x}_2 + x_2\,x_0 + x_3\,x_1$$

Örnek 2.

$f(x_4,x_3,x_2,x_1,x_0)=\Sigma(1,3,5,7,8,9,10,11,13,15,16,17,18,19,20,$
$21,22,23,24,25,26,27,28,29,30,31)$

Boole fonksiyonunu Yakin-Minimum Örtme Algoritmasini kullanarak minimumlastirin.

S_{ON1} = {00001,00011,00101,00111,01000,01001,01010,01011,01101,01111,10000, 10001,10010,10011,10100,10101,10110,10111,11000,11001,11010,11011,11100, 11101,11110,11111},

S_{OFF} = {00000,00010,00100,00110,01100,01110},

S_{ON} kümesinin ilk mintermi 00001, λ1=00001,

Tablo 2.1. 00001 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
00000	xxxx0	Asal	xxxx0
00010	xxx10	xxxx0 tarafindan kapsandi	
00100	xx1x0	xxxx0 tarafindan kapsandi	
00110	xx110	xxxx0 tarafindan kapsandi	
01100	x11x0	xxxx0 tarafindan kapsandi	
01110	x1110	xxxx0 tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.1 = {xxxx0}.

1.2. 00001 minterminin örttügü küp kümesinin belirlenmesi.

S1 = xxxxx # Q1 = {xxxxx # xxxx0} = {xxxx1}

S1 = {S1.1} = {xxxx1}.

1.3. En büyük küpün belirlenmesi.

P1.1 = S_{ON1} # S1.1 = S_{ON1} # xxxx1

S_{ON1} = {00001,00011,00101,00111,01000,01001,01010,01011,01101,01111,10000,
10001,10010,10011,10100,10101,10110,10111,11000,11001,11010,11011,
11100,11101,11110,11111} # xxxx1,
= {01000,01010,10000,10010,10100,10110,11000,11010,11100,11110}.

Yukaridaki islemlerde görüldügü gibi S1 kümesi tek elemanlidir. Bundan dolayi P1.1 kümesi (10 eleman) dir. Böylece, E1 = xxxx1, SPI = {xxxx1}.

2.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

S_{ON2} = P1.1 = {01000,01010,10000,10010,10100,10110,11000,11010,11100,11110}

Buradan, yeni S_{ON2} kümesinin ilk mintermi 01000, $\lambda 2$ = 01000,

Tablo 2.2. 01000 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
00000	x0xxx	Asal	x0xxx
00010	x0x1x	x0xxx tarafindan kapsandi	
00100	x01xx	x0xxx tarafindan kapsandi	
00110	x011x	x0xxx tarafindan kapsandi	
01100	xx1xx	Asal	xx1xx
01110	xx11x	xx1xx tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.2 = {x0xxx,xx1xx}

2.2. 01000 minterminin örttügü küp kümesinin belirlenmesi.

S2 = xxxxx # Q1 = {(xxxxx # x0xxx) # xx1xx}

= {x1xxx # xx1xx} = {x10xx},

S2 = {S2.1} = {x10xx}.

2.3. En büyük küpün belirlenmesi.

P2.1 = S_{ON2} # S2.1 = S_{ON2} # x10xx

= {01000,01010,10000,10010,10100,10110,11000,11010,11100,11110}#x10xx

= {10000,10010,10100,10110,11100,11110},

Yukaridaki islemde görüldügü gibi P2.1 kümesi (6 eleman) dir. Böylece,

E2 = x10xx, SPI = {xxxx1,x10xx}.

3.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

S_{ON3}=P2.2={10000,10010,10100,10110,11100,11110},

Buradan, yeni S_{ON3} kümesinin ilk mintermi 10000, $\lambda 3$ = 10000,

Tablo 2.3. 10000 mintermi için Q0.1 ve Q1.1 kümelerinin belirlenmesi

S_{OFF}	Q0.1	Küp Durumu	Q1.1
00000	0xxxx	Asal	0xxxx
00010	0xx1x	0xxxx tarafindan kapsandi	
00100	0x1xx	0xxxx tarafindan kapsandi	
00110	0x11x	0xxxx tarafindan kapsandi	
01100	011xx	0xxxx tarafindan kapsandi	
01110	0111x	0xxxx tarafindan kapsandi	

Yukaridaki tablodan da görüldügü gibi, Q1.3 = {0xxxx}.

3.2. 10000 minterminin örttügü küp kümesinin belirlenmesi.

S3 = xxxxx # Q1 = {xxxxx # 0xxxx}

= {1xxxx}

S3 = {S3.1} = {1xxxx }.

3.3. En büyük küpün belirlenmesi.

P3.1 = S_{ON3} # 1xxxx = {10000,10010,10100,10110,11100,11110} # 1xxxx = ∅

Yukaridaki islemler görüldügü gibi P3.1 kümesi (0 eleman) dir. Böylece,

E3 = 1xxxx,

SPI = {xxxx1,x10xx,1xxxx}.

4.1. S_{ON} kümesinin örtülmeyen kisminin belirlenmesi.

S_{ON4} = P3.1 = ∅.

Sonuç olarak, sadelestirme islemi tamamlanmistir. Çünkü S_{ON} kümesindeki bütün mintermler örtülmüstür. Fonksiyonun sadelestirilmis durumu;

SPI = {xxxx1,x10xx,1xxxx}.

$$f = x_0 + x_3 \bar{x}_2 + x_4$$

EK B

YAKIN-MINIMUM ÖRTME ALGORITMASI PROGRAM KODLARI

```
#include <stdio.h>   #include <stdlib.h>   #include <conio.h>
#include <alloc.h>   #include <dos.h>              #include <time.h>
//*************************************************************************
//                          Global Degiskenler                             *
//*************************************************************************
int degisken_sayi,cikis_sayi,birim,birim2;
unsigned int son_sayi,soff_sayi,q_sayi,sp1_sayi,s1_sayi,q_max;
unsigned int far *my_son, far *my_soff, far *my_q, far *my_s1, far *y_s1, far *my_sp1, far *y_sp1;
unsigned long int ilk_bellek,son_bellek,ara_bellek;
clock_t  ilk_zaman, son_zaman;
//*************************************************************************
int my_atoi(char x)
{  int sayi=x;  return (sayi-48);  }
//*************************************************************************
void sonuc_yaz(FILE *y_d,int z)
{  unsigned int i,j,ii,son,iki=49152;
  unsigned int far *f;
  if (soff_sayi==0)  fprintf(y_d,"f%d = 1\n\n",z);
  else
  { f=my_sp1;
    fprintf(y_d,"p=%u \n",sp1_sayi);    fprintf(y_d,"\nf%d = \n",z);
    i=0;
    do
    { for (j=0;j<birim2;j++)
      { if (birim2-j==1)
          { son=degisken_sayi%8;
            if (son==0)  son=8;
          }
          else  son=8;
          for (ii=0;ii<son;ii++)
          { if ((*f&(iki>>(2*ii)))==(iki>>(2*ii)))  fprintf(y_d,"-");
            else if  ((*f&(iki>>(2*ii)))!=0)       fprintf(y_d,"1");
            else                                   fprintf(y_d,"0");
          }
          f++;
      }
      i++; fprintf(y_d,"\n");
    } while (i<sp1_sayi);
    fprintf(y_d,"\n");
  }
}
//*************************************************************************
void q0_yap()
{  unsigned int far *fon, far *foff, far *fq, far *y_q;
  unsigned int i,j,k,t,ds,sn,q,q_adet,a_b,b_a,ii,left,right, sayi[16],bir=32768;
  q_adet=0;    my_q=(unsigned int far *)farcalloc(birim2*q_max,sizeof(unsigned int));
  fon=my_son;  foff=my_soff;
  for (k=0;k<soff_sayi;k++)
  { for (j=0;j<(2*birim);j++)  sayi[j]=0;
    i=0;
    do
    { q=( ~(*fon ^ *foff));    left=q;   right=q | *foff;
```

```
      for (ii=8;ii<16;ii++)
      {  sayi[2*i+1]+=( left & (bir >>ii) )<<(16-ii); sayi[2*i+1]+=( right & (bir >>ii) ) <<  (15-ii);  }
      fon++;  foff++;
    } while (++i<birim);
    if (q_adet==0)
    {  fq=my_q;
       for (j=0;j<birim2;j++)
       { *fq=sayi[j]; fq++;  }
       q_adet++;
    }
    else
    {     for (i=0;i<q_adet;i++)
          { fq=my_q+(i*birim2);
            a_b=0;  b_a=0;
            for (j=0;j<birim2;j++)
            { if ( (*fq&sayi[j]) == sayi[j] )  a_b++;
              else if ( (*fq|sayi[j]) == sayi[j] ) b_a++;
              fq++;
            }
            if (a_b==birim2) { a_b=100; break;  }
            else if (b_a==birim2)
            { fq=my_q+(i*birim2);
              for (j=0;j<birim2;j++)
                 *(fq+j)=*(my_q+((q_adet-1)*birim2)+j);
              q_adet--; i--;
            }
          }
    }
    (t==1) ? (foff++):(foff=foff);
    fon=my_son;
  }
  q_sayi=q_adet;
}
//*****************************************************************************
void s1_yap()
{  unsigned int far *fon, far *foff, far *fs, iki=49152, sifirbir=16384, islenecek;
   unsigned int i,j,fi,abs,bs,y,z,ds,sn,bsno,birimno,a_b,b_a, sirano[64],deger[64];
   unsigned long int fj,yno,li,tbs,ty;
   s1_sayi=1;   my_s1=(unsigned int far *)farcalloc(birim2*s1_sayi,sizeof(unsigned int));
   for (i=0;i<birim2;i++)
   { if (i==(birim2-1))
     { *(my_s1+i)=0;
       for (j=0;j < (degisken_sayi-((birim2-1)*8));j++)   *(my_s1+i)+=(iki>>(2*j));
     }
     else  *(my_s1+i)=65535;
   }
   for (fi=0;fi<q_sayi;fi++)
   { tbs=0;  ty=0;
     for (fj=0;fj<s1_sayi;fj++)
     { fon=my_s1+(fj*birim2);   foff=my_q+(fi*birim2);   ds=degisken_sayi; i=0; y=0; abs=0;
       do
       { j=0;
           if (ds>8)  { sn=8; ds-=8; }
           else    sn=ds;
           do
           { if ((*fon&(iki>>(2*j)))==(*foff&(iki>>(2*j))))   {   }
             else if ((*foff&(iki>>(2*j)))==(iki>>(2*j)))     {   }
             else if (((*fon&(iki>>(2*j)))==(iki>>(2*j)))&&
```

```
                ((*foff&(iki>>(2*j)))!=(iki>>(2*j))))
              abs++;
        else  y++;
       } while (++j<sn);
       fon++; foff++;
  } while (++i<birim2);
  if (y>0)  { ty++; abs=0; }
  tbs+=abs;
}
tbs+=ty; yno=0;
for (fj=0;fj<s1_sayi;fj++)
{ fon=my_s1+(fj*birim2);  foff=my_q+(fi*birim2);  ds=degisken_sayi;  i=0; z=0; y=0; bs=0;
  do
  { j=0;
      if (ds>8)  { sn=8; ds-=8; }
      else sn=ds;
      fon++; foff++;
  } while (++i<birim2);
  if (yno==0)  y_s1=(unsigned int far *)farcalloc(birim2*tbs,sizeof(unsigned int));
  if (y>0)
  { for (i=0;i<birim2;i++) *(y_s1+(yno*birim2)+i)=*(my_s1+(fj*birim2)+i);
    yno++;
  }
  else if (z==degisken_sayi)  { /*bisey yok*/ }
  else
  { for (i=0;i<bs;i++)
      { for (j=0;j<birim2;j++) *(y_s1+(yno*birim2)+j)=*(my_s1+(fj*birim2)+j);
        birimno=(sirano[i]-1)/8;  bsno=sirano[i]%8;
        if (bsno==0) bsno=8;
        islenecek=*(y_s1+(yno*birim2)+(birimno));
        if ((deger[i])==1)  islenecek=islenecek&(~(iki>>(2*(bsno-1))))+(sifirbir>>(2*(bsno-1)));
        else  islenecek=islenecek&(~(iki>>(2*(bsno-1))));
        *(y_s1+(yno*birim2)+(birimno))=islenecek;
        yno++;
      }
  }
}
farfree(my_s1);   my_s1=y_s1;   s1_sayi=yno;   y_s1=NULL;
if (s1_sayi>1)
{  for (i=0;i<(s1_sayi-1);i++)
     { j=i+1;
       while (j<s1_sayi)
       { a_b=0; b_a=0;
         for (y=0;y<birim2;y++)
         { if ( ( *(my_s1+(i*birim2)+y) & *(my_s1+(j*birim2)+y)  ) == *(my_s1+(j*birim2)+y) )
           { a_b++;}
           else if ((*(my_s1+(i*birim2)+y) | *(my_s1+(j*birim2)+y) )== *(my_s1+(j*birim2)+y))
           { b_a++;}
         }
         if (a_b==birim2)
         {  for (y=0;y<birim2;y++)  *(my_s1+(j*birim2)+y)=*(my_s1+((s1_sayi-1)*birim2)+y);
           s1_sayi--;
         }
         else if (b_a==birim2)
         {  for (y=0;y<birim2;y++) *(my_s1+(i*birim2)+y)=*(my_s1+((s1_sayi-1)*birim2)+y);
            s1_sayi--; i--; break;
         }
         else j++;
```

```
          }
         }
         y_s1=(unsigned int far *)farcalloc(birim2*s1_sayi,sizeof(unsigned int));
         for (li=0;li<s1_sayi;li++)
           for (i=0;i<birim2;i++)   *(y_s1+(li*birim2)+i)=*(my_s1+(li*birim2)+i);
         farfree(my_s1);         my_s1=y_s1; y_s1=NULL;
    }
  }
}
//*****************************************************************************
unsigned long int e1_bul()
{  unsigned int far *fon, far *fs1;
  unsigned int i,j,k,t,left[8],right[8],left1,right1,sayi[16], bir=32768,encok=0,onsay,say,sn;
  unsigned long int enno=0,l;
  for (l=0;l<s1_sayi;l++)
  { onsay=0;  fs1=my_s1+(l*birim2);
    for (t=0;t<(2*birim);t++)  sayi[t]=0;
    for (t=0;t<birim2;t++)     sayi[t]=*(fs1+t);
    for (i=0;i<8;i++)
    { left[i]=0;  right[i]=0;
      for (t=0;t<8;t++)
      { left[i] += (sayi[2*i]&(bir>>(2*t)))<< t ; right[i]+=(sayi[2*i]&(bir>> (2*t+1) ) ) << (t+1) ; }
      for (t=0;t<8;t++)
      { left[i]+=(sayi[2*i+1]&(bir>>(2*t)))>>(8-t); right[i]+=(sayi[2*i+1]&(bir>>(2*t+1)))>>(7-t) ; }
    }
    for (k=0;k<son_sayi;k++)
    { fon=my_son+(k*birim);  i=0; t=0; say=0;
     do
      { if (++t==2)  { t=0; fon++; }
        if (((left[i]&right[i])|*fon)==right[i]) say++;
      } while (++i<(birim));
      if (say==birim) onsay++;
    }
    if (onsay>=encok) { encok=onsay; enno=l; }
  }
  return(enno);
}
//*****************************************************************************
void sp1_ekle(unsigned long int z)
{  unsigned int i,j;
  if (sp1_sayi==0)
  { my_sp1=(unsigned int far *)farcalloc(birim2*++sp1_sayi,sizeof(unsigned int));
    for (i=0;i<birim2;i++)  *(my_sp1+i)=*(my_s1+(z*birim2)+i);
  }
  else
  { y_sp1=(unsigned int far *)farcalloc(birim2*++sp1_sayi,sizeof(unsigned int));
    for (j=0;j<sp1_sayi-1;j++)
    { for (i=0;i<birim2;i++)  *(y_sp1+(j*birim2)+i)=*(my_sp1+(j*birim2)+i); }
    j=sp1_sayi-1;
    for (i=0;i<birim2;i++)  *(y_sp1+(j*birim2)+i)=*(my_s1+(z*birim2)+i);
    farfree(my_sp1); my_sp1=y_sp1; y_sp1=NULL;
  }
}
//*****************************************************************************
void son_cikar(unsigned long int z)
{  unsigned int far *fon, far *fs1;
  unsigned int i,j,k,t,say,ds,sn,left[8],right[8],left1,right1,sayi[16], bir=32768
  fs1=my_s1+(z*birim2);
```

```
  for (t=0;t<(2*birim);t++)  sayi[t]=0;
  for (t=0;t<birim2;t++)     sayi[t]=*(fs1+t);
  for (i=0;i<8;i++)
  { left[i]=0; right[i]=0;
    for (t=0;t<8;t++)
    { left[i]+=(sayi[2*i]&(bir>>(2*t  ) ) ) << t;  right[i]+= ( sayi[2*i] & ( bir>> (2*t+1) ) ) << (t+1);  }
    for (t=0;t<8;t++)
    { left[i]+=(sayi[2*i+1]&(bir>>(2*t)))>>(8-t); right[i]+=(sayi[2*i+1]&(bir>>(2*t+1)))>>(7-t); }
  }
  for (k=0;k<son_sayi;k++)
  { fon=my_son+(k*birim);  i=0; t=0; say=0;
    do
    { if (++t==2)  { t=0; fon++; }
      if (((left[i]&right[i])|*fon)==right[i])   say++;
    } while (++i<birim);
    if (say==birim)
    { for (i=k;i<son_sayi-1;i++)
      { for (j=0;j<birim;j++)  *(my_son+(i*birim)+j)=*(my_son+((i+1)*birim)+j); }
     *(my_son+((--son_sayi)*birim))=0;   k--;
    }
  }
}
//*****************************************************************************
void dosya_okuma(FILE *o_d,int y)
{  unsigned int i,j,son_no,soff_no;
   unsigned int sayi[8]={0},bir;
   char kr;
   unsigned int far *eleman;
   my_son=(unsigned int far *)farcalloc(birim*son_sayi,sizeof(unsigned int));
   my_soff=(unsigned int far *)farcalloc(birim*soff_sayi,sizeof(unsigned int));  y--;
   while(!feof(o_d))
   { kr=getc(o_d);
     if (kr=='.')
     { do { kr=getc(o_d); } while(kr!='\n');  }
     else
     { son_no=0; soff_no=0;
       do
       { for (j=0;j<birim;j++)  sayi[j]=0;
          i=0; bir=32768; j=0;
          do
          { if (kr=='1')  { sayi[j]+=(bir>>i); }
            i++;
            if (i==16)  { i=0; j++; }
            kr=getc(o_d);
          } while (kr!=' ');
          kr=getc(o_d); i=0;
          do
          { if (i==y)
            { switch (kr)
              { case '1':   eleman=my_son; eleman+=birim*son_no;
                            for (j=0;j<birim;j++)
                            { *eleman=sayi[j];  eleman++; }  son_no++; break;
                case '0':   eleman=my_soff;  eleman+=birim*soff_no;
                            for (j=0;j<birim;j++)
                            { *eleman=sayi[j];  eleman++; }  soff_no++; break;
                default :   break;
              }
            }
```

```
            i++; kr=getc(o_d);
           } while (kr!='\n');
           kr=getc(o_d);
      } while (kr!='.');
      break;
    }
  }
}
//*****************************************************************************************
int dosya_islemi(int s,char *a, char *b)
{  FILE *okunan_dosya, *yazilan_dosya;
  char kr,ykr;
  unsigned int i,sayi,y,j;
  unsigned long int e1no;
  if (s<3)                                              return(1);
  if ((okunan_dosya=fopen(a,"r"))==NULL)   return(2);
  if ((yazilan_dosya=fopen(b,"w"))==NULL)  return(3);
  ilk_zaman=clock();
  for (i=0;i<2;i++)
  { kr=getc(okunan_dosya);
    if (kr=='.')
    { kr=getc(okunan_dosya);    ykr=kr;  kr=getc(okunan_dosya);  kr=getc(okunan_dosya);   sayi=0;
      while(kr!='\n')
      { sayi=sayi*10+my_atoi(kr);  kr=getc(okunan_dosya);  }
      switch (ykr)
      { case 'i' : degisken_sayi=sayi; break;
        case 'o' : cikis_sayi=sayi;    break;
      }
    }
  }
  birim=(degisken_sayi+15)/16;   birim2=(2*degisken_sayi+15)/16;
  for(i=1;i<=cikis_sayi;i++)
  { son_sayi=0; soff_sayi=0;  y=i;   y--;  fclose(okunan_dosya);  okunan_dosya=fopen(a,"r");
    while(!feof(okunan_dosya))
    { kr=getc(okunan_dosya);
      if (kr=='.')
      { do { kr=getc(okunan_dosya); } while(kr!='\n'); }
      else
      { do
          { do { kr=getc(okunan_dosya); } while (kr!=' ');
            kr=getc(okunan_dosya);  j=0;
            do
            { if (j==y)
              { switch (kr)
                { case '1': son_sayi++;  break;   case '0': soff_sayi++; break;   default : break;  }
              }
              j++; kr=getc(okunan_dosya);
            } while (kr!='\n');
            kr=getc(okunan_dosya);
          } while (kr!='.');
          break;
      }
    }
    fclose(okunan_dosya);  okunan_dosya=fopen(a,"r");  dosya_okuma(okunan_dosya,i);  sp1_sayi=0;
    if (son_sayi==0)
    { fprintf(yazilan_dosya,"f%d = 0\n\n",i);
      farfree(my_q); my_q=NULL; q_sayi=0; farfree(my_s1);  my_s1=NULL; farfree(y_s1);
      y_s1=NULL ; s1_sayi=0;  farfree(my_son);  my_son=NULL;   farfree(my_soff);
```

```
      my_soff=NULL;  soff_sayi=0;
    }
    else
    { if (soff_sayi!=0)
      { while (son_sayi>0)
          { if (soff_sayi<10) q_max=soff_sayi;
            else              q_max=10;
            q0_yap();    s1_yap();  farfree(my_q);  q_sayi=0;  e1no=e1_bul(); son_cikar(e1no);
            sp1_ekle(e1no);  farfree(my_s1);  my_s1=NULL;  s1_sayi=0;
          }
      }
     sonuc_yaz(yazilan_dosya,i);
    }
    farfree(my_son);  son_sayi=0; farfree(my_soff); soff_sayi=0; farfree(my_sp1); sp1_sayi=0;
  }
  son_zaman=clock();
  fprintf(yazilan_dosya,"Cozum  Zamani= %-6.3f  saniyedir.",((son_zaman-ilk_zaman)/CLK_TCK));
  fclose(okunan_dosya);  fclose(yazilan_dosya);  return(0);
}
//*************************************************************************************
//                                   A N A   P R O G R A M                             *
//*************************************************************************************
void main (int argc,char *argv[])
{  clrscr();
  switch(dosya_islemi(argc,argv[1],argv[2]))
  {  case 0 : printf(" !!! Program Sorunsuz Calisti. !!!\n");   break;
     case 1 : printf(" !!! Dosya Adlari Unutuldu !!!\n");       break;
     case 2 : printf(" !!! Dosya Okunmak Icin Acilamadi !!!\n");break;
     case 3 : printf(" !!! Dosya Yazmak Icin Acilamadi !!!\n"); break;
  }
}
```

EK C

KESIN-MINIMUM ÖRTME ALGORITMASI PROGRAM KODLARI

```
#include <stdio.h>   #include <conio.h>   #include <alloc.h>   #include <string.h>
//*****************************************************************************
//                    Global Degiskenler                                       *
//*****************************************************************************
int degisken_sayi,cikis_sayi,birim,birim2,tt=0,isaret_max,sp2_max,sp1_max;
unsigned int son_sayi,soff_sayi,q_sayi,sp1_sayi,sp2_sayi,s1_sayi,isaret_sayi;
unsigned int far *my_son, far *my_soff, far *my_q, far *my_s1,  far *y_s1;
unsigned int far *my_sp1, far *y_sp1, far *my_sp2, far *y_sp2;
unsigned int far *my_imp, far *my_isaret, far *r_sayi,far **r_bas;
unsigned int imp_max, imp_sayi,s1_max, r_adet, r_max,h_minterm,q_max;
//*****************************************************************************
int my_atoi(char x) { int sayi=x; return (sayi-48); }
//*****************************************************************************
void sonuc_yaz(FILE *y_d,int z)
{  unsigned int i,j,ii,son,iki=49152, far *f;
   if (soff_sayi==0) fprintf(y_d,"f%d = 1\n\n",z);
  else
  { if (sp2_sayi>0) fprintf(y_d,".p %u   ***\n",(sp1_sayi+sp2_sayi));
    else  fprintf(y_d,".p %u\n",sp1_sayi);
    if (sp1_sayi>0)
    { f=my_sp1;  i=0;
      do
      { for (j=0;j<birim2;j++)
          { if (birim2-j==1)
            { son=degisken_sayi%8;     if (son==0)  son=8; }
            else  son=8;
            for (ii=0;ii<son;ii++)
            { if ((*f&(iki>>(2*ii)))==(iki>>(2*ii)))  fprintf(y_d,"-");
              else if  ((* f&(iki>>(2*ii)))!=0)        fprintf(y_d,"1");
              else                                     fprintf(y_d,"0");
            } f++;
          }  i++;  fprintf(y_d,"\n");
      } while (i<sp1_sayi);   }
    f=NULL;
    if (sp2_sayi>0)
    {  f=my_sp2;  i=0;  fprintf(y_d,"++++++++++++++\n");
         do
         { for (j=0;j<birim2;j++)
           { if (birim2-j==1)
             { son=degisken_sayi%8;  if (son==0)  son=8; }
             else son=8;
             for (ii=0;ii<son;ii++)
             { if ((*f&(iki>>(2*ii)))==(iki>>(2*ii)))  fprintf(y_d,"-");
               else if  ((*f&(iki>>(2*ii)))!=0)        fprintf(y_d,"1");
               else                                    fprintf(y_d,"0");
             }  f++;
           } i++; fprintf(y_d,"\n");
         } while (i<sp2_sayi);
    }    }   }
//*****************************************************************************
void q0_yap(int gelen)
{  unsigned int far *fon, far *foff, far *fq, far *y_q, i,j,k,t,ds,sn,q,q_adet,a_b,b_a,ii,left,right;
```

```
  unsigned int sayi[16],bir=32768;
  q_adet=0;
  if (gelen)
  {  h_minterm=0;
     do
     { i=0;
          for (j=0;j<isaret_sayi;j++)
          { ii=0;
            for (k=0;k<birim;k++)
              if (*((my_son+h_minterm)+k)==*((my_isaret+j)+k))  ii++;
            if (ii==birim) i=1;
          }
     } while ((i==1)&(++h_minterm<son_sayi));
     h_minterm--;    }
  else  h_minterm=0;
  fon=(my_son+h_minterm);  foff=my_soff;
  for (k=0;k<soff_sayi;k++)
  { for (j=0;j<(2*birim);j++)  sayi[j]=0;
    i=0;
    if (q_adet==0)
    {  fq=my_q;
          for (j=0;j<birim2;j++) { *fq=sayi[j]; fq++; }
          q_adet++;      }
    else
    {     for (i=0;i<q_adet;i++)
          { fq=my_q+(i*birim2);  a_b=0;  b_a=0;
            for (j=0;j<birim2;j++)
            { if ( *fq == sayi[j] ) { a_b++; b_a++; }
              else if ( (*fq&sayi[j]) == sayi[j] )   { a_b++; }
              else if ( (*fq|sayi[j]) == sayi[j] )   { b_a++; }
              fq++;
            }
            else if (b_a==birim2)
            { fq=my_q+(i*birim2);
              for (j=0;j<birim2;j++)
                *(fq+j)=*(my_q+((q_adet-1)*birim2)+j);    } }
            fq=my_q+(birim2*q_adet);
            for (j=0;j<birim2;j++)  *(fq+j)=sayi[j];
            q_adet++;  }
    }  fon=(my_son+h_minterm);
  } q_sayi=q_adet;
}
//*****************************************************************************
void s1_yap()
{  unsigned int far *fon, far *foff, far *fs, i,j,fi,abs,bs,y,z,ds,sn,bsno,birimno,a_b,b_a;
   unsigned long int fj,yno,li,tbs,ty;
   unsigned int iki=49152, sifirbir=16384, islenecek;
   int sirano[64],deger[64];
   s1_sayi=1;
   for (i=0;i<birim2;i++)
   { if (i==(birim2-1))
     { *(my_s1+i)=0;
       for (j=0;j < (degisken_sayi-((birim2-1)*8));j++)
           *(my_s1+i)+=(iki>>(2*j));    }
     else  *(my_s1+i)=65535;    }
   for (fi=0;fi<q_sayi;fi++)
   { tbs=0;  ty=0;
     for (fj=0;fj<s1_sayi;fj++)
```

```
{ fon=my_s1+(fj*birim2);   foff=my_q+(fi*birim2);
 ds=degisken_sayi;  i=0;  y=0;  abs=0;
 do
  { j=0;
      if (ds>8) { sn=8; ds-=8; }
      else       sn=ds;
      fon++;  foff++;
  } while (++i<birim2);
 if (y>0) { ty++;   abs=0; }
 tbs+=abs;     }
tbs+=ty;
if (tbs>(s1_max-1))
{  s1_max=tbs;  farfree(y_s1);  y_s1=NULL;
     y_s1=(unsigned int far *)farcalloc(birim2*s1_max,sizeof(unsigned int));
     for (j=0;j<s1_sayi;j++)
       for (i=0;i<birim2;i++)
         *(y_s1+(j*birim2)+i)=*(my_s1+(j*birim2)+i);
     farfree(my_s1);  my_s1=NULL;
     my_s1=(unsigned int far *)farcalloc(birim2*s1_max,sizeof(unsigned int));
     for (j=0;j<s1_sayi;j++)
       for (i=0;i<birim2;i++)
         *(my_s1+(j*birim2)+i)=*(y_s1+(j*birim2)+i);      }
 do
  { j=0;
      if (ds>8) { sn=8;  ds-=8; }
      else       sn=ds;
      do
      { if ((*fon&(iki>>(2*j)))==(*foff&(iki>>(2*j))))   { z++; }
        else if ((*foff&(iki>>(2*j)))==(iki>>(2*j)))     { z++; }
        else if (((*fon&(iki>>(2*j)))==(iki>>(2*j)))&&
                 ((*foff&(iki>>(2*j)))!=(iki>>(2*j))))
        { sirano[bs]=i*8+j+1;
          if ((*foff&(iki>>(2*j)))!=0)  deger[bs]=0;
          else                          deger[bs]=1;
          bs++;
        }
        else  y++;
      } while (++j<sn);
      fon++;  foff++;
  } while (++i<birim2);
   for (i=0;i<bs;i++)
      { for (j=0;j<birim2;j++) *(y_s1+(yno*birim2)+j)=*(my_s1+(fj*birim2)+j);
        birimno=(sirano[i]-1)/8;  bsno=sirano[i]%8;
        if (bsno==0) bsno=8;
        islenecek=*(y_s1+(yno*birim2)+(birimno));
        if ((deger[i])==1)
          islenecek=islenecek&(~(iki>>(2*(bsno-1))))+(sifirbir>>(2*(bsno-1)));
        else
          islenecek=islenecek&(~(iki>>(2*(bsno-1))));
        *(y_s1+(yno*birim2)+(birimno))=islenecek;
        yno++;  }      }     }
for (j=0;j<yno;j++)
  for (i=0;i<birim2;i++)
      *(my_s1+(j*birim2)+i)=*(y_s1+(j*birim2)+i);
if (s1_sayi>1)
{  for (i=0;i<(s1_sayi-1);i++)
      { j=i+1;
        while (j<s1_sayi)
```

```
          { a_b=0;  b_a=0;
            for (y=0;y<birim2;y++)
            { if ( *(my_s1+(i*birim2)+y) == *(my_s1+(j*birim2)+y)  ) { a_b++;  b_a++;    }
           else if ( ( *(my_s1+(i*birim2)+y) & *(my_s1+(j*birim2)+y) ) == *(my_s1+(j*birim2)+y) )
              { a_b++;}
              else if ((*(my_s1+(i*birim2)+y) | *(my_s1+(j*birim2)+y) ) == *(my_s1+(j*birim2)+y) )
              { b_a++;}    }
            if (a_b==birim2)
            {  for (y=0;y<birim2;y++)
                  *(my_s1+(j*birim2)+y)=*(my_s1+((s1_sayi-1)*birim2)+y);
              s1_sayi--;   }
            else if (b_a==birim2)
            {  for (y=0;y<birim2;y++)
                  *(my_s1+(i*birim2)+y)=*(my_s1+((s1_sayi-1)*birim2)+y);
              s1_sayi--;  i--;  break;   }
            else j++;   } }
    }   }   }
//*****************************************************************************
unsigned long int e1_bul()
{  unsigned int far *fon, far *fs1, far *y_imp, far *y_isaret, i,j,k,l,t,left[8],right[8],left1,right1,sayi[16];
   unsigned int onsay,say,sn,kesis_sayi,a_b,b_a;
   unsigned int sifirbir=16384, bir=32768, iki=49152 ;
   if (s1_sayi>r_adet)
   {  r_adet=s1_sayi;  farfree(r_sayi);   r_sayi=NULL;
      r_sayi=(unsigned int far *)farcalloc(r_adet,sizeof(unsigned int));
      for(j=0;j<r_adet;j++) {  farfree(*(r_bas+j));  *(r_bas+j)=NULL; }
      farfree(r_bas);  r_bas=NULL;
      r_bas=(unsigned int far **)farcalloc(r_adet,sizeof(unsigned int));
      for(j=0;j<r_adet;j++)
           *(r_bas+j)=(unsigned int far *)farcalloc(birim*r_max,sizeof(unsigned int));    }
   for (l=0;l<s1_sayi;l++)
   { onsay=0;  fs1=my_s1+(l*birim2);
     for (t=0;t<(2*birim);t++)  sayi[t]=0;
     for (t=0;t<birim2;t++)     sayi[t]=*(fs1+t);
     for (i=0;i<8;i++)
     { left[i]=0;   right[i]=0;
       for (t=0;t<8;t++)
       {  left[i] += ( sayi[2*i] & ( bir>> (2*t  ) ) ) << t ;
            right[i]+= ( sayi[2*i] & ( bir>> (2*t+1) ) ) << (t+1) ;     }
       for (t=0;t<8;t++)
       {  left[i] += ( sayi[2*i+1] & ( bir>> (2*t  ) ) ) >> (8-t) ;
            right[i]+= ( sayi[2*i+1] & ( bir>> (2*t+1) ) ) >> (7-t) ;    }     }
   for (i=0;i<s1_sayi-1;i++)
   { for (j=i+1;j<s1_sayi;j++)
     { kesis_sayi=0;
       for (k=0;k<*(r_sayi+i);k++)
           for (t=0;t<*(r_sayi+j);t++)
           { say=0;
             for (sn=0;sn<birim;sn++)
               if ( *( *(r_bas+i)+(birim*k)+sn) == *( *(r_bas+j)+(birim*t)+sn) )
                 say++;
               if (say==birim) kesis_sayi++;   }
      if ( ( *(r_sayi+i)==kesis_sayi) && (kesis_sayi< *(r_sayi+j)) )
      {  for (k=0;k< (*(r_sayi+(s1_sayi-1))*birim);k++)
            *( *(r_bas+i)+k)= *( *(r_bas+(s1_sayi-1))+k);
          *(r_sayi+i)=*(r_sayi+(s1_sayi-1));
          for (sn=0;sn<birim2;sn++)
            *(my_s1+(birim2*i)+sn)=*(my_s1+(birim2*(s1_sayi-1))+sn);
```

```
          s1_sayi--; i--;      }
    if ( ( *(r_sayi+j)==kesis_sayi) && (kesis_sayi< *(r_sayi+i)) )
    {  for (k=0;k< (*(r_sayi+(s1_sayi-1))*birim);k++)
          *( *(r_bas+j)+k) = *( *(r_bas+(s1_sayi-1))+k);
         *(r_sayi+j)=*(r_sayi+(s1_sayi-1));
         for (sn=0;sn<birim2;sn++)
           *(my_s1+(birim2*j)+sn)=*(my_s1+(birim2*(s1_sayi-1))+sn);
         s1_sayi--; j--;
    }   }   }
 if (s1_sayi==1)
 {  for (i=0;i< *(r_sayi+0);i++)
       for (j=0;j<isaret_sayi;j++)
       { say=0;
         for (k=0;k<birim;k++)
           if (*( *(r_bas+0)+(birim*i)+k) == *(my_isaret+(birim*j)+k)) say++;
         if (say==birim)
         { for (k=0;k<birim;k++)
             *(my_isaret+(birim*j)+k)=*(my_isaret+(birim*(isaret_sayi-1))+k);
           isaret_sayi--;   }     }
   for (j=0;j<imp_sayi;j++)
   { t=0;
       for (k=0;k<birim2;k++)
       { for (l=0;l<8;l++)
         { if ( ((*(my_s1+k) & (iki>>(2*l))) +
                   ( *(my_imp+(j*birim2)+k) & (iki>>(2*l)))
                  )
                  ==
                  ( sifirbir>>(2*l) )
             )
           { t=1; break; }
         }
         if (t) break;
       }
       if (t) continue;
       a_b=0;
       for (k=0;k<birim2;k++)
        if ( ( *(my_s1+k) & *(my_imp+(j*birim2)+k)  ) == *(my_imp+(j*birim2)+k) )
        {  a_b++; }
        if (a_b==birim2)
        {  for (k=0;k<birim2;k++)
           *(my_imp+(j*birim2)+k)=*(my_imp+((imp_sayi-1)*birim2)+k);
          imp_sayi--;
        };   }
   return(1);
 }
 else
 { if (isaret_sayi>(isaret_max-1))
   { isaret_max+=10;
     y_isaret=(unsigned int far *)farcalloc(birim*isaret_max,sizeof(unsigned int));
     for (j=0;j<isaret_sayi;j++)
     { for (i=0;i<birim;i++)
          *(y_isaret+(j*birim)+i)=*(my_isaret+(j*birim)+i);       }
     farfree(my_isaret); my_isaret=NULL; my_isaret=y_isaret; y_isaret=NULL;     }
   for (j=0;j<birim;j++)
     *(my_isaret+(birim*isaret_sayi)+j)=*(my_son+h_minterm+j);
   isaret_sayi++;
   for (i=0;i<s1_sayi;i++)
   {  for (j=0;j<imp_sayi;j++)
```

```
          { t=0;
            for (k=0;k<birim2;k++)
            { for (l=0;l<8;l++)
              { if ( ( ( *(my_s1+(i*birim2)+k) & (iki>>(2*l)) ) +
                              ( *(my_imp+(j*birim2)+k)  & (iki>>(2*l)) )
                            )
                            == ( sifirbir>>(2*l)
                          )
                        )
                 { t=1; break; }    }  }
            a_b=0;  b_a=0;
            for (k=0;k<birim2;k++)
            { if ( *(my_s1+(i*birim2)+k) == *(my_imp+(j*birim2)+k)  )
              { a_b++; b_a++;    }
            else if ( ( *(my_s1+(i*birim2)+k) & *(my_imp+(j*birim2)+k) )==*(my_imp+(j*birim2)+k) )
              { a_b++; }
            else if ( ( *(my_s1+(i*birim2)+k) | *(my_imp+(j*birim2)+k) ) == *(my_imp+(j*birim2)+k) )
              { b_a++; }  }
            if (a_b==birim2)
            { for (k=0;k<birim2;k++)
               *(my_imp+(j*birim2)+k)=*(my_imp+((imp_sayi-1)*birim2)+k);
              imp_sayi--;  }
               for (j=0;j<imp_sayi;j++)
               {  for (k=0;k<birim2;k++) *(y_imp+(j*birim2)+k)=*(my_imp+(j*birim2)+k);    }
               farfree(my_imp); my_imp=NULL; my_imp=y_imp; y_imp=NULL;  }
            for (sn=0;sn<birim2;sn++)
              *(my_imp+(imp_sayi*birim2)+sn)=*(my_s1+(birim2*i)+sn);
            imp_sayi++;  }   }
    return(0);
  }  }
//*****************************************************************************
unsigned long int e2_bul()
{  unsigned int far *fon, far *fs1,encok=0,onsay,say,sn,bir=32768;
   unsigned int i,j,k,t,left[8],right[8],left1,right1,sayi[16];
   unsigned long int enno=0,l;
   for (l=0;l<imp_sayi;l++)
   { onsay=0;  fs1=(my_imp+(l*birim2));
     for (t=0;t<(2*birim);t++)  sayi[t]=0;
     for (t=0;t<birim2;t++)     sayi[t]=*(fs1+t);
     for (i=0;i<8;i++)
     { left[i]=0;  right[i]=0;
       for (t=0;t<8;t++)
       { left[i] += ( sayi[2*i] & ( bir>> (2*t  ) ) ) << t ;
           right[i]+= ( sayi[2*i] & ( bir>> (2*t+1) ) ) << (t+1) ;      }
       for (t=0;t<8;t++)
       { left[i] += ( sayi[2*i+1] & ( bir>> (2*t  ) ) ) >> (8-t) ;
           right[i]+= ( sayi[2*i+1] & ( bir>> (2*t+1) ) ) >> (7-t) ;     }     }
     for (k=0;k<son_sayi;k++)
     { fon=my_son+(k*birim);  i=0;  t=0;  say=0;
       do
       { if (++t==2) { t=0; fon++; }
           if (((left[i]&right[i])|*fon)==right[i])  say++;
       } while (++i<(birim));
       if (say==birim) onsay++;    }
     if (onsay>encok) { encok=onsay;  enno=l; }   }
   return(enno);
}
//*****************************************************************************
```

```
void sp1_ekle(unsigned long int z)
{  unsigned int i,j;
   if (sp1_sayi==0)
   { for (i=0;i<birim2;i++)
      *(my_sp1+i)=*(my_s1+(z*birim2)+i);
    sp1_sayi++;   }
   else
   { if (sp1_sayi>(sp1_max-1))
    { sp1_max+=10;
     y_sp1=(unsigned int far *)farcalloc(birim2*sp1_max,sizeof(unsigned int));
      for (j=0;j<sp1_sayi;j++)
      { for (i=0;i<birim2;i++)
            *(y_sp1+(j*birim2)+i)=*(my_sp1+(j*birim2)+i);       }
      farfree(my_sp1);  my_sp1=y_sp1;  y_sp1=NULL;     }
    for (i=0;i<birim2;i++)
     *(my_sp1+(sp1_sayi*birim2)+i)=*(my_s1+(z*birim2)+i);
    sp1_sayi++;
   }   }
//*****************************************************************************
void sp2_ekle(unsigned long int z)
{  unsigned int i,j,t,k,l,a_b,b_a,sifirbir=16384, iki=49152;
   if (sp2_sayi==0)
   { for (i=0;i<birim2;i++)
      *(my_sp2+i)=*(my_imp+(z*birim2)+i);
    sp2_sayi++;   }
   else
   { if (sp2_sayi>(sp2_max-1))
    { sp2_max+=10;
     y_sp2=(unsigned int far *)farcalloc(birim2*sp2_max,sizeof(unsigned int));
      for (j=0;j<sp2_sayi;j++)
      { for (i=0;i<birim2;i++)
            *(y_sp2+(j*birim2)+i)=*(my_sp2+(j*birim2)+i);      }
      farfree(my_sp2);  my_sp2=y_sp2;   y_sp2=NULL;     }
    for (i=0;i<sp2_sayi;i++)
    {     t=0;
          for (k=0;k<birim2;k++)
          {  for (l=0;l<8;l++)
            { if ( ( ( *(my_sp2+(i*birim2)+k) & (iki>>(2*l)) ) +
                              ( *(my_imp+(z*birim2)+k)  & (iki>>(2*l)) )
                        )
                        == ( sifirbir>>(2*l)
                      )
                    )
              { t=1; break; }   }
            if (t) break;   }
          if (t&&tt) continue;
          a_b=0;           b_a=0;
          for (k=0;k<birim2;k++)
          { if ( *(my_sp2+(i*birim2)+k) == *(my_imp+(z*birim2)+k)  )
            { a_b++; b_a++; }
          else if ((*(my_sp2+(i*birim2)+k)&*(my_imp+(z*birim2)+k))==*(my_imp+(z*birim2)+k) )
            { a_b++; }
          else if ((*(my_sp2+(i*birim2)+k) | *(my_imp+(z*birim2)+k))==*(my_imp+(z*birim2)+k) )
            { b_a++; }}
          if (a_b==birim2)
          {   for (k=0;k<birim2;k++)
              *(my_sp2+(i*birim2)+k)=*(my_sp2+((sp2_sayi-1)*birim2)+k);
            sp2_sayi--; i--;        }
```

```
          else if (b_a==birim2)
          {  b_a=1000;  break;  };     }
    if (b_a!=1000)
    {  for (i=0;i<birim2;i++)
          *(my_sp2+(sp2_sayi*birim2)+i)=*(my_imp +(z*birim2)+i);
         sp2_sayi++;     }   }
  for (i=0;i<birim2;i++)
   *(my_imp+(z*birim2)+i)=*(my_imp+((imp_sayi-1)*birim2)+i);
  imp_sayi--;}

//*****************************************************************************
void son_cikar(unsigned long int z)
{  unsigned int far *fon, far *fs1,bir=32768;
  unsigned int i,j,k,t,say,ds,sn,left[8],right[8],left1,right1,sayi[16];
  fs1=my_s1+(z*birim2);
  for (t=0;t<(2*birim);t++)  sayi[t]=0;
  for (t=0;t<birim2;t++)     sayi[t]=*(fs1+t);
  for (i=0;i<8;i++)
  { left[i]=0;  right[i]=0;
    for (t=0;t<8;t++)
    { left[i] += ( sayi[2*i] & ( bir>> (2*t  ) ) ) << t ;
       right[i]+= ( sayi[2*i] & ( bir>> (2*t+1) ) ) << (t+1) ;     }
    for (t=0;t<8;t++)
    { left[i] += ( sayi[2*i+1] & ( bir>> (2*t  ) ) ) >> (8-t) ;
      right[i]+= ( sayi[2*i+1] & ( bir>> (2*t+1) ) ) >> (7-t) ;     }   }
  for (k=0;k<son_sayi;k++)
  { fon=my_son+(k*birim); i=0; t=0; say=0;
    do
    { if (++t==2) { t=0; fon++; }
     if (((left[i]&right[i])|*fon)==right[i]) say++;
    } while (++i<birim);
    if (say==birim)
    { for (i=k;i<son_sayi-1;i++)
      { for (j=0;j<birim;j++)  *(my_son+(i*birim)+j)=*(my_son+((i+1)*birim)+j);       }
     *(my_son+((--son_sayi)*birim))=0; k--;
    }   }  }
//*****************************************************************************
void son2_cikar(unsigned long int z)
{  unsigned int far *fon, far *fs1,bir=32768;
  unsigned int i,j,k,t,say,ds,sn,left[8],right[8],left1,right1,sayi[16];
  fs1=(my_imp+(z*birim2));
  for (t=0;t<(2*birim);t++)  sayi[t]=0;
  for (t=0;t<birim2;t++)     sayi[t]=*(fs1+t);
  for (i=0;i<8;i++)
  { left[i]=0;  right[i]=0;
    for (t=0;t<8;t++)
    {left[i]+=(sayi[2*i]&(bir>> (2*t  ) ) ) << t ; right[i]+= ( sayi[2*i] & ( bir>> (2*t+1) ) ) << (t+1);}
    for (t=0;t<8;t++)
    { left[i] += ( sayi[2*i+1] & ( bir>> (2*t  ) ) ) >> (8-t) ;
      right[i]+= ( sayi[2*i+1] & ( bir>> (2*t+1) ) ) >> (7-t) ;     }   }
  for (k=0;k<son_sayi;k++)
  { fon=my_son+(k*birim); i=0; t=0; say=0;
    do
    { if (++t==2) { t=0; fon++; }     if (((left[i]&right[i])|*fon)==right[i]) say++;
    } while (++i<birim);
    if (say==birim)
    { for (i=k;i<son_sayi-1;i++)
      { for (j=0;j<birim;j++)   *(my_son+(i*birim)+j)=*(my_son+((i+1)*birim)+j);       }
```

```
      *(my_son+((--son_sayi)*birim))=0; k--;
    }   }   }
//*****************************************************************************
void dosya_okuma(FILE *o_d,int y)
{  unsigned int i,j,son_no,soff_no,far *eleman,sayi[8]={0},bir;
   char kr;
   my_son=(unsigned int far *)farcalloc(birim*son_sayi,sizeof(unsigned int));
   my_soff=(unsigned int far *)farcalloc(birim*soff_sayi,sizeof(unsigned int));
   y--;
   while(!feof(o_d))
   { kr=getc(o_d);
     if (kr=='.')
     { do { kr=getc(o_d);} while(kr!='\n'); }
     else
     { son_no=0;  soff_no=0;
       do
       { for (j=0;j<birim;j++)  sayi[j]=0;
           i=0;   bir=32768;  j=0;
           do
           { if (kr=='1') { sayi[j]+=(bir>>i); }              i++;
             if (i==16) { i=0; j++; }
             kr=getc(o_d);
           } while (kr!=' ');
           kr=getc(o_d);  i=0;
           do
           { if (i==y)
             { switch (kr)
               { case '1': eleman=my_son;  eleman+=birim*son_no;
                           for (j=0;j<birim;j++) { *eleman=sayi[j]; eleman++; }
                           son_no++; break;
                 case '0': eleman=my_soff;  eleman+=birim*soff_no;
                           for (j=0;j<birim;j++) { *eleman=sayi[j]; eleman++; }
                           soff_no++; break;
                 default : break;     }    }
             i++; kr=getc(o_d);
           } while (kr!='\n');
           kr=getc(o_d);
       } while (kr!='.');
       break;
     }   }   }
//*****************************************************************************
int dosya_islemi(int s,char *a, char *b)
{  FILE *okunan_dosya, *yazilan_dosya;   char kr,ykr;   unsigned int i,sayi,y,j;
    unsigned long int e1no;
   if (s<3)                              return(1);
   if ((okunan_dosya=fopen(a,"r"))==NULL)    return(2);
   if ((yazilan_dosya=fopen(b,"w"))==NULL)  return(3);
   if ((strcmp(a,"rd84.txt")==0)||(strcmp(a,"spla.txt")==0))
   { tt=1; }
   for (i=0;i<2;i++)
   { kr=getc(okunan_dosya);
     if (kr=='.')
     { kr=getc(okunan_dosya);  ykr=kr;   kr=getc(okunan_dosya);
       kr=getc(okunan_dosya);  sayi=0;
       while(kr!='\n')
       { sayi=sayi*10+my_atoi(kr);  kr=getc(okunan_dosya); }
       switch (ykr)
       { case 'i' : degisken_sayi=sayi; break;  case 'o' : cikis_sayi=sayi;    break;       }     }   }
```

```
birim=(degisken_sayi+15)/16;   birim2=(2*degisken_sayi+15)/16;
fprintf(yazilan_dosya,".i %d\n",degisken_sayi);   fprintf(yazilan_dosya,".o %d\n",cikis_sayi);
for(i=1;i<=cikis_sayi;i++)
{ son_sayi=0;  soff_sayi=0;  imp_sayi=0;  y=i;  y--;
  fclose(okunan_dosya);   okunan_dosya=fopen(a,"r");
  while(!feof(okunan_dosya))
  { kr=getc(okunan_dosya);
    if (kr=='.')
    { do { kr=getc(okunan_dosya); } while(kr!='\n'); }
    else
    { do
        { do { kr=getc(okunan_dosya); } while (kr!=' ');
          kr=getc(okunan_dosya); j=0;
          do
          { if (j==y)
            { switch (kr)
              { case '1': son_sayi++;  break; case '0': soff_sayi++; break;   default : break;       }     }
            j++; kr=getc(okunan_dosya);
          } while (kr!='\n');
          kr=getc(okunan_dosya);
        } while (kr!='.');
        break;       }     }
  fclose(okunan_dosya);        okunan_dosya=fopen(a,"r");
  dosya_okuma(okunan_dosya,i);  sp1_sayi=0;
  if (son_sayi==0)
  { fprintf(yazilan_dosya,"f%d = 0\n\n",i);
    farfree(my_q);      my_q=NULL;       q_sayi=0;     farfree(my_s1);     my_s1=NULL;
    farfree(y_s1);      y_s1=NULL;      s1_sayi=0;       farfree(my_son);    my_son=NULL;
    farfree(my_soff);   my_soff=NULL; soff_sayi=0;     }
  else
  { if (soff_sayi!=0)
    { if (son_sayi<20)       { isaret_max=10; }
        else if (son_sayi<100) { isaret_max=20; }
        else                   { isaret_max=50; }
        if (soff_sayi<10)       q_max=soff_sayi;
        else                    q_max=20;
        if (degisken_sayi<9)   { r_max=32;  }
        else if (son_sayi<129) { r_max=128; }
        else                   { r_max=256; }
        my_q=(unsigned int far *)farcalloc(birim2*q_max,sizeof(unsigned int));
        s1_max=64; my_s1=(unsigned int far *)farcalloc(birim2*s1_max,sizeof(unsigned int));
        y_s1=(unsigned int far *)farcalloc(birim2*s1_max,sizeof(unsigned int));
        my_isaret=(unsigned int far *)farcalloc(birim*isaret_max,sizeof(unsigned int));
        sp1_sayi=0;  sp1_max=64;
        my_sp1=(unsigned int far *)farcalloc(birim*sp1_max,sizeof(unsigned int));
        imp_max=256;
       my_imp=(unsigned int far *)farcalloc(birim2*imp_max,sizeof(unsigned int));
        r_adet=32;   r_sayi=(unsigned int far *)farcalloc(r_adet,sizeof(unsigned int));
        r_bas=(unsigned int far **)farcalloc(r_adet,sizeof(unsigned int));
        for(j=0;j<r_adet;j++)
          *(r_bas+j)=(unsigned int far *)farcalloc(birim*r_max,sizeof(unsigned int));
        while ((son_sayi>0)&(isaret_sayi<son_sayi))
        { q0_yap(1);   s1_yap();   q_sayi=0;   e1no=e1_bul();
          if (e1no)
          { son_cikar(0);    sp1_ekle(0);        }
          s1_sayi=0;    }
        q_sayi=0;  s1_sayi=0; sp2_sayi=0;  sp2_max=32;
        if (son_sayi>0)
```

```
        { my_sp2=(unsigned int far *)farcalloc(birim*sp2_max,sizeof(unsigned int));          }
        while (son_sayi>0)
        { e1no=e2_bul();       son2_cikar(e1no);   sp2_ekle(e1no);       }      }
    sonuc_yaz(yazilan_dosya,i);    }
    farfree(my_son);      son_sayi=0;   farfree(my_soff);    soff_sayi=0;
    farfree(my_sp1);      sp1_sayi=0;    farfree(my_sp2);      sp2_sayi=0;
    farfree(my_isaret); isaret_sayi=0;    farfree(my_s1);       s1_sayi=0;
    farfree(my_q);         q_sayi=0;   }
    fprintf(yazilan_dosya,".e");   fclose(okunan_dosya);   fclose(yazilan_dosya);   return(0);
}
//*****************************************************************************
//                    A N A   P R O G R A M                                    *
//*****************************************************************************
void main (int argc,char *argv[])
{  clrscr();
  switch(dosya_islemi(argc,argv[1],argv[2]))
  {  case 0 : printf(" !!! Program Sorunsuz Calisti. !!!\n");   break;
     case 1 : printf(" !!! Dosya Adlari Unutuldu !!!\n");      break;
     case 2 : printf(" !!! Dosya Okunmak Icin Acilamadi !!!\n");break;
     case 3 : printf(" !!! Dosya Yazmak Icin Acilamadi !!!\n"); break;
  }
}
```