UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

CAROLINE DOMINGUES PORTO DO NASCIMENTO BARBIERI

# APERFEIÇOAMENTO DO MÉTODO CLAUSE-COLUMN TABLE PARA A GERAÇÃO EFICIENTE DE IMPLICANTES PRIMOS

Ilha Solteira

2014

CAROLINE DOMINGUES PORTO DO NASCIMENTO BARBIERI

# APERFEIÇOAMENTO DO MÉTODO CLAUSE-COLUMN TABLE PARA A GERAÇÃO EFICIENTE DE IMPLICANTES PRIMOS

Orientador: Prof. Dr. Alexandre César Rodrigues da Silva

Dissertação apresentada à Faculdade de Engenharia - UNESP – Campus de Ilha Solteira, para obtenção do título de Mestre em Engenharia Elétrica.
Área de Conhecimento: Automação.

Ilha Solteira

2014

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

B236a Barbieri, Caroline Domingues Porto do Nascimento .
Aperfeiçoamento do método clause-column table para a geração eficiente de implicantes primos / Caroline Domingues Porto do Nascimento Barbieri. -- Ilha Solteira: [s.n.], 2014
88 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira. Área de conhecimento: Automação, 2014

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. Geração de Implicantes primos. 2. Minimização de funções booleanas. 3. Métodos de minimização de funções booleanas.

unesp **UNIVERSIDADE ESTADUAL PAULISTA**
**CAMPUS DE ILHA SOLTEIRA**
**FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA**

## CERTIFICADO DE APROVAÇÃO

**TÍTULO:** Aperfeiçoamento do método Clause-Column Table para a geração eficiente de implicantes primos

**AUTORA: CAROLINE DOMINGUES PORTO DO NASCIMENTO BARBIERI**
**ORIENTADOR: Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA**

Aprovada como parte das exigências para obtenção do Título de Mestre em Engenharia Elétrica , Área: AUTOMAÇÃO, pela Comissão Examinadora:

Prof. Dr. ALEXANDRE CESAR RODRIGUES DA SILVA
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. LUIS ANTONIO F DE OLIVEIRA
Departamento de Matemática / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. MARCOS AURELIO BATISTA
Instituto de Biotecnologia / Universidade Federal de Goiás

Data da realização: 11 de novembro de 2014.

Aos meus avós maternos

**Dedico**

# AGRADECIMENTOS

À Deus que por sua presença sempre ouve as minhas orações, e me capacita para tudo aquilo que Ele me destina.

Aos meus avós, pelo modelo de vida passado, com muito esforço e espírito de luta alçaram conquistas gloriosas, sem deixar de lado os maiores valores da vida. Acreditaram, deram apoio, e acima de tudo me ensinaram o poder da fé, elementos essências para esta conquista.

À minha mãe Ângela Cristina Domingues Porto, que me mostrou que é preciso coragem para vencer, um exemplo de força e de mulher. Mãe obrigada pela vida te amo.

Ao meu Pai Wilson Barbieri.

À minha querida tia Carla Cristiane Domingues Porto pelo imenso carinho e principalmente pelas importantes revisões ortográficas, que me ajudaram na escrita deste trabalho.

Ao meu caro orientador professor Dr. Alexandre César Rodrigues da Silva pelas oportunidades que me proporcionou, conselhos, disponibilidade, paciência e principalmente pelo grande respeito durante o período em que trabalhamos juntos.

Ao Samuel, companheiro e grande amigo, a quem devo a felicidade e o amor.

Aos meus amigos Jean, Uilian, Melissa, Alexandre e Cindy pela amizade e ajuda.

A CAPES pelo auxílio financeiro.

A todos que contribuíram indiretamente para a realização deste trabalho.

“A persistência é o menor caminho para o êxito.”

**Charles Chaplin**

# Resumo

A geração eficiente de implicantes primos é um fator importante na fase de cobertura dos mintermos em métodos de minimização de funções booleanas. Este trabalho apresenta uma versão aprimorada do método denominado de Clause-Column Table, utilizado na geração de implicantes primos. Neste novo algoritmo adicionou-se o teorema da adjacência e um novo critério de parada. Estas modificações evitaram a geração de termos nulos e iterações desnecessárias que ocorriam no algoritmo original. O algoritmo original e o aprimorado foram implementados em linguagem C e comparados. O método Clause-Column Table Aprimorado também foi comparado com o método Quine-McCluskey e Expander. Os resultados comprovaram que a versão aprimorada gera menos iterações que a versão original, e que na maioria das funções analisadas evitou-se a geração de termos nulos. Ao comparar com o método de Quine-McCluskey e o Expander comprovou-se que o método Clause-Column Table Aprimorado é superior na geração dos implicantes primos, pois em alguns casos elimina aqueles que não são necessários para a cobertura da função. De posse dos implicantes primos o problema de cobertura dos mintermos foi formulado como um problema de programação linear inteira 0 e 1, em que a solução se abre a todos os avanços ocorridos na área de programação linear visando a obtenção de uma solução mínima.



**Palavras – Chave:** Minimização. Função booleana. Implicantes primos.

## Abstract

Efficient generation of prime implicants is an important factor in the coverage phase of minterms in minimization's methods of Boolean functions. This research presents an improved version of the method called Clause-Column Table, used to generate prime implicants. In this new algorithm was added to the adjacency theorem and a new stopping criterion. These modifications prevented the generation of null terms and unnecessary iterations that occurred in the original algorithm. The original and improved algorithms were implemented in C language and compared. The Clause-Column Table Improved method was compared with the Expander and Quine-McCluskey method. The results proved that the improved version generates fewer iterations than the original version, and that in most functions analyzed it was avoided the generation of null terms. Comparing Quine-McCluskey method and the Expander it was proved that the Clause-Column Table Enhanced method is superior in the generation of prime implicants, since in some cases eliminates those who are not required to cover the function. In ownership of the prime implicants the cover problem of minterms was formulated as an integer linear programming problem of 0 and 1, where the solution is open to all advances in the area of linear programming in order to obtain a minimal solution.


**Key – Words**: Minimization.  Boolean function and Prime implicants.

# LISTA DE FIGURAS

## LISTA DE TABELAS

# SUMÁRIO

# 1 INTRODUÇÃO

A utilização de circuitos digitais tem sido amplamente disseminada com o avanço da tecnologia e principalmente dos dispositivos eletrônicos digitais, o que amplia e mantém a relevância da pesquisa em métodos de minimização de funções booleanas. O emprego em diferentes tipos de aplicações como, por exemplo, telecomunicações, controle e automação também têm contribuído para a constante evolução dos projetos de circuitos digitais.

A convergência destes avanços resultou num grande desafio. É preciso lidar com circuitos digitais cada vez maiores e mais complexos. Além disso, é necessário fazer com que o tempo e o custo de implementação sejam sempre decrescentes. A complexidade está no fato de que cada componente utilizado para implementar o circuito, ocupa um determinado espaço físico para produzí-lo. Assim, diversos pesquisadores têm recorrido à otimização lógica, que é uma forma de obter circuitos minimizados e com a mesma funcionalidade.

A ideia básica empregada na minimização de funções booleanas é trabalhar com as funções através dos métodos de geração de implicantes primos. O conjunto de axiomas e teoremas pertencentes à álgebra de Boole permite a realização de simplificações algébricas de uma função booleana, que mantém o resultado inalterado diminuindo o número de termos e literais. Pelas regras de implementação o número de termo produto da expressão é proporcional ao número de portas lógicas no circuito.

É neste cenário de minimização de funções booleanas que o presente trabalho se insere, propondo uma melhoria no método desenvolvido por Das (1972), adicionando ao algoritmo original o teorema da adjacência e um novo critério de parada. Estas modificações evitaram a geração de termos nulos e iterações desnecessárias que ocorriam no algoritmo original.

Os algoritmos foram implementados em linguagem C e os resultados comprovaram que a versão aprimorada é mais eficiente que a original, pois o número de iterações na geração dos implicantes primos é menor. Cabe salientar que na maioria dos casos analisados os termos nulos não foram gerados, na versão aprimorada.

O método Clause-Column Table Aprimorado também foi comparado com a primeira fase do método de Quine-McCluskey e com o programa Expander que implementa o Teorema de Expansão de Shannon, comprovando que o método aprimorado é superior na geração de implicantes primos.

De posse dos implicantes primos, o problema de cobertura dos mintermos é formulado como um problema de programação linear inteira 0 e 1.

O trabalho está organizado da seguinte forma: Neste Capítulo apresenta-se uma introdução da importância da geração eficiente de implicantes primos no contexto de minimização de funções booleanas.

No Capítulo 2, apresentam-se os conceitos e a importância da álgebra de Boole para a simplificação de circuitos digitais.

No Capítulo 3, apresentam-se alguns métodos de geração de implicantes primos encontrados na literatura.

No Capítulo 4, aborda-se em detalhes os métodos de geração de implicantes primos denominados Quine-McCluskey e Teorema de Expansão de Shannon.

No Capítulo 5, apresenta-se o método de Das desenvolvido em 1972 para a geração eficiente de implicantes primos. Na seção 5.1, descreve-se as definições do método Clause-Column Table e na seção 5.2, o algoritmo para melhor esclarecimento.

No Capítulo 6, apresenta-se o método de Das que foi aprimorado eliminando algumas deficiências encontradas. Descreve-se na seção 6.1 o algoritmo para melhor esclarecimento, na seção 6.2 é exposto a cobertura dos mintermos como sendo um problema de programação linear inteira 0 e 1, e na última seção a estrutura de dados utilizada na implementação dos métodos Clause-Column Table e Clause-Column Table Aprimorado.

No Capítulo 7, os gráficos e tabelas dos resultados obtidos são apesentados.

E por último apresentam-se as conclusões obtidas através da realização deste trabalho de pesquisa.

# 2 ÁLGEBRA DE BOOLE

Os conceitos matemáticos utilizados na construção de circuitos digitais são conhecidos como Álgebra de Boole, onde define-se somente dois valores para as variáveis, ou seja, uma variável pode assumir o valor 1 ou 0, que representam a condição verdadeira ou falsa, respectivamente.

Assim como em outras áreas da matemática na álgebra booleana existem regras, leis e postulados.

Qualquer circuito lógico, não importando sua complexidade, pode ser descrito usando as três operações Booleanas básicas definidas com OR, AND e NOT. No entanto, para compreendê-las é necessário definir alguns termos, como, variável, complemento e literal. Uma variável é um símbolo que pode assumir o valor lógico 0 ou 1, cujo complemento é a negação da mesma, por exemplo, a variável A quando negada é representada por $\overline{A}$ (lido como "A barrado" ou "A negado"). Se A = 1, então $\overline{A} = 0$. Um literal é a variável ou o complemento de uma variável.

A adição booleana corresponde à função de uma porta lógica "OR", em que o termo-soma é uma soma de literais. Em circuitos lógicos, um termo soma é produzido por uma operação OR, que será igual a 1 quando um ou mais literais do termo for 1 e igual a 0 somente se cada um dos literais for 0, conforme apresentado a seguir:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 1$$

A multiplicação booleana corresponde à função de uma porta lógica "AND", cujo termo-produto é o produto de literais. Em circuitos lógicos, um termo-produto é resultado de uma operação AND, que será igual a 1 se cada um dos literais no termo for igual a 1 e 0 quando um ou mais literais for 0. Sendo assim:

$$0 . 0 = 0$$

$$0 . 1 = 0$$

$$1 . 0 = 0$$

$$1 . 1 = 1$$

A operação de complementação é simplesmente o valor complementar ao que a variável assume. Visto que uma variável booleana poder assumir um entre somente dois valores, o valor complementar será 1 se a variável vale 0 e será 0 se a variável vale 1. Portanto:

$$\bar{0} = 1$$

$$\bar{1} = 0$$

## 2.1 LEIS E POSTULADOS DA ÁLGEBRA DE BOOLE

Define-se expressão booleana como sendo uma combinação finita de variáveis booleanas (0, 1), através das operações OR, AND e NOT.

Uma expressão pode ser manipulada e simplificada algebricamente através das leis e postulados empregadas na álgebra de Boole. Apresentam-se a seguir alguns postulados e leis que referem-se à álgebra de Boole.

$$A + 0 = A$$

$$A + 1 = 1$$

$$A + A = A$$

$$A + \bar{A} = 1$$

$$A . 0 = 0$$

$$A . 1 = A$$

$$A . A = A$$

$$A . \bar{A} = 0$$

$$\bar{\bar{A}} = A$$

**Lei Comutativa:**

- **Para adição:** quando a operação OR é aplicada a ordem das variáveis não interfere.

$$A+ B = B + A$$

- **Para multiplicação:** quando a operação AND é aplicada a ordem das variáveis não interfere.

$$A . B = B . A$$

**Lei Associativa:**

- **Para adição:** quando a operação OR é aplicada em mais de duas variáveis, o resultado é o mesmo independente da forma em que foram agrupadas.

$$A+ (B + C) = (A+B) + C$$
$$(A+ B) + C = A+ (B+ C)$$

- **Para multiplicação:** quando a operação AND é aplicada em mais de duas variáveis, o resultado é o mesmo independente da forma em que foram agrupadas.

$$A. (B.C) = (A.B). C$$
$$(A. B). C = A.(B. C)$$

**Lei Distributiva:**

Uma expressão pode ser expandida multiplicando cada termo da expressão, como na álgebra convencional. Tal lei indica que pode-se fatorar (colocar em evidência uma variável) a expressão, ou seja, dada uma soma de dois ou mais termos e cada termo contiver uma variável em comum, essa variável poderá ser colocada em evidência.

$$A . (B + C) = A . B + A . C$$

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

Todas as leis e postulados descritos nesta seção são uteis na simplificação de expressões lógicas, ou seja, na redução do número de termos. Dessa forma, a expressão reduzida produz um circuito mais simples do que o produzido pela expressão original.

## 2.2 TEOREMA de DE MORGAN

O teorema de De Morgan define que o complemento de um produto (lógico) equivale à soma (lógica) das negações de cada variável do referido produto. Sob a forma de equação, têm-se:

$$\overline{A.B.C} = \overline{A} + \overline{B} + \overline{C}$$

De Morgan também definiu o dual do teorema anterior, ou seja, o complemento de uma soma (lógica) equivale ao produto das negações individuais das variáveis:

$$\overline{A + B + C} = \overline{A} \cdot \overline{B} \cdot \overline{C}$$

Como um exemplo da aplicação do teorema de De Morgan para duas variáveis, têm-se:

$$\overline{A \cdot \overline{B}} = \overline{A} + B$$

$$\overline{\overline{A} + \overline{B}} = A \cdot B$$

## 2.3 FUNÇÕES BOOLEANAS

Define-se função booleana como sendo uma expressão lógica em que cada uma das n variáveis podem assumir independentemente o valor 0 ou 1, assim existirão $2^n$ possíveis combinações para as variáveis a serem consideradas na determinação do valor da função.

Tem-se, como exemplo, a expressão W = A + B + C, onde W representa a função Booleana, que depende das variáveis A, B e C.

Dada à expressão que descreve uma função booleana pode-se determinar como a função se comporta para qualquer combinação das variáveis de entrada. O comportamento da função

é descrito através da tabela verdade, que consiste num conjunto de colunas, onde são listadas todas as combinações possíveis entre as variáveis de entrada e a saída da função. Na Tabela 1 apresenta-se a tabela verdade para a função W (A, B, C) = A + B + C.

**Tabela 1** – Tabela verdade para a função W (A, B, C) = A + B + C

| **A** | **B** | **C** | **W= A + B + C** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

Fonte: Próprio Autor

Uma função booleana pode ser representada através da soma de produto ou produto de soma.

### 2.3.1 Soma de produtos

A cada combinação das variáveis de entradas pode-se associar um termo-produto em que todas as variáveis da função estão presentes, e que é construído da seguinte forma: se a variável assume o valor 0 ela deve aparecer barrada, e se a variável assume 1 deve-se aparecer não-barrada. Como é exemplificado na Tabela 2.

**Tabela 2 –** Tabela apresentando a forma canônica de soma de produtos

| A | B | C | Mintermo |
|---|---|---|---|
| 0 | 0 | 0 | $\bar{A}.\bar{B}.\bar{C}$ |
| 0 | 0 | 1 | $\bar{A}.\bar{B}.C$ |
| 0 | 1 | 0 | $\bar{A}.B.\bar{C}$ |
| 0 | 1 | 1 | $\bar{A}.B.C$ |
| 1 | 0 | 0 | $A.\bar{B}.\bar{C}$ |
| 1 | 0 | 1 | $A.\bar{B}.C$ |
| 1 | 1 | 0 | $A.B.\bar{C}$ |
| 1 | 1 | 1 | $A.B.C$ |

Fonte: Próprio Autor

Cada termo produto formado com todas as variáveis na forma complementada ou não é denominado mintermo. Note que, para um dado mintermo, se os valores das variáveis associadas forem substituído, obtém-se 1. Sendo assim, para obter a equação em soma de produtos de uma função a partir de sua tabela verdade, é necessário realizar a operação OR entre os mintermos associados aos 1s da função.

**Tabela 3** - Tabela verdade representada pelos mintermos

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **0** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **1** |
| 1 | 0 | 0 | **0** |
| 1 | 0 | 1 | **1** |
| 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | **0** |

Fonte: Próprio Autor

A Tabela 3 ilustra que F é função das variáveis A, B e C. Os valores de A, B e C para os quais F=1 são (0,1,0), (0, 1, 1), (1,0,1) e (1,1,0). Portanto, os mintermos associados a essas condições são $\overline{A}.B.\overline{C}$ , $\overline{A}.B.C$ , $A.\overline{B}.C$ e $A.B.\overline{C}$, respectivamente. Por conseguinte, a equação em forma de soma de produtos para F é a operação OR entre os produtos:

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

### 2.3.2 Produto de somas

Para obter equações em forma de produto de soma, basta obter o dual da equação na forma de soma de produtos. A cada combinação das variáveis de entrada pode-se associar um termo produto em que todas as variáveis da função estão presentes, e que é construído da seguinte forma: se a variável assume o valor 1 deve-se aparecer barrada e se a variável assume 0 deve-se aparecer não-barrada, como apresentado na Tabela 4.

**Tabela 4** - Tabela apresentando a forma canônica de produto de somas

| A | B | C | Maxtermo |
|---|---|---|---|
| 0 | 0 | 0 | $A+B+C$ |
| 0 | 0 | 1 | $A+B+\overline{C}$ |
| 0 | 1 | 0 | $A+\overline{B}+C$ |
| 0 | 1 | 1 | $A+\overline{B}+\overline{C}$ |
| 1 | 0 | 0 | $\overline{A}+B+C$ |
| 1 | 0 | 1 | $\overline{A}+B+\overline{C}$ |
| 1 | 1 | 0 | $\overline{A}+\overline{B}+C$ |
| 1 | 1 | 1 | $\overline{A}+\overline{B}+\overline{C}$ |

Fonte: Próprio Autor

Cada termo soma formado anteriormente é denominado maxtermo. Note que, para um dado maxtermo, se os valores das variáveis associadas forem substituído, obtém-se 0. Sendo assim, para obter a equação em produto de soma de uma função a partir de sua tabela verdade, é necessário realizar a operação AND entre os maxtermos associados aos 0s da função.

**Tabela 5** - Tabela verdade representada pelos maxtermos.

| A | B | C | F |
|---|---|---|---|
| 0 | 0 | 0 | **0** |
| 0 | 0 | 1 | **0** |
| 0 | 1 | 0 | **1** |
| 0 | 1 | 1 | **1** |
| 1 | 0 | 0 | **0** |
| 1 | 0 | 1 | **1** |
| 1 | 1 | 0 | **1** |
| 1 | 1 | 1 | **0** |

Fonte: Próprio Autor

A Tabela 5, ilustra que F é função das variáveis A, B e C. Os valores de (A, B, C) para os quais F=0 são (0,0,0), (0,0,1), (1,0,0) e (1,1,1). Os maxtermos 0 associados a essas condições são $A + B + C$, $A+B+\overline{C}$, $\overline{A} + B + C$ e $\overline{A} + \overline{B} + \overline{C}$ respectivamente. Por conseguinte, a equação em soma de produtos para F é a operação AND entre os produtos:

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

## 2.4 SIMPLIFICAÇÃO DE CIRCUITOS LÓGICOS

A utilização prática da álgebra Booleana é na simplificação de expressões ou funções que representam um determinado circuito lógico.

Uma vez obtida à expressão de um circuito lógico pode-se reduzi-la a uma forma mais simples, obtendo-se o número mínimo de termos ou variáveis da expressão, resultando em circuito de menor custo. Neste trabalho adotou-se como critério de custo a quantidade de entradas nas portas lógicas. Existem diversos procedimentos de minimização de funções booleanas, sendo que os mais frequentes são os métodos algébricos, mapa de Karnaugh e métodos tabulares.

### 2.4.1 Método algébrico

Através dos postulados e teoremas presentes na Álgebra de Boole, expostos nas seções 2.1 e 2.2, pode-se simplificar expressões booleanas. Nem sempre é evidente qual teorema ou postulado deve ser aplicado para obter o resultado mais simplificado. Entretanto com a experiência, bons resultados podem ser obtidos.

Para exemplificar a simplificação de expressões booleanas através do método algébrico considere a função F (A, B, C) = $\overline{A}B\overline{C} + \overline{A}BC + A\overline{B}C + AB\overline{C}$.

O primeiro passo é identificar pares de mintermos que se diferenciam por apenas um literal, para que a lei distributiva seja aplicada. Os mintermos $\overline{A}B\overline{C}$ e $\overline{A}BC$, por exemplo, possuem os mesmo literais exceto pelo literal C. Sendo assim, pode-se fatorar esses mintermos, obtendo-se:

$$F = \overline{A}B\overline{C}+\overline{A}BC+A\overline{B}C+AB\overline{C}$$

$$F = \overline{A}B+A\overline{B}C+AB\overline{C}$$

$$F = \overline{A}B +\overline{B}C +A\overline{B}C$$

$$F = \overline{A}B + A\overline{B}C +B\overline{C}$$

Assim, pela simplificação algébrica, obteve-se uma expressão em soma de produtos que é simplificada, pois o número de operações e literais foram reduzidos ao mínimo.

### 2.4.1 Mapa de Karnaugh

O mapa de Karnaugh é um método gráfico utilizado para simplificar expressões de circuitos lógico, identificando visualmente grupos de mintermos ou maxtermos que podem ser simplificados. É formado por $2^n$ células, em que n é o número de variáveis de entrada, e cada célula representa um mintermo, maxtermo ou *don't care state* (DAGHLIAN, 1995).

No mapa de Karnaugh de uma função representada na forma canônica produto de somas, as células correspondentes aos maxtermos da função têm o valor 0 e as restantes têm o valor 1. Qualquer par de células na horizontal ou vertical (células adjacentes) corresponde a mintermos/maxtermos que diferem em apenas um literal. As células na coluna mais à direita são adjacentes às células da coluna mais da esquerda, bem como, as células na linha superior

são adjacentes às células da linha inferior. A Figura 1 ilustra o mapa de Karnaugh de 4 variáveis, onde cada célula representa um número decimal. Este número é formado pelo valor lógico 0 ou 1, em que as variáveis A, B, C e D podem assumir em cada célula.

**Figura 1-** Mapa de Karnaugh com 4 variáveis de entrada.

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **00** | 0 | 1 | 3 | 2 |
| **01** | 4 | 5 | 7 | 6 |
| **11** | 12 | 13 | 15 | 14 |
| **10** | 8 | 9 | 11 | 10 |

Fonte: Próprio Autor

Na Figura 2 apresenta-se o mapa de Karnaugh para a função $F(A, B, C) = \sum m\ (2, 6)$. O mapa contém um par de 1s que são adjacentes verticalmente; o primeiro representa $A\overline{B}\overline{C}$ e o segundo, $A\overline{B}C$.

**Figura 2** – Mapa de Karnaugh da função $F\ (A, B, C) = \sum m\ (2, 6)$

| A \ BC | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| **0** | | | | 1 |
| **1** | | | | 1 |

$B\overline{C}$

Fonte: Próprio Autor

Quando uma variável aparece nas formas complementada e não complementada em um agrupamento, a variável é eliminada da expressão. As variáveis que não alteram para todos os quadros do agrupamento permanecem na expressão final.

Deve ficar claro que quanto maior o grupo de 1s mais variáveis são eliminadas. Esse princípio é usado para se obter a expressão lógica simplificada a partir do mapa de Karnaugh.

O mapa de Karnaugh possui diversas vantagens em relação ao método algébrico utilizado para simplificar expressões. Normalmente o mapa requer menos passos, principalmente para expressões que contenham muitos termos e sempre gera a expressão mínima.

Existem técnicas diferentes, porém mais complexas, que são utilizadas para minimizar funções booleanas com mais de seis variáveis de entrada. Tais técnicas são mais adequadas para funções com muitas variáveis onde o método algébrico e o mapa de Karnaugh são impraticáveis. A maioria dessas técnicas ou métodos de minimização podem ser implementadas computacionalmente para realizar o processo de simplificação (TOCCI, 2007).

No próximo capítulo apresentam-se alguns métodos de geração de implicantes primos encontrados na literatura.

# 3 MÉTODOS DE GERAÇÃO DE IMPLICANTES PRIMOS

Na implementação de circuitos digitais as funções booleanas devem ser minimizadas com o objetivo de encontrar um circuito equivalente, em que o custo de implementação seja o mínimo possível. Além disso, o algoritmo de minimização deve ter um tempo de execução aceitável. Vários são os critérios de custo descritos na literatura.

Encontrar uma solução mínima é um problema de difícil abordagem, na medida em que, seria necessário utilizar métodos exatos e exaustivos, que garantissem a cobertura de todas as possibilidades de simplificação. Assim, a minimização exata requer muitos recursos computacionais, visto que a complexidade do algoritmo aumenta exponencialmente conforme o número de variáveis de entrada (SILVA, 1993). A fim de contornar tal situação, pesquisadores recorrem a métodos heurísticos, que geram soluções aproximadamente mínimas num tempo de execução aceitável, pois não há necessidade de buscá-las exaustivamente até encontrar uma solução ótima.

Segundo Besslich (1979), é difícil decidir qual método é mais eficiente para um determinado problema de minimização de funções booleanas, pois as técnicas de simplificação utilizadas influenciam diretamente na geração dos implicantes primos e em alguns casos, por exemplo, é mais indicado o uso de técnicas heurísticas e em outros, o de técnicas exatas que por sua vez garantem uma solução mínima. Portanto, classificam-se neste trabalho os métodos em exatos, heurísticos, e os que utilizam diagrama de decisão binário (DDB).

Encontram-se na literatura diversos métodos de minimização de funções booleanas, considerados exatos, que geram todos os implicantes primos, para então obter uma solução ótima, sendo que a maioria foi desenvolvida baseada na forma tabular de representação de funções booleanas (McCLUSKEY, 1956; SAFAEI, 2007; JAIN, 2008; PRASAD, 2009; QAWASNEH, 2009; RATHORE, 2014).

Rathore (2014) propôs um método tabular, onde uma função booleana de n variáveis é mapeada sistematicamente por meio da tabela verdade. O tamanho do mapa é reduzido em 50% de linhas ou colunas, eliminando-se uma das variáveis de cada vez, na coluna ou na linha. O procedimento é repetido até que o tamanho do mapa torne-se 1 x 1, contendo um único elemento. Devido à simplificação desenvolvida a partir das expressões lógicas, a função mínima é garantida.

Tal método apresenta-se mais simples do que o mapa de Karnaugh e o Quine-McCluskey, podendo lidar com qualquer número de variáveis.

Prassad (2009) descreveu um modelo para estimar a complexidade de circuitos integrados digitais, como base no método de simplificação de Quine-McCluskey, propondo um modelo que permite analisar a viabilidade do projeto e o desempenho antes da realização do circuito. O comportamento do método foi analisado, utilizado com base para diferentes números de termos produto e introduziu-se um modelo matemático para prever a complexidade no espaço booleano.

Como mencionou-se anteriormente os métodos heurísticos surgiram a fim de sanar os problemas originados com a execução dos métodos exatos, procurando produzir soluções, utilizando poucos recursos computacionais (MOHAMED, 1997; LIN, 2009; AMARU, 2014).

O programa MINI (HONG, 1974) foi um dos primeiros algoritmos heurísticos, desenvolvido pela IBM em meados da década de 70. Em seguida surgiu o programa Presto (1981). Já no final da década de 80 surgiram métodos importantes na história da minimização, como o proposto por Brayton (1984) denominado Espresso. O Espresso foi desenvolvido combinando as melhores técnicas heurísticas utilizadas nos métodos MINI e PRESTO. O algoritmo começa expandindo todos os implicantes ao máximo. Os implicantes que são cobertos por um implicante expandido são removidos. Esse processo é chamado de EXPANSÃO. Em seguida, é executado o processo de COBERTURA IRREDUNDANTE, que é semelhante à tabela de cobertura do método de Quine-McCluskey. Uma cobertura irredundante é obtida através dos implicantes primos expandidos encontrados na fase de expansão. Dessa forma, a primeira solução é encontrada. Contudo, essa solução pode não ser a mais satisfatória, podendo haver outras soluções melhores, como por exemplo, com menos termos ou literais. Assim, para encontrar uma cobertura melhor, o algoritmo reduz os implicantes primos para o menor número de literais possíveis, que ainda cobre a função booleana original. Tal processo é chamado de REDUÇÃO. Os processos de EXPANSÃO-REDUÇÃO são executados iterativamente até que uma cobertura melhore a última encontrada.

O Espresso tornou-se tão bem sucedido que foi incorporado em diversas ferramentas de minimização de funções booleanas. Também foi utilizado como parâmetro de comparação para validar métodos que surgiram posteriormente reivindicando superioridade em diversos aspectos (BISWAS, 1996; FISER, 2004; HAVLICKA, 2001; MIKHTONYUK, 2008).

A ferramenta de minimização de funções booleanas denominada BOOM (Boolean Minimization), desenvolvida por Hlavicka (2001) simplifica funções booleanas com única ou com múltiplas saídas, mostrando ser vantajosa, sobretudo para problemas com grandes quantidade de variáveis de entrada e com grande número de *don´t care states*, superando o

método Espresso, tanto em minimalidade quanto em tempo de execução, que cresce linearmente conforme o número de variáveis aumenta.

O algoritmo consiste em duas fases principais: a geração de implicantes primos e a solução do problema de cobertura (PC).

A geração de implicantes para funções de uma única saída é realizada em duas etapas: a Busca direta de cobertura (CD-Search) que gera um conjunto suficiente de implicantes necessários para cobrir a função inicial e, em seguida, os implicantes gerados são expandidos em implicantes primos na fase de Expansão dos implicantes (IE). As fases são executadas repetidamente, a fim de obter melhores resultados.

Em relação à minimização de funções booleana com múltiplas saídas o algoritmo executa outras fases que levam em conta a solução da cobertura do grupo e de cada saída independentemente.

Mikhtonyuk (2008) propôs um método a fim de superar o Espresso em alguns aspectos. A implementação do método foi baseada em execução paralela. Trata-se de uma forma de computação em que os cálculos são realizados simultaneamente, atuando sob o princípio de que grandes problemas podem ser divididos em problemas menores, que são resolvidos concorrentemente. Assim, o custo e o tempo de implementação são reduzidos na minimização de funções booleanas, especificadas na forma canônica e com grande número de variáveis. O método *Boolean Function Minimization Based on Intel Architecture* demonstrou superar significativamente o método Espresso em tempo de execução e sem perda de qualidade da função.

Existem também os métodos heurísticos que utilizam algoritmos genéticos a fim de melhorar a qualidade na geração dos implicantes primos (HATA, 1997; FISER, 2001; ASTHANA, 2014).

Fiser (2001) propôs um método de minimização de funções booleanas utilizando uma abordagem original para a geração de implicantes por inclusão de literais. É feita uma seleção dos literais recém-incluídos baseando-se em heurísticas, que trabalham com a frequência de ocorrência de literal, ou seja, o literal que aparece o maior número de vezes na função. Em vez de utilizar os literais diretamente para a geração de implicantes, são realizadas algumas mutações na escolha aleatória dos literais para melhorar a qualidade dos implicantes gerados. Essas mutações são muito eficientes para a busca de uma solução ótima, pois além de contornar o problema de mínimos, altera levemente a direção da busca. Na implementação foi utilizado o método BOOM, considerado eficiente especialmente por minimizar funções com centenas de variáveis de entrada. O método proposto foi testado tanto em *benchmarks* padrão quanto em

problemas de maior dimensão, gerados aleatoriamente. Os resultados comprovaram que o novo algoritmo é muito rápido e que, para grandes circuitos proporciona melhores resultados do que o Espresso. O critério de qualidade selecionado foi a soma do número de literais e termos. Em todos os casos, encontrou-se uma solução igual ou melhor do que a gerada pelo BOOM, e em menor tempo de execução do que o Espresso.

Igualmente Hata (1997) utilizou em seu trabalho algoritmo genético e heurística para minimizar funções lógicas com valores múltiplos. Codificou-se uma função lógica com múltiplos valores como um "cromossomo", cujo comprimento permite ser alterado e corresponde ao número de implicantes da expressão. A função de aptidão avalia os seguintes itens: como as saídas podem representar corretamente expressão lógica, de quantos implicantes uma expressão lógica necessita e quantas conexões são necessárias para uma expressão lógica.

O método emprega a função objetivo, sistemas de recombinação e mutação. Essas operações podem minimizar expressões lógicas. Os resultados experimentais mostram que o método gera bons resultados em relação ao número de implicantes necessários nas expressões de minimização e evita a solução mínima local, comparada com o método baseado em computação neural.

Atualmente, Sarkar (2013) desenvolveu um algoritmo combinando meta-heurísticas e técnicas de algoritmos genéticos, a fim de minimizar circuitos reversíveis, os quais se caracterizam por ter o mesmo número de entradas e saídas.

Devido ao baixo consumo de energia, a perda de informação inerente e outros fatores associados à computação, circuitos reversíveis estão se tornando cada vez mais importante em termos de computação para os dias atuais e futuros. No entanto, devido a várias abordagens de síntese da lógica booleana clássica como Mapa de Karnaugh e o método de Quine-McCluskey não poderem ser aplicado diretamente para sintetizar uma lógica reversível, foi proposto uma versão modificada do método Quine-McCluskey, incluído os algoritmo genéticos *Simulated Annealing* (SA) e *Ant Colony Optimization* (ACO).

Esta abordagem produz circuitos ótimos ou quase ótimos para a maioria dos casos. A principal vantagem deste algoritmo é que fatores constantes podem ser modificados ou ajustados pelo usuário para obter um melhor resultado.

No mesmo contexto métodos baseados em BDD's (Diagrama de Decisão Binária) têm sido amplamente empregados para minimizar funções booleanas. Foram sugeridos por Akers (1978) no final dos anos 70, direcionando a uma abordagem própria do problema de minimização. O conceito básico é converter a lista de mintermos em uma árvore binária, em que os nós são elementos de decisão e os ramos são associados aos valores lógicos 0 e 1,

dependendo da decisão tomada (AKERS, 1978; OLIVEIRA 1998; LIN 2005; CHOWDURY, 2010).

Relacionados aos métodos de utilização de BBD's, foram encontrados algumas características, como por exemplo, a ordem das variáveis na construção do diagrama binário influencia diretamente na geração da árvore minimizada ideal. Portanto, existe a possibilidade de não se encontrarem, após a minimização, as funções mais otimizadas possíveis. Existem, contudo, algoritmos que encontram uma ordenação eficaz para estruturar uma árvore binária, cuja minimização conduz ao diagrama ótimo, como nos trabalhos de (WU, 2000; HALDER, 2007; BANSAL, 2013).

No capítulo seguinte apresentam-se dois métodos de geração de implicantes primos, que foram utilizados a fim de validar o método proposto neste trabalho, o Quine-McCluskey e o Teorema de Expansão de Shannon.

# 4 Métodos de Quine-McCluskey e o Teorema de Expansão de Shannon

## 4.1 MÉTODO DE QUINE-MCCLUSKEY

O método clássico de Quine-McCluskey é utilizado para a minimização de funções booleanas, empregando-se procedimentos tabulares e iterativos com o uso sistemático da propriedade $AB+A\overline{B}=A$.

O método consiste em duas fases. Na primeira gera-se todos os implicantes primos e na segunda realiza-se uma cobertura mínima encontrando os implicantes primos que cobrem os mintermos da função.

Conforme definido anteriormente um mintermo é um termo produto em que todas as variáveis aparecem uma vez na forma de literal.

Define-se como implicante de uma função booleana como sendo uma cobertura de um ou mais mintermos. Quando um implicante não pode ser mais reduzido é considerado um implicante primo.

Define-se como *don't care states* as combinações das variáveis de entrada que correspondem a situações irrelevantes para o funcionamento do circuito lógico.

Neste contexto, minimizar uma função lógica é equivalente a encontrar um conjunto ótimo de implicantes primos que realize a função desejada com o menor custo.

Alguns critérios de custo definidos na literatura são:

- Menor número de literais (um literal é uma variável na forma complementada ou não);
- Menor número de literais numa expressão do tipo soma de produto ou produto de soma;
- Menor número de termos numa expressão.

### 4.1.1 Algoritmo para a geração de implicantes primos através do Quine-McCluskey

A primeira fase do método Quine-McCluskey consiste nos seguintes passos:

1. Listar todos os mintermos, na representação binária, e agrupá-los pelo número de dígitos 1's;

2. Comparar os mintermos adjacentes. Os mintermos que combinarem pelo menos uma vez devem ser marcados, isto é, dois termos são combináveis se diferirem apenas um dígito na sua representação binária. As posições dos dígitos que diferem recebem a notação X, formando assim um novo grupo;
3. Combinar da mesma forma os implicantes gerados no grupo anterior. O processo deve ser repetido até que não haja mais combinações;
4. Os mintermos que não foram marcados são considerados implicantes primos da função.

Aplica-se a primeira fase do método de Quine-McCluskey a função F(A, B, C, D) = ∑m(2, 3, 5, 7, 11, 13), cujo mapa de Karnaugh está apresentado na Figura 3.

**Figura 3**- Representação dos mintermos no mapa de Karnaugh

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | | | 1 | 1 |
| 01 | | 1 | 1 | |
| 11 | | 1 | | |
| 10 | | | 1 | |

| AB \ CD | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 3 | 2 |
| 01 | 4 | 5 | 7 | 6 |
| 11 | 12 | 13 | 15 | 14 |
| 10 | 8 | 9 | 11 | 10 |

**Fonte:** Próprio Autor

Na Figura 4, apresentam-se os grupos formados a partir dos mintermos da função na representação binária. Os mintermos agrupados pelo número de 1s', que diferem em apenas um dígito são combinados e, portanto são marcados com ∗.

**Figura 4** - Grupos do método Quine-McCluskey

| Grupo 0 | Grupo 1 |
|---|---|
| 0010 * | 001X |
| 0011 * | 0X11 |
| 0101 * | 01X1 |
| 0111 * | X011 |
| 1011 * | X101 |
| 1101 * | |

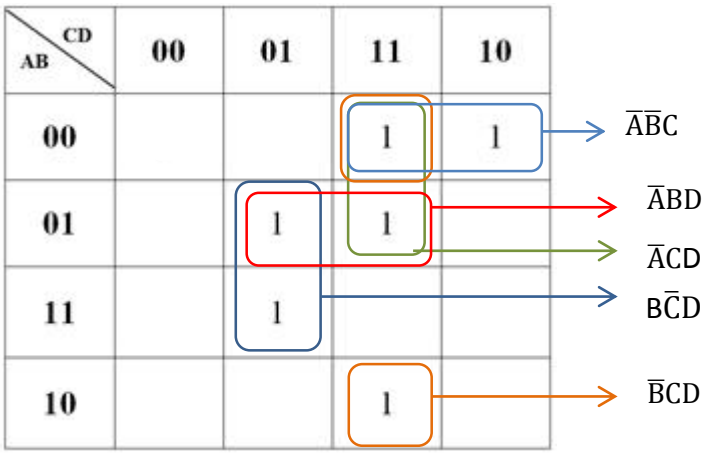
Fonte: Próprio Autor

Destaca-se que os mintermos do Grupo 1 não foram combinados e, portanto não foram marcados, formando o conjuntos dos implicantes primos da função F(A, B, C, D) = ∑ m(2, 3, 5, 7, 11, 13).

Os implicantes primos obtidos são: $001X + 0X11 + 01X1 + X011 + X101 = \bar{A}\,\bar{B}C + \bar{A}CD + \bar{A}BD + \bar{B}CD + B\bar{C}D$.

Através do mapa de Karnaugh, apresentado na Figura 5, pode-se analisar as coberturas geradas com a execução da primeira fase do método de Quine-McCluskey.

**Figura 5-** Comportamento do método Quine-McCluskey no mapa de Karnaugh




Fonte: Próprio Autor

Na segunda fase do método Quine-McCluskey realiza-se uma cobertura mínima, encontrando os implicantes primos que cobrem a função. Os seguintes passos são realizados.

1- Cria-se uma tabela onde as linhas representam implicantes primos e as colunas os mintermos da função. Caso exista *don't care states*, os mesmos não devem aparecer na tabela;
2- Marca-se com um X as interseções entre os mintermos (colunas) cobertos por cada implicante primo (linhas);
3- As linhas que coincidem com as colunas que possuem somente um X, representam as linhas dos implicantes primos essenciais.

Na Figura 6, apresenta-se a tabela de cobertura formada a partir do conjunto de implicantes primos obtidos na primeira fase do método e que pertencem ao Grupo 1.

**Figura 6-** Tabela de cobertura do método de Quine- McCluskey

| **Implicantes primos** | **Mintermos** | | | | | |
|---|---|---|---|---|---|---|
| | 2 | 3 | 5 | 7 | 11 | 13 |
| (2,3) 001X | X | X | | | | |
| (3,7) 0X11 | | X | | X | | |
| (5,7) 01X1 | | | X | X | | |
| (3,11) X011 | | X | | | X | |
| (5,13) X101 | | | X | | | X |

Fonte: Próprio Autor

Nota-se que os mintermos 2, 11 e 13 possuem somente uma célula marcada com X. Os implicantes correspondentes representam os primos essenciais. Portanto, os termos 001X, X011 e X101 são os primeiros implicantes primos essenciais obtidos.

A tabela de cobertura é reduzida com a retirada das linhas essenciais e das colunas que os mintermos já foram cobertos, dando origem à solução mínima da função. Com as linhas e colunas eliminadas obtém-se os implicantes primos, como pode-se observar na Figura 7.

**Figura 7** – Implicantes primos obtidos através do método de Quine-McCluskey

| **Implicantes primos** | **Mintermos** | | |
|---|---|---|---|
| | 3 | 5 | 7 |
| (3,7) 0X11 | X | | X |
| (5,7) 01X1 | | X | X |

Fonte: Próprio Autor

Pode-se notar que falta cobrir apenas o mintermo 7. A cobertura pode ser realizada através dos implicantes primos 0X11 e 01X1, pois qualquer uma das alternativas resulta em um conjunto mínimo.

Com tal característica, podem-se obter duas expressões mínimas para a função:

$$F(A, B, C, D) = 001X + X011 + X101 + 0X11 = \overline{A}\,\overline{B}C + \overline{B}CD + B\overline{C}D + \overline{A}CD$$

ou

$$F(A, B, C, D) = 001X + X011 + X101 + 01X1 = \overline{A}\,\overline{B}C + \overline{B}CD + B\overline{C}D + \overline{A}BD$$

Apresenta-se na próxima seção um método de geração de implicantes primos, desenvolvido por Sheinman, que utiliza-se do Teorema de Expansão de Shannon.

É importante salientar que muitos dos métodos de minimização de funções booleanas descritos na literatura foram baseados no método de Quine-McCluskey. A grande deficiência deste método é a fase de geração dos implicantes primos, pois o tempo de execução e o consumo de memória crescem exponencialmente como o aumento da quantidade de variáveis. Por isso a busca incessante por métodos eficiente para a geração de implicantes primos.

## 4.2 TEOREMA DE EXPANSÃO DE SHANNON

O método de Sheinman (1962) é uma técnica simples e iterativa para determinar os implicantes primos de uma função booleana.

O Teorema de Expansão desenvolvido por Shannon permite expandir qualquer função booleana sobre qualquer uma de suas variáveis. Quando uma função é expandida sobre uma variável x, por exemplo, duas novas funções são formadas e chamadas de funções resíduo de x (SHANNON, 1948).

Considerando uma função booleana qualquer $F(x_1, x_2 ..... x_n)$, é expandida por meio do Teorema de Expansão de Shannon, formam-se outras duas funções que são resíduo de $x_1$ por exemplo:

$$F(x_1, x_2 ..... x_n) = x_1 . F(1, x_2 ..... x_n) + \overline{x_1} . F(0, x_2 ..... x_n).$$

São atribuídos às variáveis pesos binários, na ordem mostrada, como sendo $x_1$ a de maior peso e $x_n$ a de menor. Cada posição da variável possui um peso atribuído, a variável que está mais a esquerda é a mais significativa e a variável mais a direita é a menos significativa como, por exemplo, para uma função de 3 variáveis $x_1$, $x_2$ e $x_3$ têm-se $x_1 = 2^2$, $x_2 = 2^1$ e $x_3 = 2^0$.

A partir da escolha da variável de maior peso, os resíduos podem ser formados como descritos a seguir:

$$F(x_1, x_2 ..... x_n) = \sum mj = x_1 R_1 + \overline{x_1} R_2$$

Onde mj representa um termo produto qualquer, e o índice j é o decimal equivalente ao termo produto associado.

$R_2$ corresponde a soma dos decimais que são menores do que o peso assumido pela variável a ser expandida $x_1$ e $R_1$ a soma dos decimais que são maiores do que o peso de $x_1$, sendo que cada um desses números é subtraído o peso de $x_1$.

Como exemplo considere à função: $F(A, B, C, D) = \sum(0, 1, 8, 9, 1O) = AR_1 + \overline{A} R_2$, em que a ordem das variáveis (A, B, C, D) indica os pesos binários relativos, cuja variável A é de maior peso e a expansão inicia-se a partir da variável, sendo assim obtém-se:

$$R_2 = f(B, C, D) = \sum m\ (0, 1)\quad e$$

$$R_1 = f(B, C, D) = \sum m(8-8), (9-8), (10-8) =$$

$$R_1 = \sum m(0, 1, 2)$$

Cada um dos resíduos pode agora, ser expandido em termos da variável B, que corresponde a variável de maior peso do resíduo. Desse modo, podem-se expandir todas as variáveis da função, conforme a expressão:

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

### 4.2.1 Geração de implicantes primos através do Teorema de Expansão de Shannon

O método de geração de implicantes primos através do Teorema de Expansão de Shannon foi proposto por Scheinman em 1962, utilizado para minimizar funções com grande número de variáveis e *don't care states* (SCHEINMAN, 1962). Conforme já mencionada, o Teorema de Expansão de Shannon permite que qualquer função booleana seja expandida sobre uma de suas variáveis.

A geração de implicantes primos será ilustrada, utilizando-se a função F(A, B, C) = $\sum m\ (1,5,7)$ e consiste nos seguintes passos:

1 - Ordene os mintermos em notação decimal, que representam a função. Disponha-os em forma de coluna, como ilustrado na Figura 8. Se a função contiver *don't care states* os mesmos devem ser tratados como se fossem mintermos.

Figura 8 – Passo 1 do algoritmo

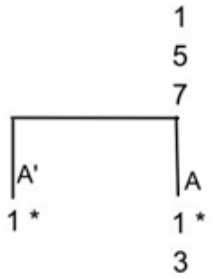
1
5
7

Fonte: Próprio Autor

2 – Selecione a variável de maior peso e divida os decimais da coluna inicial em outras duas colunas. Os decimas comuns às colunas A e A’ devem ser marcados nas respectivas colunas para indicar que são termos redundantes em relação à variável A. Se alguma coluna conter somente decimais marcados, toda a coluna deve ser eliminada.

Como A é a variável de maior peso, uma coluna é denominada de A e a outra de A’. A coluna A’ deve conter os decimais da função que são menores que 4 (peso da variável binária A). A outra coluna rotulada por A deve conter os decimais maiores ou iguais a 4, primeiramente subtraindo 4 de cada decimal maior que 4. Dessa forma, têm-se 5 – 4 = 1 e 7 – 4 = 3, cujo valor 4 correspondente a variável A, como apresenta-se na Figura 9.

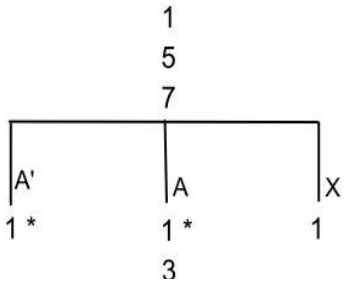
Figura 9 – Passo 2 do algoritmo




Fonte: Próprio Autor

3 – Insira uma terceira coluna, rotulada por X, consistindo dos decimais comuns às duas colunas A e A’. Como pode-se perceber na Figura 10, o decimal 1 é redundante e portanto deve estar incluindo na coluna X.

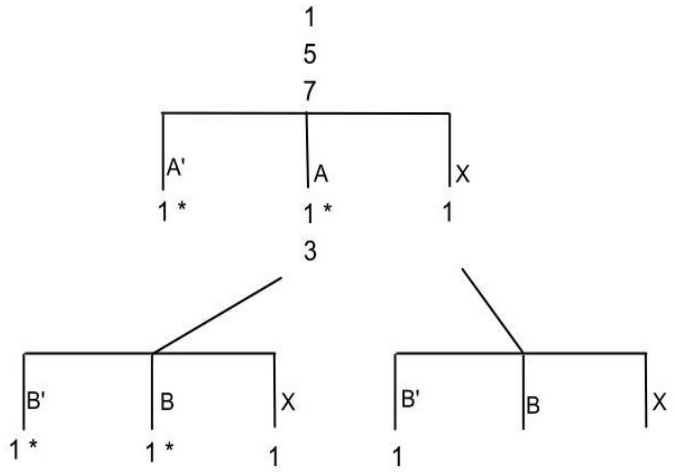
Figura 10 – Passo 3 do algoritmo




Fonte: Próprio Autor

4 – Analise cada coluna, se alguma contiver todos os decimais marcados a mesma deve ser excluída, e as outras colunas devem ser expandidas sobre a variável B. Assim, restam as colunas A e X para ser expandida sobre a variável B, como ilustrado na Figura 11.

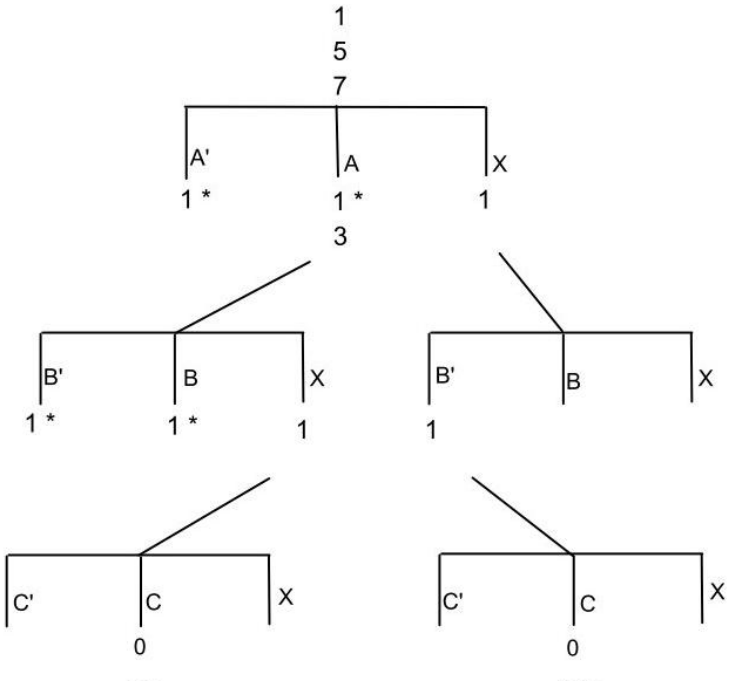
Figura 11 – Passo 4 do algoritmo




Fonte: Próprio Autor

Resta somente a variável C para ser expandida, a de menor peso, que no algoritmo são consideradas como folhas, em que os resíduos devem ser sempre 0.

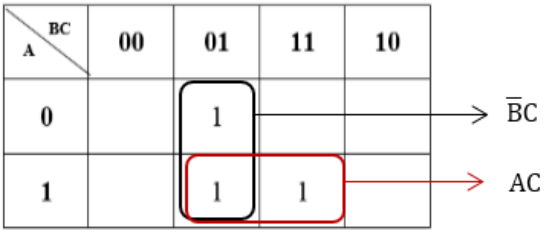
Figura 12 – Último passo do algoritmo




Fonte: Próprio Autor

Como é visto na Figura 12 os implicantes primos são obtidos traçando o caminho que começa nas folhas (variável de menor peso) até a raiz (variável de maior peso). A variável que corresponde às colunas contidas no caminho representam os implicantes primos. Portanto os implicantes primos gerados pelo método de Expansão de Shannon são: $AC + \overline{B}C$, cujas coberturas no mapa de Karnaugh são ilustradas na Figura 13.

**Figura 13** – Mapas de Karnaugh representando a função $F(A, B, C) = \sum m(1,5,7)$




Fonte: Próprio Autor

Salienta-se que o objetivo do método de Sheinman é obter os implicantes primos. Para obter uma solução mínima global deve-se formular como um problema de cobertura.

No próximo capítulo apresenta-se o método de geração de implicantes primos denominado Clause-Column Table, em que fundamentou-se este trabalho de pesquisa. Este método se mostrou atraente por apresentar diversas características importantes para a geração eficiente de implicantes primos. Uma destas características é o fato de não gerar todos os implicantes primos de uma função booleana a ser simplificada, como na primeira fase do método de Quine-McCluskey.

O método de Das (1972) representa as funções booleanas em forma de tabela, onde cada coluna contém uma proposição lógica, por isto o método é denominado de Clause-Column Table.

# 5 MÉTODO CLAUSE-COLUMN TABLE

O método Clause-Column Table utiliza-se de uma técnica tabular e iterativa que permite a geração de implicantes primos de uma determinada função booleana especificada na forma canônica ou não, na forma normal Disjuntiva (soma de produto) ou na forma normal Conjuntiva (produto de soma). O resultado pode ser rapidamente obtido mesmo no caso de funções com muitos termos e variáveis.

Uma função booleana está em sua forma canônica quando todas as variáveis aparecem em cada um dos termos e podem ser especificadas na forma normal Disjuntiva ou Conjuntiva.

Considera-se que a forma Disjuntiva (mintermos) é obtida somando os produtos lógicos em que a função assume o valor 1. Na forma Conjuntiva (maxtermos) é obtida multiplicando as somas lógicas em que a função assume valor 0.

O método Clause-Column Table é representado através de tabelas. Cada coluna da tabela é composta por um maxtermo da função booleana. Pode-se notar na Tabela 6, que a coluna M1 representa o termo soma (maxtermo) $\overline{A}+B+C$, a coluna M2 o termo $A+\overline{B}+C$, e assim por diante. Cabe salientar que o método original também permite o uso de mintermos para representar as colunas da tabela.

**Tabela 6 -** Representação tabular dos maxtermos no método Clause-Column Table.

| **M1** | **M2** | **M3** | **M4** |
|---|---|---|---|
| $\overline{A}$ | A | $\overline{A}$ | $\overline{A}$ |
| B | $\overline{B}$ | B | $\overline{B}$ |
| C | C | $\overline{C}$ | C |

Fonte: Próprio Autor

## 5.1 DEFINIÇÕES DO MÉTODO CLAUSE-COLUMN TABLE

O método Clause-Column Table desenvolvido por Das (1972) apresenta várias definições que são necessárias para o entendimento do método.

Definição 1: Designa-se como literal uma variável em sua forma complementada ou não. Se A representa um literal qualquer, $\overline{A}$ representa o seu complemento.

Definição 2: Cada termo de uma função booleana na representação de soma de produto ou produto de soma é uma coluna da tabela.

Definição 3: Se alguma coluna contiver apenas um literal, este literal é definido como sendo essencial.

Definição 4: As tabelas podem possuir algumas colunas que são chamadas de colunas dominadas. Quando duas colunas possuírem alguns literais idênticos, sendo uma coluna com mais literais que a outra, e esta coluna for composta pelos mesmos literais da coluna com menos literais, então a coluna com mais literal é considerada dominada e deve ser excluída.

Definição 5: A Tabela Clause-Column Table pode ser reduzida eliminando todas as colunas que contêm literais essenciais, colunas com conjuntos idênticos de literais e colunas dominadas. Quando uma coluna ou parte dela é eliminada denomina-se a nova tabela de reduzida.

É definido também um processo de ordenação entre os literais. Este processo de ordenação nada mais é do que selecionar um literal em sua forma complementada ou não que mais aparece na tabela. Se vários literais aparecerem em mesma quantidade de colunas, pode-se escolher qualquer um deles aleatoriamente para que posteriormente seja expandido.

Definição 6*:* Se uma função booleana for representada em soma de produto deve-se obter a sua função dual.

Cabe ressaltar que se A assume ser um literal essencial, então o literal $\overline{A}$ não aparece sozinho em outra coluna, isto é $\overline{A}$ não é um literal essencial também.

Ressalta-se também que quando dois literais A e $\overline{A}$, por exemplo, tornam-se essenciais em seguida um do outro, o algoritmo deve ser interrompido.

## 5.2 DESCRIÇÃO DO ALGORITMO CLAUSE-COLUMN TABLE

O algoritmo de Das (1972) utiliza-se de um método tabular e iterativo. O método torna-se particularmente vantajoso quando as funções booleanas são dadas na forma de produto de soma. O algoritmo é composto pelos seguintes passos:

**Passo 1:** Se a função booleana estiver na representação canônica produto de soma execute o próximo passo, senão obtenha a forma canônica produto de soma. Forme a tabela com as colunas de proposições em que cada coluna representa um termo soma.

**Passo 2:**

I. Selecione os literais essenciais;
II. Exclua todas aquelas colunas da tabela que contém literais essenciais;
III. Se A é um literal essencial, elimine o $\overline{A}$;
IV. Havendo duas colunas iguais elimine uma delas;
V. Exclua as colunas dominadas, se houver;
VI. Forme uma tabela dos termos restantes chamada de Clause-Column Table reduzida;

Repita o passo 2 iterativamente até que todas as colunas sejam eliminadas ou execute o passo 3.

**Passo 3:** Encontre a frequência de ordenação dos literais, ou seja, identifique o literal de maior incidência.

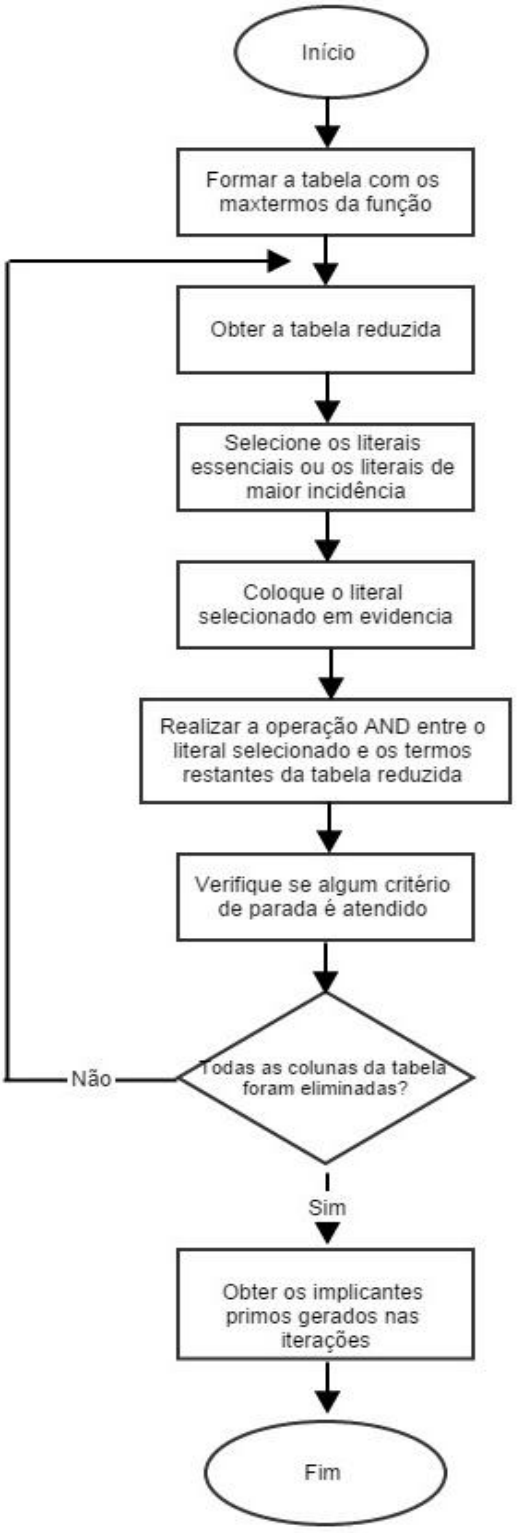
**Passo 4:** Selecione o literal de maior incidência na tabela reduzida Clause-Column Table. Se houver literais com a mesma quantidade de ocorrências selecione qualquer um deles aleatoriamente. As seguintes operações devem ser executadas:

I. Elimine todas as colunas que contiver o literal selecionado, e também elimine o complemento, se houver, a partir das colunas restantes.
II. Forme a tabela a Clause-Column Table consistindo dos literais restantes das colunas a partir do qual o literal $\overline{A}$ foi eliminado e o resto das colunas se houver.
III. Exclua o literal A a partir de todas as colunas restantes da tabela Clause-Column Table. Repita o passo 2.

**Passo 5:** Siga os passos 2 – 4 iterativamente até que todas as colunas da tabela Clause-Column Table sejam eliminadas. No final devem-se eliminar os termos redundantes e obter o conjunto de termos.

O fluxograma apresentado na Figura 14 ilustra o algoritmo Clause-Column Table descrito anteriormente.

**Figura 14** – Fluxograma representado o algoritmo do método Clause-Column Table




Fonte: Próprio Autor

Ao minimizar funções booleanas através do método Clause-Column Table constatou-se algumas deficiências como, por exemplo, a geração de termos que não são implicantes. Das rotulou estes termos de termos nulos, pois não pertencem ao conjunto de solução. Foi constatado também que o método gera algumas iterações desnecessárias, visto que ao gerar termos nulos utilizam-se mais iterações do que o mínimo necessário.

Além dos termos considerados nulos ocasionarem uma solução infactível, encontrou-se outras características que conduzem ao mesmo problema. O algoritmo em alguns casos não gera implicantes primos que seriam essenciais para cobertura dos mintermos. Tais situações serão melhores ilustradas no decorrer desta seção.

Considere a função $F_1\ (A, B, C) = \sum m\ (1, 2, 3, 4, 5, 6)$:

**1º Passo:** Se a função booleana estiver na representação canônica produto de soma execute o próximo passo, senão obtenha a forma canônica produto de soma. Monte a tabela com as colunas de proposições em que cada coluna representa um termo soma.

Salienta-se que o método torna-se significativamente mais rápido quando as funções são minimizadas a partir dos maxtermos.

Como a função $F_1$ pela soma dos mintermos obtém - se assim o dual da função, como apresentado a seguir:

$$F_1(A, B, C) = \sum m\,(1, 2, 3, 4, 5, 6) =$$

$$= F_1(A, B, C) = \prod M(0, 7)$$

Tabela 7 – Tabela inicial do método Clause-Column Table

| **M0** | **M7** |
|---|---|
| A | $\overline{A}$ |
| B | $\overline{B}$ |
| C | $\overline{C}$ |

Fonte: Próprio Autor

**2º Passo:** Analisando a Tabela 7, nota-se que nenhum um critério deste passo é atendido. Execute o próximo passo.

**3º Passo:** Todos os literais possuem a mesma incidência na tabela. Desse modo qualquer um dos literais pode ser selecionado. Seleciona-se então o literal A.

**4º Passo:** A coluna que contiver o literal selecionado deve ser excluída e o seu complemento $\overline{A}$ deve ser excluído das demais colunas, formando a tabela reduzida, como pode-se observar na Figura 15.

Realiza-se a operação AND entre o literal selecionado e as colunas da tabela reduzida para obter os implicantes primos.

Figura 15- Tabela reduzida do método Clause-Column Table

| | | **M7** |
|---|---|---|
| A | **and** | $\overline{B}$ |
| | | $\overline{C}$ |

Fonte: Próprio Autor

Depois de realizar a operação AND geraram-se os implicantes primos: $A\overline{B}$ + A $\overline{C}$.

**2º Iteração:**

**2º Passo:** Como o literal A foi selecionado na iteração anterior ele é excluído permanentemente, como visualiza-se na Tabela 8.

Tabela 8 - Tabela referente à 2ª iteração do método Clause-Column Table

| **M0** | **M7** |
|---|---|
| B | $\overline{A}$ |
| C | $\overline{B}$ |
| | $\overline{C}$ |

Fonte: Próprio Autor

**3º Passo:** Todos os literais possuem a mesma incidência então, qualquer um deles pode ser selecionado. Neste caso o literal B é selecionado.

**4º Passo:** Todas as colunas que contiver o literal B devem ser excluídas e das demais colunas o seu complemento $\overline{B}$, formando a tabela reduzida.

Realiza-se novamente a operação AND entre o literal selecionado e a tabela reduzida para obter os implicantes primos. Esta operação é apresentada na Figura 16.

Figura 16- Tabela reduzida do método Clause-Column Table

| | | **M7** |
|---|---|---|
| B | **and** | $\overline{A}$ |
| | | $\overline{C}$ |

Fonte: Próprio Autor

Como resultado da operação AND entre a variável selecionada e a coluna restante geraram-se os implicantes primos: $\overline{A}B + B\overline{C}$

**3ª Iteração:**

**2º Passo:** Como o literal B foi selecionado na iteração anterior ele é excluído permanentemente.

A Tabela 9 contém apenas um literal em uma das colunas. Este literal é considerado essencial e deve ser selecionado.

Tabela 9 - Tabela referente à 3ª iteração do método Clause-Column Table

| **M0** | **M7** |
|---|---|
| C | $\overline{A}$ |
| | $\overline{B}$ |
| | $\overline{C}$ |

Fonte: Próprio Autor

**3º Passo:** Não é necessário encontrar o literal de maior incidência, pois existe literal essencial na tabela.

**4º Passo:** Todas as colunas que contiver o literal essencial devem ser excluídas e das demais colunas o seu complemento $\overline{C}$, formando a tabela reduzida apresentada na Figura 17.

Realiza-se novamente a operação AND entre o literal essencial selecionado e a tabela reduzida para obter os implicantes primos.

**Figura 17** - Tabela reduzida do método Clause-Column Table

| | | M7 |
|---|---|---|
| C | **and** | $\overline{A}$ |
| | | $\overline{B}$ |

Fonte: Próprio Autor

Como resultado da operação AND entre o literal selecionado e a coluna restante da tabela reduzida geraram-se os implicantes primos: $\overline{A}C + \overline{B}C$.

**4ª Iteração:**

**2º Passo:** O literal essencial selecionado na iteração anterior é excluído permanente.

Na Tabela 10 não existe literal essencial, assim o 3º passo deve ser executado.

Tabela 10 - Tabela referente à 4º iteração do método Clause-Column Table

| M7 |
|---|
| $\overline{A}$ |
| $\overline{B}$ |
| $\overline{C}$ |

Fonte: Próprio Autor

**3º Passo:** Todos os literais possuem a mesma incidência então, qualquer um pode ser selecionado. Neste caso o literal $\overline{A}$ é selecionado.

**4º Passo:** A coluna que contiver o literal $\overline{A}$ deve ser excluída, assim não resta nenhuma coluna e o $\overline{A}$ é o próprio implicante primo.

**5ª Iteração:**

**2º Passo:** O literal essencial selecionado na iteração anterior é excluído permanentemente.

Na Tabela 11 não existe literal essencial, assim o 3º passo deve ser executado.

Tabela 11 - Tabela referente à 5ª iteração do método Clause-Column Table

| **M7** |
|---|
| $\overline{B}$ |
| $\overline{C}$ |

Fonte: Próprio Autor

**3º Passo:** Todos os literais possuem a mesma incidência então, qualquer um deles pode ser selecionado. Neste caso seleciona-se o literal $\overline{B}$ é selecionado.

**4º Passo:** A coluna que contiver o literal $\overline{B}$ deve ser excluída, assim não resta nenhuma coluna e o $\overline{B}$ é o próprio implicante primo da iteração atual.

**6ª Iteração:**

**2º Passo:** O literal essencial selecionado na iteração anterior é excluído permanentemente. Como a tabela atual não possui literal essencial, assim o 3º passo deve ser executado.

**Tabela 12** - Tabela referente à 6ª iteração do método Clause-Column Table

| **M7** |
|---|
| $\overline{C}$ |

Fonte: Próprio Autor

**3º Passo:** Como pode-se observar na Tabela 12 só resta um literal e portanto deve ser selecionado.

**4º Passo:** A coluna que contiver o literal $\overline{C}$ deve ser excluída, assim não resta nenhuma coluna e o $\overline{C}$ é o próprio implicante primo.

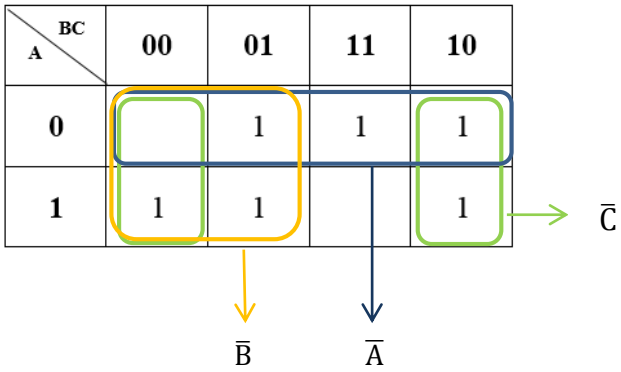
**7ªIteração:**

**2º Passo:** Como todas as colunas foram excluídas finaliza-se a execução do algoritmo.

**5º Passo:** Após a execução do algoritmo obteve-se como implicantes primos o seguinte conjunto de termos: $A\overline{B} + A\overline{C} + \overline{A}B + B\overline{C} + \overline{A}C + \overline{B}C + \overline{A} + \overline{B} + \overline{C}$.

Constata-se com a minimização da função $F_1(A, B, C) = \sum m\ (1, 2, 3, 4, 5, 6)$, que o método Clause-Column Table gerou o conjunto: $A\overline{B} + A\overline{C} + \overline{A}B + B\overline{C} + \overline{A}C + \overline{B}C + \overline{A} + \overline{B} + \overline{C}$, considerados implicantes primos. Porém, os termos $\overline{A} + \overline{B} + \overline{C}$ não são implicantes primos e, portanto são termos nulos, o que tornam a solução infactível. Pode-se notar esta infactibilidade através do mapa de Karnaugh apresentado na Figura 18.

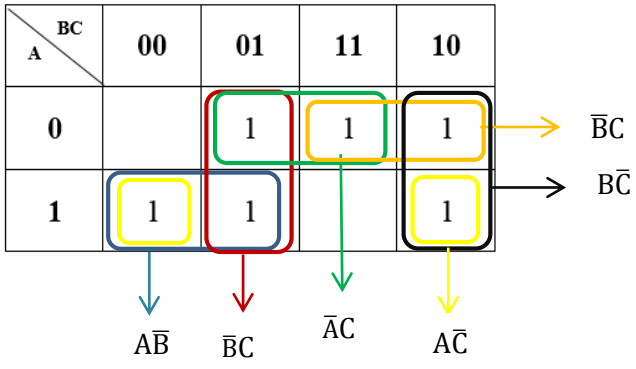
**Figura 18** – Mapa de Karnaugh com os termos nulos gerados.




Fonte: Próprio Autor

O termo $\overline{A}\ \overline{B}\ \overline{C}$ não representa um mintermo, ou seja, não poderia estar contido nos implicantes primos $\overline{A}$, $\overline{B}$ e $\overline{C}$. Somente os implicantes primos que são apresentados no mapa de Karnaugh da Figura 19 deveriam ser gerados.

**Figura 19**- Mapa de Karnaugh com os implicantes primos gerados




Fonte: Próprio Autor

Além do problema da geração dos termos nulos identificou-se mais outras duas deficiências do método de Das (1972). A primeira relacionada às funções em que os implicantes primos não são gerados, acarretando em mintermos sem cobertura e a segunda relacionada ao

critério de aleatoriedade incluso no algoritmo de Das. Neste critério, literais que possuem a mesma incidência na tabela podem ser selecionados aleatoriamente. Na maioria das funções booleanas esta aleatoriedade gera corretamente os implicantes primos. Contudo, para algumas funções analisadas o critério de aleatoriedade gerou resultados incoerentes. Estas situações descritas são apresentadas a seguir. Considere a função $F_2(A, B, C, D) = \sum m\ (0, 5, 6)$.

**1º Passo:** Forme uma tabela em que cada coluna é composta por um maxtermo da função (maxtermo/mintermo). Se a função booleana inicial estiver na forma de produto de soma, execute o próximo passo, senão obtenha o seu dual.

$$F_2(A, B, C) = \sum m\ (0, 5, 6)$$

A forma dual da função é representada por:

$$F_2(A, B, C) = \prod M(1, 2, 3, 4, 7)$$

**Tabela 13**- Tabela inicial formada pelos maxtermos

| **M1** | **M2** | **M3** | **M4** | **M7** |
|---|---|---|---|---|
| A | A | A | $\overline{A}$ | $\overline{A}$ |
| B | $\overline{B}$ | $\overline{B}$ | B | $\overline{B}$ |
| $\overline{C}$ | C | $\overline{C}$ | C | $\overline{C}$ |

Fonte: Próprio Autor

**2º Passo:** Analisando a Tabela 13 nota-se que nenhum critério deste passo é atendido. Execute o próximo passo.

**3º Passo:** Os literais A, $\overline{B}$ e $\overline{C}$ possuem a mesma incidência na tabela. Deste modo qualquer um dos literais pode ser selecionado. Seleciona-se aleatoriamente o literal A.

**4º Passo:** A coluna que contiver o literal selecionado deve ser excluída, e o seu complemento $\overline{A}$ deve ser excluído das demais colunas. Assim a tabela reduzida é formada, como é apresentado na Tabela 14.

**Tabela 14** – Tabela reduzida referente a 1ª iteração da função $F_2$.

| **M4** | **M7** |
|---|---|
| B | $\overline{B}$ |
| C | $\overline{C}$ |

Fonte: Próprio Autor

Pode-se notar que a tabela reduzida possui mais de uma coluna, portanto deve-se selecionar o literal que mais aparece novamente. Neste caso, seleciona-se o literal B, formando outra tabela reduzida apresentada na Figura 20.

Aplica-se a operação AND entre o literal selecionado e a tabela reduzida para obter os implicantes primos.

**Figura 20** – Tabela reduzida para gerar os implicantes primos

| AB **and** | **M7** |
|---|---|
| | $\overline{C}$ |

Fonte: Próprio Autor

Depois de realizar a operação AND gerou-se o implicante primo: $AB\overline{C}$, e a 1ª iteração é finalizada. Posteriormente, deve-se retornar ao 2º Passo para prosseguir a minimização.

**2ª Iteração:**

**2º Passo:** Como o literal A foi o primeiro a ser selecionado na iteração anterior ele é excluído permanentemente, como pode-se observar na Tabela 15.

A coluna M3 está contida na M7 e, portanto deve ser excluída, formando uma tabela reduzida.

**Tabela 15** – Primeira tabela da 2ª iteração

| **M1** | **M2** | **M3** | **M4** | **M7** |
|---|---|---|---|---|
| | | | $\overline{A}$ | $\overline{A}$ |
| B | $\overline{B}$ | $\overline{B}$ | B | $\overline{B}$ |
| $\overline{C}$ | C | $\overline{C}$ | C | $\overline{C}$ |

Fonte: Próprio Autor

**3º Passo:** O literal B é selecionado.

**4º Passo:** Todas as colunas que contiver o literal B devem ser excluídas e das demais colunas o seu complemento $\overline{B}$, formando a tabela reduzida.

Visto que, a tabela possui mais de uma coluna deve-se selecionar o literal que mais aparece novamente. Neste caso, seleciona-se o literal C, formando outra tabela reduzida, apresentada na Figura 21.

**Figura 21** – Tabela Reduzida para gerar os implicantes primos da 2ª iteração.

B **and**

| M1 |
|---|
| C |

Fonte: Próprio Autor

Aplica-se a operação AND entre o literal selecionado e a última tabela reduzida para obter os implicantes primos. Posteriormente, gerou-se o implicante primo: BC.

**3ª Iteração:**

**2º Passo:** Como o literal B foi selecionado na iteração anterior ele é excluído permanentemente.

A coluna M1 da Tabela está contida na coluna M3 e, portanto a coluna M3 deve ser excluída.

**Tabela 16** – Tabela da 3ª iteração da função $F_2$

| M1 | M2 | M4 |
|---|---|---|
| | | $\overline{A}$ |
| | $\overline{B}$ | B |
| $\overline{C}$ | C | C |

Fonte: Próprio Autor

Após eliminar a coluna contida pode-se perceber através da Tabela 16 que existe somente um literal em uma das colunas. Este literal é considerado essencial e deve ser selecionado.

**3º Passo:** Não é necessário encontrar o literal de maior incidência, pois existe literal essencial na tabela.

**4º Passo:** Todas as colunas que contiver o literal essencial devem ser excluídas e das demais colunas o seu complemento C, formando a tabela reduzida, que ainda possui mais de uma coluna e, portanto deve-se selecionar o literal que mais aparece novamente. Neste caso, seleciona-se o literal $\overline{A}$ formando outra tabela reduzida.

Realiza-se a operação lógica AND entre os literais selecionados e a tabela reduzida apresentada na Figura 22 para obter os implicantes primos.

**Figura 22** – Tabela reduzida utilizada para realizar a operação AND

$\overline{A}\overline{C}$ **and**

| **M1** |
|---|
| $\overline{B}$ |

Fonte: Próprio Autor

Como resultado da operação AND foram gerados os implicantes primos $\overline{A}\,\overline{B}\,\overline{C}$.

**4ª Iteração:**

**2º Passo:** O primeiro literal selecionado na iteração anterior é excluído permanentemente.

A Tabela 17 que é a tabela inicial da 4ª iteração não possui literal essencial, assim o 3º passo deve ser executado.

**Tabela 17** – Tabela da 4ª Iteração da função $F_2$

| **M1** | **M4** |
|---|---|
| $\overline{B}$ | $\overline{A}$ |
| C | C |

Fonte: Próprio Autor

**3º Passo:** O literal C possui a maior incidência, portanto é selecionado.

**4º Passo:** Como o literal selecionado está contido em todas as colunas, as mesmas são excluídas. Sendo assim, o literal C é o próprio implicante primo da iteração atual.

**5ª Iteração:**

**2º Passo:** O literal essencial selecionado na iteração anterior é excluído permanentemente.

A Tabela 18 não possui literal essencial, assim o 3º passo deve ser executado.

**Tabela 18** – Tabela reduzida referente a 5ª iteração da função $F_2$

| **M1** | **M4** |
|---|---|
| $\overline{B}$ | $\overline{A}$ |

Fonte: Próprio Autor

**3º Passo:** Todos os literais possuem a mesma incidência então, qualquer um deles pode ser selecionado aleatoriamente. Neste caso o literal $\overline{B}$ é selecionado.

**4º Passo:** Como pode-se observar através da Figura 23 a coluna que continha o literal $\overline{B}$ foi excluída.

**Figura 23** – Tabela Reduzida para gerar os implicantes primos da 5ª iteração

$\overline{B}$ **and**

| **M4** |
|---|
| $\overline{A}$ |

Fonte: Próprio Autor

Realiza-se a operação AND entre o literal selecionado e a tabela reduzida para obter os implicantes primos, que são: $\overline{A}\ \overline{B}$.

**6ª Iteração:**

**2º Passo:** O literal essencial selecionado na iteração anterior é excluído permanentemente. A tabela atual possui literal essencial e deve ser selecionado.

**Tabela 19** – Tabela inicial da 6ª iteração referente à função $F_2$

| M7 |
|---|
| $\overline{A}$ |

Fonte: Próprio Autor

**3º Passo:** Existe literal essencial na Tabela 19, então execute o 4º Passo.

**4º Passo:** Como só existe um literal, o mesmo é considerado implicante primo da iteração atual.

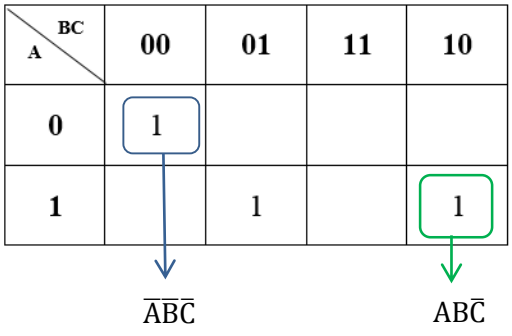
**7ª Iteração:**

**2º Passo:** Como todas as colunas foram excluídas, a execução do algoritmo finaliza-se.

**5º Passo:** Após a execução do algoritmo obteve-se como implicantes primos o conjunto de termos: $AB\overline{C} + BC + \overline{A}B\overline{C} + C + \overline{A}\overline{B}$.

A Figura 24 ilustra o mapa de Karnaugh para a função $F_2(A, B, C) = \sum m\,(0, 5, 6)$. Nota-se que o implicante primo $A\overline{B}C$, não foi gerado através do método Clause-Column Table. Esta situação representa a deficiência do algoritmo descrita anteriormente, pois um implicante primo considerado essencial não foi gerado, originando uma solução infactível.

**Figura 24** – Mapa de Karnaugh da função $F_2(A, B, C) = \sum m\,(0, 5, 6)$



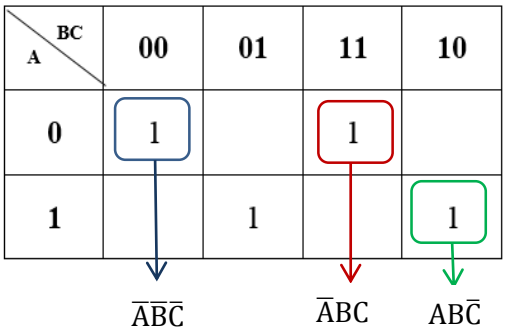

Fonte: Próprio Autor

Semelhante à situação apresentada, o método apresenta outra falha em relação a geração dos implicantes. Considerando a função $F_3\,(A, B, C, D) = \prod M(1, 2, 4, 7)$ foram gerados os implicantes primos: $AB\overline{C} + \overline{A}BC + \overline{A}B\overline{C} + \overline{A}\overline{B} + \overline{A}C$. Observa-se no mapa de Karnaugh apresentado na Figura 25, que um dos implicantes primos também não foi gerado e, portanto o critério de aleatoriedade adotado por Das não é válido, pois é baseado no principio de seleção

aleatória dos literais que ocorrem o mesmo número de vezes na tabela, visto que o literal selecionado é utilizado para a geração do implicante primo da iteração atual.

Simplificando a função $F_3$, obtém-se uma tabela reduzida cujos literais possuem a mesma ocorrência. Quando o literal B é selecionado, gera-se o implicante primo $A\bar{B}\bar{C}$ e quando seleciona-se o $\bar{B}$ é gerado o implicante primo $A\bar{B}C$, logo o critério de aleatoriedade é falho, pois dependendo do literal selecionado é gerado um implicante diferente. Neste caso, a seleção resultou num literal sem cobertura.

**Figura 25** – Mapa de Karnaugh da $F_3\,(A, B, C, D) = \prod M(1, 2, 4, 7)$




Fonte: Próprio Autor

Diante das deficiências apresentadas pelo método Clause-Column Table, procurou-se aprimorá-lo a fim de gerar implicantes primos com eficiência. Origina-se então, uma nova versão do método de Das denominado de Clause-Column Table Aprimorado, que é apresentado no próximo capítulo.

# 6 MÉTODO CLAUSE-COLUMN TABLE APRIMORADO

Com a finalidade de sanar algumas deficiências encontradas no método Clause-Column Table, utilizado para a geração de implicantes primos, desenvolveu-se uma nova versão denominada de Clause-Column Table Aprimorado. Neste novo algoritmo foram feitas algumas modificações necessárias para evitar a geração de termos nulos e diminuir a quantidade de iterações utilizadas na geração dos implicantes primos.

Para reduzir o número de iterações adicionou-se a propriedade algébrica $A\,B + A\,\overline{B} = A$ pertencente à álgebra booleana. Ao aplicá-la nas colunas referentes à tabela inicial formada pelos maxtermos da função booleana vários termos são eliminados.

O método original gera na fase inicial a tabela ilustrada na Figura 26, em que as colunas rotuladas por $M_1$, $M_2$, $M_3$ e $M_4$ representam maxtermos da função. Nota-se que as colunas $M_1$ e $M_4$ podem ser simplificadas algebricamente de acordo com o teorema da adjacência. O mesmo ocorre para as colunas $M_2$ e $M_3$. O método original não realiza tal simplificação.

**Figura 26**- Simplificação algébrica aplicada nas colunas da tabela inicial do algoritmo

| M1 | M2 | M3 | M4 |
|---|---|---|---|
| $\overline{A}$ | A | A | $\overline{A}$ |
| $\overline{B}$ | $\overline{B}$ | B | $\overline{B}$ |
| $\overline{C}$ | C | C | C |

$\longrightarrow$

| M1 | M2 |
|---|---|
| $\overline{A}$ | A |
| $\overline{B}$ | C |

Fonte: Próprio Autor

Além do emprego do teorema da adjacência incluiu-se no algoritmo original um novo critério de parada com a finalidade de evitar a geração de termos nulos. Se restar apenas uma coluna a partir da segunda iteração o algoritmo deve ser interrompido.

## 6.1 DESCRIÇÃO DO ALGORITMO CLAUSE-COLUMN TABLE APRIMORADO

O algoritmo Clause-Column Table Aprimorado é composto pelos seguintes passos:

**1º Passo:** Forme uma tabela em que cada coluna é composta por um maxtermo da função. Se a função booleana inicial estiver na forma de produto de soma, execute o próximo passo, senão obtenha o seu dual.

**2º Passo:** Execute iterativamente até que todas as colunas sejam excluídas, caso contrário o 3º passo deve ser executado:

a) Se algum critério de parada do 6º passo é atendido, interrompa o algoritmo, caso contrário continue;
b) Se houver colunas dominadas, elimine-as;
c) Se existir colunas redundantes elimine aleatoriamente uma delas;
d) Simplifique as colunas restantes de acordo com o teorema de adjacência (apenas uma vez em cada iteração).
e) Se houver literal essencial, selecione-o. Se houver mais de um literal essencial selecione um deles aleatoriamente;
f) Verifique novamente critério de parada 6 –c;
g) Elimine todas as colunas que contém literais essenciais;
h) Elimine o complemento do literal essencial selecionado das colunas restantes;

**3º Passo:** Se houver um literal essencial selecionado no passo anterior execute o 5º passo, caso contrário execute o 4º passo.

**4º Passo:** Verifique na tabela reduzida a quantidade de incidência de cada um dos literais na forma complementada ou não.

a) Se houver literais com a mesma incidência selecione um deles aleatoriamente;
b) Se o literal com a maior incidência estiver contido em todas as colunas ele deve ser considerado implicante primo da iteração atual;

c) Execute o 6º passo.

**5º Passo**:

a) Todas as colunas que contiver o literal selecionado devem ser eliminadas;
b) O complemento do literal selecionado também deve ser eliminado das demais colunas;
c) Forme a tabela reduzida com o restante das colunas;
d) Se na tabela reduzida houver mais de uma coluna selecione o literal essencial, senão escolha aleatoriamente o de maior incidência e repita os passos 5-a) e 5-b).
e) Realize a operação lógica AND entre o literal selecionado e a tabela reduzida, cujos termos resultantes são implicantes primos.

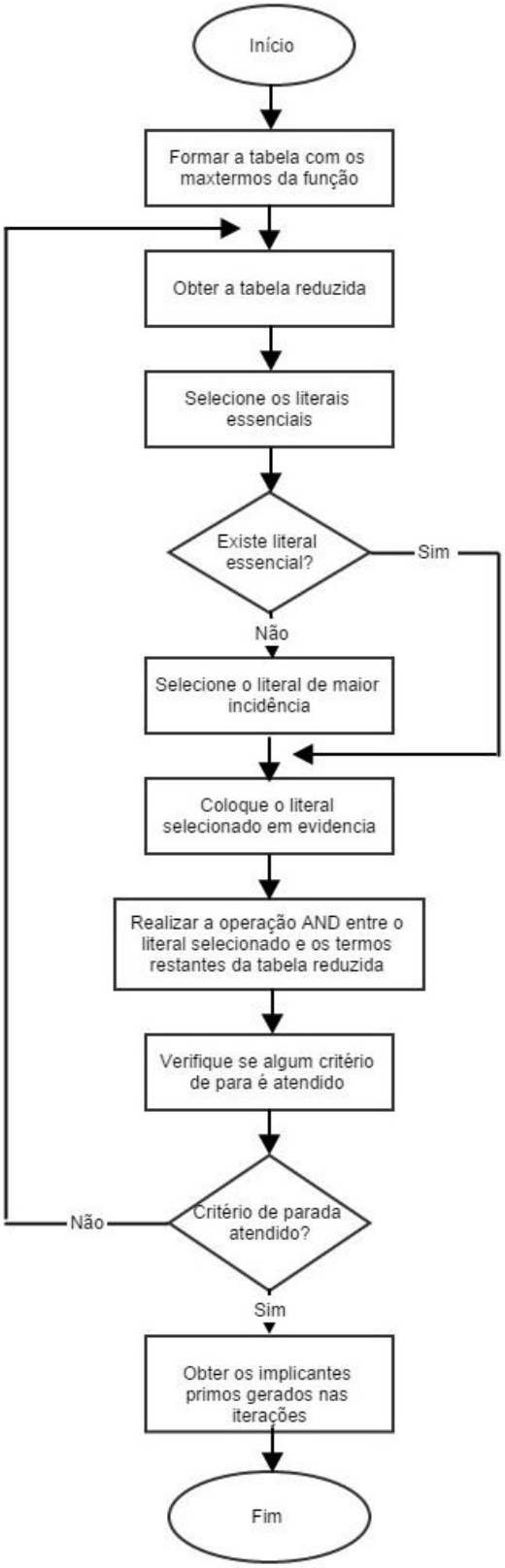
**6º Passo:** Verifique se algum critério de parada é atendido.

a) Todas as colunas sejam excluídas;
b) Dois literais, complementados ou não, tornem-se essenciais ao mesmo tempo;
c) Um literal torne-se essencial e na iteração seguinte o seu complemento também;
d) Reste apenas uma coluna a partir da segunda iteração.

**7º Passo:** Uma vez atingido algum critério de parada reúna os implicantes primos de cada iteração, eliminando os termos redundantes e forme o conjunto de solução.

A Figura 27 apresenta o fluxograma do algoritmo do método Clause-Column Table Aprimorado.

**Figura 27** – Fluxograma apresentando o algoritmo Clause-Column Table Aprimorado




**Fonte:** Próprio Autor

O algoritmo implementado será ilustrado, para maior clareza, utilizando-se como exemplo a função de 4 variáveis, $F_4(A, B, C, D) = \prod M(0, 1, 2, 3, 8, 9, 10, 11)$.

**1º Passo:** Forme uma tabela em que cada coluna é composta por um maxtermo. Se a função booleana inicial estiver na forma de produto de soma, execute o próximo passo, senão obtenha o seu dual.

$$F_4\ (A, B, C, D) = \prod M(0, 1, 2, 3, 8, 9, 10, 11)$$

$$F_4(A, B, C, D) = (A + B + C + D).(A + B + C + \overline{D}).(A + B + \overline{C} + D).(A + B + \overline{C} + \overline{D}).$$
$$(\overline{A} + B + C + D).(\overline{A} + B + C + \overline{D}).(\overline{A} + B + \overline{C} + D).(\overline{A} + B + \overline{C} + \overline{D})$$

**Tabela 20** - Tabela inicial do método Clause-Column Table Aprimorado

| **M0** | **M1** | **M2** | **M3** | **M8** | **M9** | **M10** | **M11** |
|---|---|---|---|---|---|---|---|
| A | A | A | A | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ |
| B | B | B | B | B | B | B | B |
| C | C | $\overline{C}$ | $\overline{C}$ | C | C | $\overline{C}$ | $\overline{C}$ |
| D | $\overline{D}$ | D | $\overline{D}$ | D | $\overline{D}$ | D | $\overline{D}$ |

**Fonte**: Próprio Autor

**2º Passo:** Analisando a Tabela 20, nota-se que o passo 2- d (Simplifique as colunas restantes de acordo com o teorema de adjacência) deve ser executado. O teorema de adjacência aplica-se nas colunas M0 e M1, M2 e M3, M8 e M9, M10 e M11. Em seguida tem-se a Tabela 21 que foi reduzida formando as colunas M1, M3, M9 e M10.

**Tabela 21** - Tabela do método Clause-Column Table Aprimorado após a simplificação algébrica

| **M1** | **M3** | **M9** | **M11** |
|---|---|---|---|
| A | A | $\overline{A}$ | $\overline{A}$ |
| B | B | B | B |
| C | $\overline{C}$ | C | $\overline{C}$ |

**Fonte:** Próprio Autor

**3º Passo:** No passo anterior não houve nenhum literal essencial selecionado. Execute o 4º passo.

**4º Passo:** Na Tabela 22 pode-se verificar que o literal B é o que mais ocorre e está contido em todas as colunas da tabela, portanto é considerado o implicante primo da iteração atual.

**Tabela 22** - Tabela referente à 1ª iteração do método Clause-Column Table Aprimorado

| M1 | M3 | M9 | M11 |
|---|---|---|---|
| A | A | $\overline{A}$ | $\overline{A}$ |
| B | B | B | B |
| C | $\overline{C}$ | C | $\overline{C}$ |

**Fonte:** Próprio Autor

**6º Passo:** Nenhum critério de parada é atendido. Execute o 2º passo novamente.

**2ª Iteração:**

**2º Passo:** Como o literal B foi o primeiro selecionado na iteração anterior ele é excluído permanentemente, como pode-se observar na Tabela 23.

**Tabela 23** - Tabela referente à 2ª iteração do método Clause-Column Table Aprimorado

| M1 | M3 | M9 | M11 |
|---|---|---|---|
| A | A | $\overline{A}$ | $\overline{A}$ |
| C | $\overline{C}$ | C | C |

**Fonte**: Próprio Autor

Nota-se na Tabela 23 que o teorema de adjacência aplica-se nas colunas M1 e M3, M9 e M11, formando a tabela 24 que reduziu-se, composta agora pelas colunas M3 e M11. Posteriormente o critério de parada 6-b é atendido (Um literal e seu complemento tornem-se essenciais ao mesmo tempo).

**Tabela 24** - Tabela após a simplificação algébrica

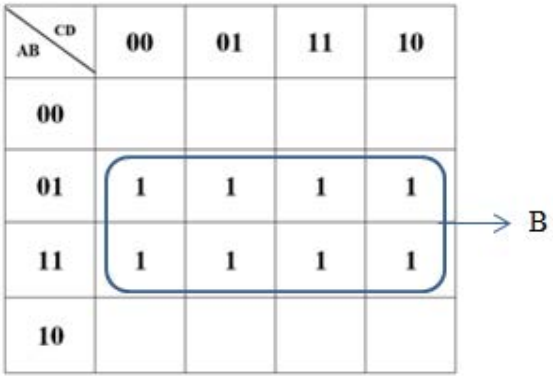
| M3 | M11 |
|---|---|
| A | $\overline{A}$ |

**Fonte:** Próprio Autor

**7º Passo:** Uma vez atingido o critério de parada, deve-se reunir todos os termos obtidos no 5º passo, pois representam implicantes primos.

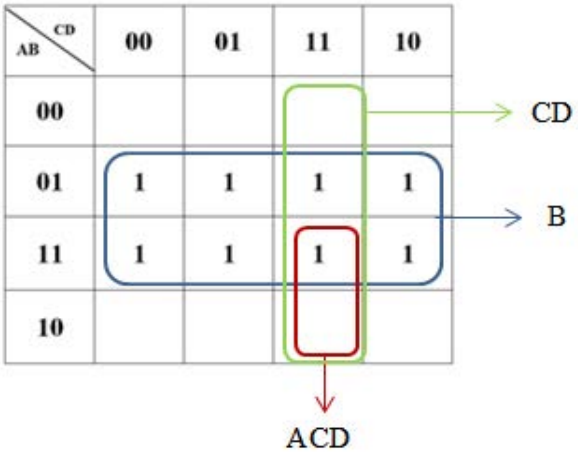
Diante do apresentado nota-se através da Figura 28 que o método Clause-Column Table Aprimorado gerou somente o implicante primo essencial B, ao contrário do método original que minimizando a mesma função gerou o conjunto de implicantes primos B+ACD+AC, cujos termos ACD + AC são nulos e, portanto invalidam o resultado, como é ilustrado na Figura 29. Constata-se então que a simplificação algébrica, além do novo critério de parada, evitou a geração de termos nulos e reduziu o número de iterações.

**Figura 28** – Cobertura dos mintermos gerada pelo método Clause-Column Table Aprimorado




**Fonte:** Próprio Autor

**Figura 29** – Cobertura dos mintermos gerada pelo método Clause-Column Table




**Fonte:** Próprio Autor

Utiliza-se de outro exemplo para ilustrar como o critério de parada, presente somente no algoritmo Clause-Column Table Aperfeiçoado, evita a geração dos termos nulos.

O algoritmo será ilustrado, utilizando-se da função $F_5(A, B, C) = \prod M(0, 7)$, de 3 variáveis e consiste nos seguintes passos:

**1º Passo:** Forme uma tabela em que cada coluna é composta por um maxtermo. Se a função booleana estiver na forma de produto de soma, execute o próximo passo, senão obtenha o seu dual.

$$F_5(A, B, C) = \prod M(0, 7)$$

**Tabela 25** – Tabela inicia referente à função $F_5(A, B, C) = \prod M(0, 7)$

| M0 | M7 |
|---|---|
| A | $\overline{A}$ |
| B | $\overline{B}$ |
| C | $\overline{C}$ |

**Fonte:** Próprio Autor

**2º Passo:** Analisando a Tabela 25 nota-se que nenhum critério deste passo é atendido. Execute o próximo passo.

**3º Passo:** No passo anterior não houve nenhum literal essencial selecionado. Execute o 4º passo.

**4º Passo:** Todos os literais possuem a mesma incidência na tabela. Sendo assim, qualquer um dos literais pode ser selecionado. Seleciona-se então o literal A.

**5º Passo:** A coluna que contiver o literal selecionado deve ser excluída, e o seu complemento $\overline{A}$ deve ser excluído das demais colunas, formando tabela reduzida apresentada na Figura 30.

Realiza-se a operação AND entre o literal selecionado e a tabela reduzida para obter os implicantes primos.

**Figura 30** - Tabela reduzida do método Clause-Column Table Aprimorado

A **and**

| **M7** |
|---|
| $\overline{B}$ |
| $\overline{C}$ |

**Fonte:** Próprio Autor

Depois de realizar a operação AND geraram-se os implicantes primos: A $\overline{B}$ + A$\overline{C}$.

**6º Passo:** Nenhum critério de parada é atendido, assim o 2º passo deve ser executado novamente, iniciando outra iteração.

**2ª Iteração:**

**2º Passo:** Como o literal A foi selecionado na iteração anterior ele é excluído permanentemente, como pode-se observar na Figura 31.

**Figura 31**- Tabela referente à 2ª iteração do método Clause-Column Table Aprimorado

| **M0** | **M7** |
|---|---|
| B | $\overline{A}$ |
| C | $\overline{B}$ |
| | $\overline{C}$ |

**Fonte:** Próprio Autor

**3º Passo:** No passo anterior não houve nenhum literal essencial selecionado. Execute o 4º passo.

**4º Passo:** Todos os literais possuem a mesma incidência então qualquer um deles pode ser selecionado. Neste caso o literal B é selecionado.

**5º Passo:** Todas as colunas que contiver o literal B devem ser excluídas e das demais colunas o seu complemento $\overline{B}$, formando a tabela reduzida.

Realiza-se novamente a operação AND entre o literal selecionado e a tabela reduzida, apresentada na Figura 32, para obter os implicantes primos.

**Figura 32-** Tabela reduzida do método Clause-Column Table Aprimorado

| | | **M7** |
|---|---|---|
| B | **and** | $\overline{A}$ |
| | | $\overline{C}$ |

**Fonte:** Próprio Autor

Posteriormente a operação AND geraram-se os implicantes primos: A $\overline{A}$B+ B $\overline{C}$

**6º Passo**: Nenhum critério de parada é atendido, deve-se começar outra iteração.

**3ª Iteração:**

**2º Passo:** Como o literal B foi selecionado na iteração anterior ele é excluído permanentemente.

Pode-se perceber na Tabela 26 que há somente um literal em uma das colunas, que é considerado essencial e deve ser selecionado.

**Tabela 26 -** Tabela inicial referente à 3ª iteração da função $F_5$

| **M0** | **M7** |
|---|---|
| C | $\overline{A}$ |
| | $\overline{B}$ |
| | $\overline{C}$ |

**Fonte:** Próprio Autor

**3º Passo:** No passo anterior o literal essencial C foi selecionado e portando o 5º Passo deve ser executado.

**5º Passo:** Todas as colunas que contiver o literal essencial devem ser excluídas e das demais colunas o seu complemento $\overline{C}$, formando a tabela reduzida ilustrada na Figura 33.

Realiza-se novamente a operação AND entre o literal essencial selecionado e a tabela reduzida para obter os implicantes primos.

**Figura 33**- Tabela reduzida do método Clause-Column Table Aprimorado

C **and**

| **M7** |
|---|
| $\overline{A}$ |
| $\overline{B}$ |

**Fonte:** Próprio Autor

Posteriormente a operação AND geraram-se os implicantes primos: A $\overline{A}C + \overline{B}C$

**4ª Iteração:**

**2º Passo:** Como o literal essencial C foi selecionado na iteração anterior deve-se excluí-lo permanentemente.

Pode-se perceber através da Tabela 27 que restou apenas uma coluna. Em virtude dessa característica o algoritmo deve ser interrompido, pois um dos critérios de parada é atendido.

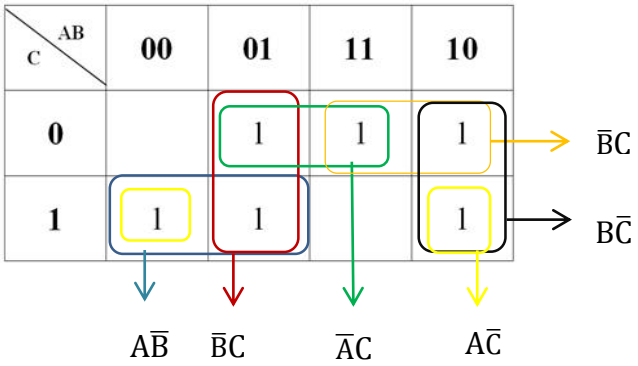
**Tabela 27** - Tabela final do algoritmo Clause-Column Table Aprimorado

| **M7** |
|---|
| $\overline{A}$ |
| $\overline{B}$ |
| $\overline{C}$ |

**Fonte:** Próprio Autor

Obtiveram-se como solução conjunto de solução os implicantes primos $A\overline{B} + A\overline{C} + \overline{A}B + B\overline{C} + \overline{A}C + \overline{B}C$. Salienta-se que o critério de parada interrompeu a execução do algoritmo, evitando a geração de termos nulos, como pode-se observar no mapa de Karnaugh apresentado na Figura 34. Caso contrário, o método continuaria sendo executado, gerando os literais $\overline{A} + \overline{B} + \overline{C}$ como se fossem implicantes primos, como apresentou-se na seção 4.2 através do algoritmo original Clause-Column Table.

**Figura 34-** Mapa de Karnaugh representando a função $F_5\ (A, B, C) = \prod M(0\ 7)$.




**Fonte:** Próprio Autor

Através da minimização da função $F_3$, seção 5.2, utilizando-se do algoritmo Clause-Column Table original, notou-se que o critério de aleatoriedade na seleção dos literais é falho. A seleção influencia diretamente na geração de um implicante primo e, por conseguinte num conjunto de implicantes primos factível ou não.

Uma possibilidade para resolver tal situação é aplicar ao algoritmo Aprimorado, características de algoritmos bio-inspirados, como por exemplo, o ACO - *Ant Colony Optimization* (DORIGO, 1999), que permite traçar o melhor caminho para encontrar a os implicantes primos factíveis. Sendo assim, a função $F_3$ foi minimizada utilizando-se dois caminhos; o primeiro escolhendo o literal B; o segundo o literal $\overline{B}$.

**1º Passo:** Forme uma tabela em que cada coluna é composta por um maxtermo da função. Se a função booleana inicial estiver na forma de produto de soma, execute o próximo passo, senão obtenha o seu dual.

$$F_3(A, B, C, D) = \prod M(1, 2, 4, 7)$$

**Tabela 28** - Tabela inicial do método Aprimorado

| M1 | M2 | M4 | M7 |
|---|---|---|---|
| A | A | $\overline{A}$ | $\overline{A}$ |
| B | $\overline{B}$ | B | $\overline{B}$ |
| $\overline{C}$ | C | C | $\overline{C}$ |

**Fonte:** Próprio Autor

**2º Passo:** Analisando a Tabela 28 nota-se que nenhum um critério é atendido. Execute o próximo passo.

**3º Passo:** Todos os literais possuem a mesma incidência, desse modo qualquer um destes pode ser selecionado. Seleciona-se o A.

**1 º caminho:**

**4º Passo:** A coluna que contiver o literal selecionado deve ser excluída, e o seu complemento $\overline{A}$ das demais colunas. Desse modo, a tabela reduzida é formada.

Através da Tabela 29 pode-se notar que há mais de uma coluna, portanto deve-se selecionar o literal que mais aparece novamente. Neste caso, seleciona-se o literal B, formando outra Tabela reduzida apresentada na Figura 36.

**Tabela 29** - Tabela Reduzida da função $F_3$

| M4 | M7 |
|---|---|
| B | $\overline{B}$ |
| C | $\overline{C}$ |

**Fonte:** Próprio Autor

Realiza-se a operação AND entre o literal selecionado e a tabela reduzida mostrada na Figura 36 para obter os implicantes primos.

**Figura 35** – Tabela reduzida utilizada na geração dos implicantes primos

AB **and**

| M7 |
|---|
| $\overline{C}$ |

**Fonte:** Próprio Autor

Depois de realizar a operação AND gerou-se o implicante primo $AB\overline{C}$.

**2 º caminho:**

**3º Passo:** Todos os literais possuem a mesma incidência, desse modo qualquer um dos literais pode ser selecionado aleatoriamente. Seleciona-se então o A.

**4º Passo:** A coluna que contiver o literal selecionado deve ser excluída, e o seu complemento $\overline{A}$ das demais colunas, formando a Tabela 30 que foi reduzida.

**Tabela 30** – Tabela reduzida referente ao 2º caminho da função $F_3$

| **M4** | **M7** |
|---|---|
| B | $\overline{B}$ |
| $\overline{C}$ | C |

**Fonte:** Próprio Autor

Pode-se notar que a tabela possui mais de uma coluna, portanto deve-se selecionar o literal que mais aparece novamente. Neste caso, seleciona-se o literal $\overline{B}$, formando outra tabela reduzida apresentada na Figura 37.

Realiza-se a operação AND entre o literal selecionado e a tabela reduzida para obter os implicantes primos.

**Figura 36** – Tabela Reduzida para obter os implicantes primos da ultima iteração

$A\overline{B}$ **and**

| **M4** |
|---|
| C |

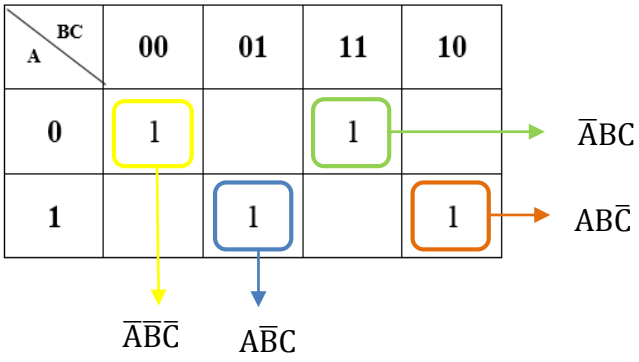
**Fonte:** Próprio Autor

Realizando a operação AND gerou-se o implicante primo $A\overline{B}C$.

Concluiu-se a minimização quando um dos critérios de parada foi atendido, gerando duas soluções, uma para cada caminho. Através do 1º caminho geraram-se os implicantes primos: $AB\overline{C} + \overline{A}BC + \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B} + \overline{A}C$. Já através do 2º caminho geraram-se: $AB\overline{C} + A\overline{B}C + \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B} + \overline{A}C$.

Posteriormente, eliminam-se os termos redundantes, formando o conjunto de solução: $A\bar{B}\bar{C} + A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B} + \bar{A}C$.

**Figura 37** – Mapa de Karnaugh para a função $F(A, B,C) = \sum m\,(0, 3,5, 6)$




**Fonte**: Próprio Autor

Em relação à deficiência do critério de aleatoriedade propõe-se a metodologia que demonstrou-se neste exemplo para saná-la. Contudo a mesma não foi implementada neste trabalho.

Após a geração dos implicantes primos deve-se realizar a fase de cobertura para encontrar os implicantes primos essenciais que realizam a cobertura mínima da função. Esta cobertura foi formulada com sendo um problema de programação linear inteira 0 e 1.

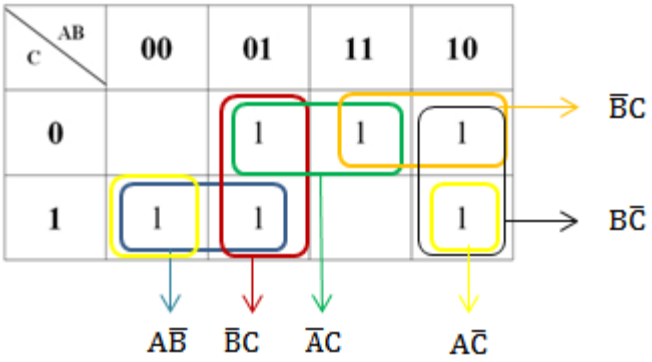
## 6.2 COBERTURA DE FUNÇÕES BOOLEANAS ATRAVÉS DE PROGRAMAÇÃO INTEIRA 0 E 1

A obtenção de uma cobertura mínima de uma função booleana com o menor número de termos produtos pode ser resolvida em duas etapas. A primeira consiste em gerar todos os implicantes primos da função. Vários algoritmos tratam do assunto. Nos capítulos 5 e 6, apresentaram-se os métodos de geração de implicante primos Clause-Column Table Aprimorado, a 1ª fase do método de Quine-McCluskey e o Expander (EMER, 2002) que implementa o Teorema de expansão de Shannon.

A segunda consiste em selecionar um conjunto mínimo de implicantes, que é denominado de problema de cobertura, que pode ser expresso como um problema de programação linear restritamente inteiro (SILVA, 1993).

Considere a função booleana F(A, B, C) = $\sum m\ 1, 2, 3, 4, 5, 6$, minimizada na seção 6.1 através do método Clause-Column Table Aprimorado, obtiveram-se os implicantes primos: $A\overline{B} + A\overline{C} + \overline{A}B + B\overline{C} + \overline{A}C + \overline{B}C$, cujo mapa de Karnaugh está apresentado na Figura 38.

**Figura 38** – Mapa de Karnaugh para a função $\sum$ m (1,2,3,4,5,6)




Fonte: Próprio Autor

A partir dos implicantes primos obtidos formula-se a cobertura da função como sendo um problema de programação linear inteira 0 e 1. As variáveis do problema são os implicantes primos e as restrições são inequações de "≥ 1", em que o lado esquerdo é a soma dos implicantes primos que cobrem cada mintermo.

Com tal característica, o número de implicantes primos é igual o de variáveis do problema e o número de mintermos é igual o de restrições.

Considerando-se: a = $A\overline{B}$, b = $A\overline{C}$, c = $\overline{A}B$, d = $B\overline{C}$, e = $\overline{A}C$, f = $\overline{B}C$ têm-se o problema de programação linear:

MIN 3. (a + b + c + d + e + f)

a + b ≥ 1

c + e ≥ 1

c + d ≥ 1

a + f ≥ 1

e + f ≥ 1

b + d ≥ 1

a, b, c, d, e, f ∈ {0 , 1}

Nota-se que todos os mintermos estão inclusos na cobertura da função e correspondem às restrições no problema de cobertura, cuja solução é dada por $A\overline{B} + B\overline{C} + \overline{A}C$ e o custo é igual a 9.

O critério de custo empregado é a soma aritmética ponderada de todos os implicantes da função. Cabe salientar que o custo inicial era 18, posteriormente a formulação e solução do problema de cobertura o custo foi reduzido pela metade.

Na próxima seção apresenta-se a estrutura básica utilizada na implementação dos métodos Clause-Column Table e Clause-Column Table Aprimorado.

## 6.3 ESTRUTURA DE DADOS UTILIZADA NA IMPLEMENTAÇÃO DOS MÉTODOS

Os algoritmos para a obtenção dos implicantes primos utilizando-se do método Clause-Column Table e do método Clause-Column Table Aprimorado foram implementados em linguagem de programação C ansi, que é uma linguagem compilada e estruturada.

Algumas caraterísticas da linguagem favorecem a implementação e otimização do método Clause-Column Table, como, a alocação dinâmica da memória.

Para formar as tabelas de proposições do método utilizou-se da alocação dinâmica, pois não se conhece inicialmente quantos são os termos e variáveis de entrada da função que será alocada na matriz.

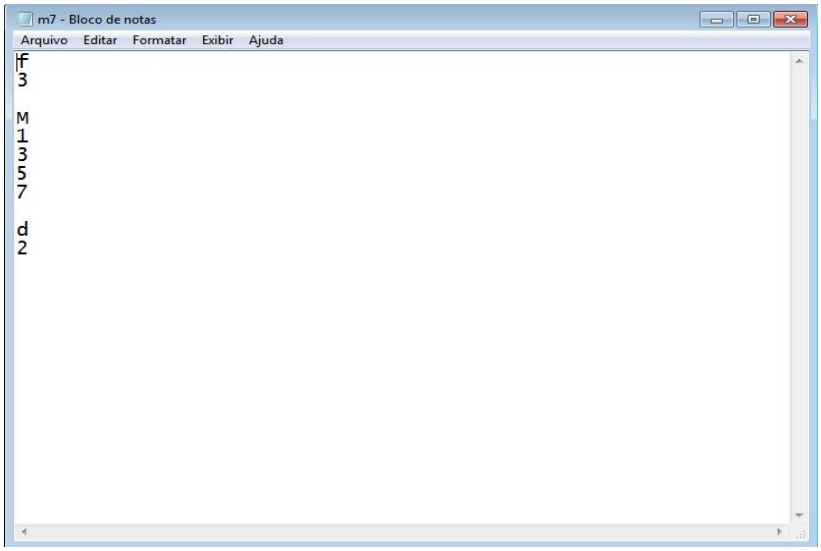
A matriz contém os maxtermos da função, que é manipulada através dos passos do algoritmo, formando ao final de cada iteração uma estrutura com os implicantes primos obtidos, que são então inseridos em uma lista encadeada. Quando um critério de parada do algoritmo é atingido a lista é percorrida a partir do inicio obtendo o resultado de todas as iterações.

Para utilização do programa Clause-Column Table original e Aprimorado deve-se seguir um padrão estabelecido, em que a entrada do algoritmo é através de um arquivo ".txt" e deve seguir as instruções abaixo:

1. A primeira linha do arquivo deve conter o caractere "f" referente à função;
2. Na segunda linha deve-se digitar o número de variáveis da função a ser minimizada;
3. A terceira linha deve permanecer em branco;
4. A quarta linha deve conter o caractere "M" referente à mintermo;

5. As linhas seguintes devem conter os mintermos da função na representação decimal. Cada linha deve conter apenas um mintermo;
6. A linha subsequente ao último mintermo deve permanecer em branco;
7. A próxima linha deve conter o caractere “d” referente ao *dont’care states;*
8. Caso haja *dont’care states,* eles devem ser digitados nas próximas linhas, como na Figura 39.

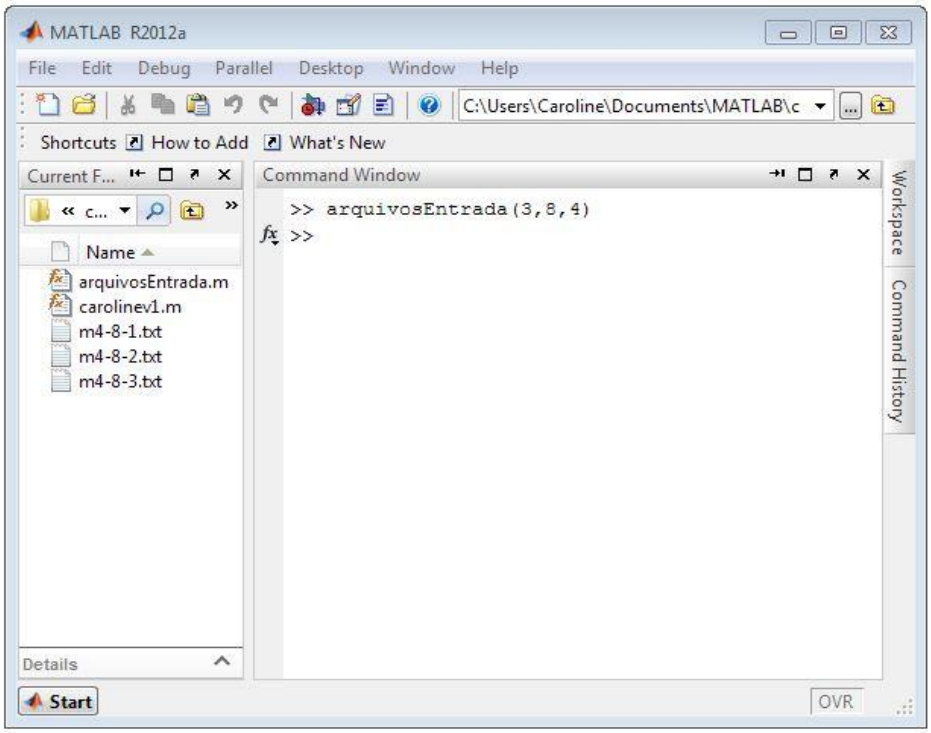
**Figura 39** - Entrada para execução do algoritmo Clause-Column Table Aprimorado




**Fonte:** Próprio Autor

Para gerar o arquivo de entrada“.txt” no padrão estabelecido pelo programa Clause-Column Column Table Aprimorado utilizou-se o programa Matlab ®, que é ilustrado na Figura 40. Observa-se que na tela principal possui uma função na seguinte forma: >> arquivosEntrada(3, 8, 4), cujo o primeiro número (3) representa a quantidade de funções, o segundo o número de mintermos e por último a quantidade de variáveis, tal função gera os arquivos com as características estabelecidas pelo usuário.

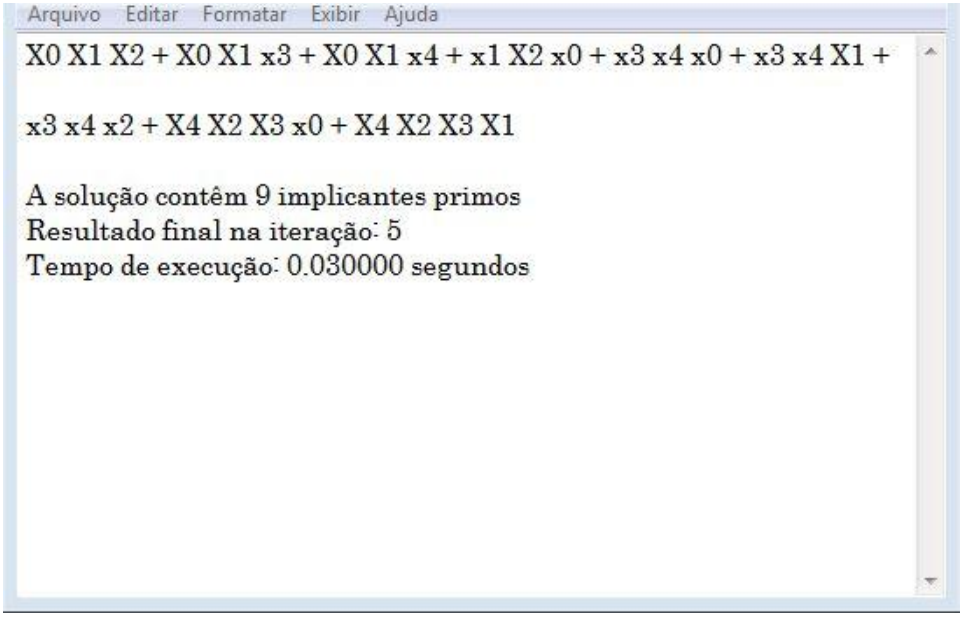
**Figura 40** – Tela do programa Matlab ®




**Fonte:** Próprio Autor

Após a execução do programa é gerado um arquivo ".txt" com o resultado da minimização, contendo o número de implicantes primos gerados, iterações que foram utilizadas parar obter a solução e o tempo de execução consumido pelo algoritmo. Os implicantes primos da função são representados pelos caracteres "x" e "X", em que o "x" representa uma variável não barrada e o "X" uma variável barrada ( $\overline{X}$ ), como pode-se observar na Figura 41.

**Figura 41** – Tela do Arquivo de entrada para execução do algoritmo Clause-Column Table Aprimorado




**Fonte:** Próprio Autor

No próximo capitulo apresenta-se os resultados obtidos através deste trabalho. Os resultados são apresentados em tabela e gráficos.

# 7 RESULTADOS OBTIDOS

Com objetivo de avaliar a eficiência da geração de implicantes primos através do método Clause-Column Table Aprimorado foram utilizadas centenas de funções booleanas que foram comparadas com os métodos Clause-Column Table, Quine-McCluskey e o Expander. Os parâmetros de comparação utilizados foram à quantidade de implicantes primos gerados, o número de iterações e o tempo de execução.

Os métodos Clause-Column Table original e Aprimorado geraram implicantes primos para função com até 60 variáveis de entrada. Entretanto, para comparar os resultados foram utilizadas funções com até 12 variáveis de entrada, pois quando este número excede são gerados resultados muito extensos, impossibilitando a validação destes.

## 7.1 COMPARAÇÃO DOS MÉTODOS CLAUSE-COLUMN TABLE ORIGINAL E APRIMORADO

Algumas funções minimizadas, através do método Clause-Column Table Aprimorado (I) e do método Clause-Column Table (II), foram tabeladas para melhor ilustrar os resultados gerados.

Nota-se na Tabela 31 que o método Clause-Column Table Aprimorado foi superior ao original.

Na Figura 42 apresenta-se um gráfico comparando a quantidade de iterações utilizadas na geração de implicantes primos pelos dois métodos. Pode-se perceber que o número de iterações utilizadas pelo Clause-Column Table Aprimorado é menor do que as utilizadas pelo método original. À medida que aumenta-se o número de variáveis, o método Aprimorado torna-se significativamente mais vantajoso, pois o número de iterações é ainda mais reduzido com a inclusão do teorema da adjacência

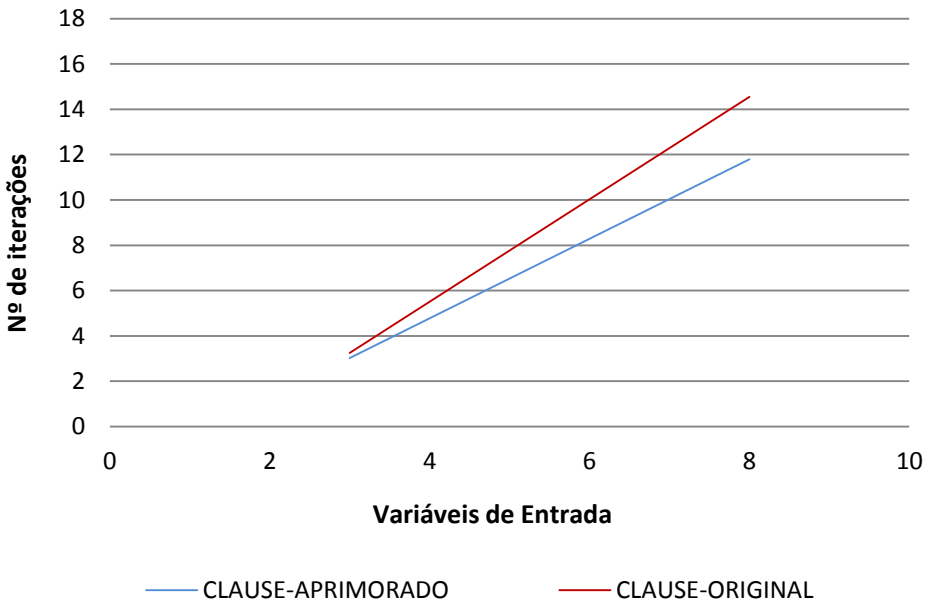
**Tabela 31** - Comparação entre os resultados apresentados pelo método original e aperfeiçoado ao minimizar algumas funções.

| **Características da função** | | **Tempo de Execução (s)** | | **Quantidade de implicantes** | | **Número de Iterações** | |
|---|---|---|---|---|---|---|---|
| Nº de variáveis | Nº de mintermos | I | **II** | I | **II** | I | **II** |
| 4 | 12 | 0,10 | **0,15** | 7 | **11** | 5 | **8** |
| 8 | 140 | 2,34 | **3,40** | 13 | **17** | 11 | **17** |
| 12 | 199 | 258,90 | **322,80** | 22 | **25** | 15 | **24** |

**Fonte:** Próprio Autor

Utilizando-se de outra base de dados mais comparações e resultados foram gerados. Estes resultados são apresentados através dos gráficos abaixo.

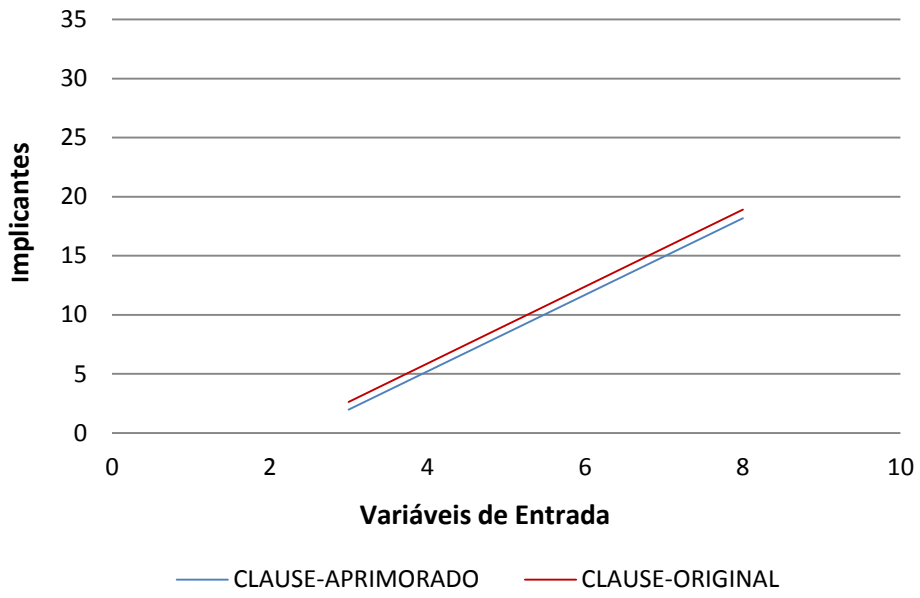
**Figura 42** - Gráfico comparativo da quantidade de iterações




**Fonte:** Próprio Autor

Na Figura 43 apresenta-se um gráfico, comparando o número de implicantes primos gerados pelos métodos Aprimorado e Original. É possível perceber claramente que os mesmos apresentam comportamentos semelhantes, pois a diferença está no fato de que o método Aprimorado gera menos termos nulos, que são considerados implicantes primos pelo método Original.

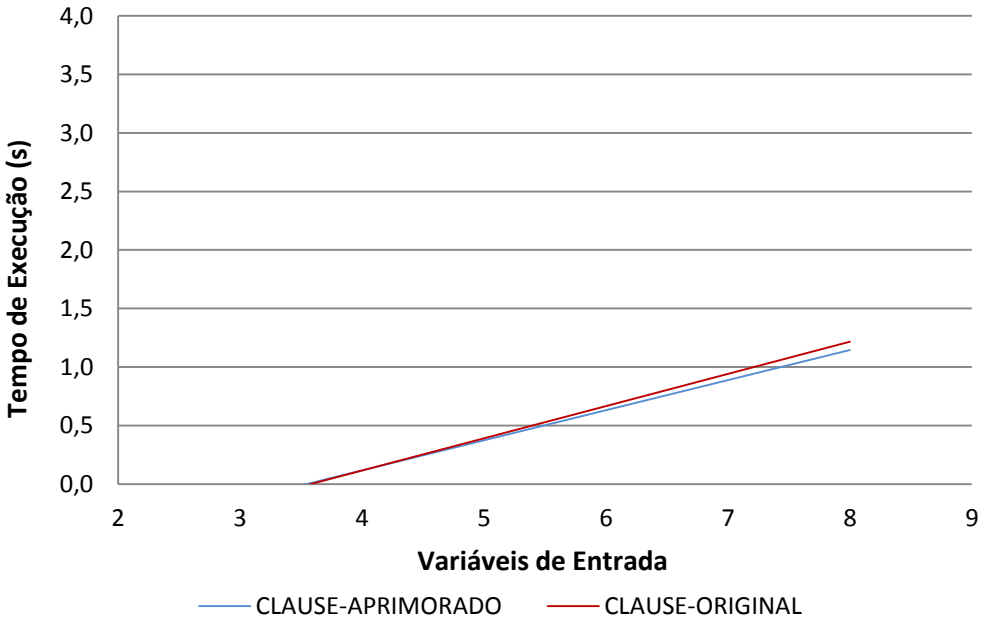
**Figura 43** - Gráfico comparativo do número de implicantes primos




**Fonte:** Próprio Autor

Como previsto, os dois métodos utilizam praticamente o mesmo tempo de execução para obter o conjunto de implicantes primos, visto que a estrutura de dados utilizada na implementação dos algoritmos são iguais. Na Figura 44 apresenta-se o tempo de execução utilizado pelos métodos Clause-Column Table original e Aprimorado.

**Figura 44** – Gráfico comparativo do tempo de execução utilizado na execução dos algoritmos




**Fonte:** Próprio Autor

## 7.2 COMPARAÇÃO DO MÉTODO APRIMORADO COM A 1ª FASE DO QUINE-MCCLUSKEY E O PROGRAMA EXPANDER

Para comparar o método Aprimorado com a 1ª fase do método Quine-McCluskey e o Expander utilizou-se somente a geração de implicantes como parâmetro, pois as estruturas de dados empregadas na implementação são diferentes e, portanto comparar o tempo de execução e o número de iteração seria impraticável, como é citado no trabalho de Besslich (1979).

Na Tabela 32 nota-se que o método Clause-Column Table Aprimorado gera menos implicantes do que a 1ª fase do método Quine-McCluskey.

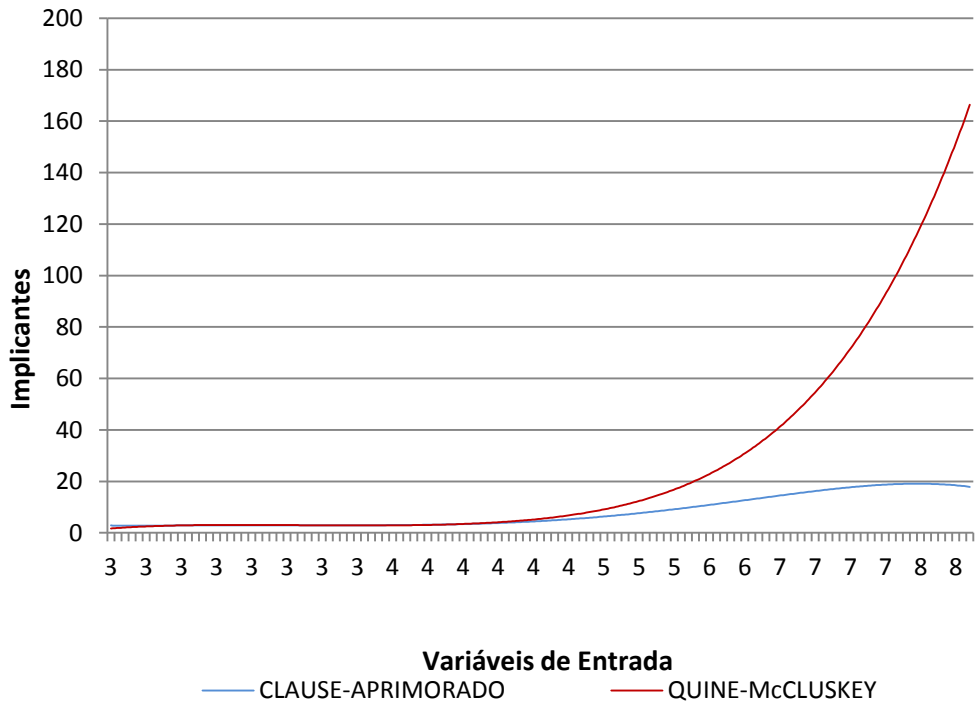
**Tabela 32** – Implicantes gerados pelos métodos Clause- Column Table Aprimorado e Quine-McCluskey.

| **Características da função** | | **Quantidade de implicantes** | |
|---|---|---|---|
| Nº de mintermos | Nº de variáveis | Clause-Column Table Aprimorado | **Quine-McCluskey** |
| 24 | 5 | 9 | **18** |
| 30 | 6 | 6 | **16** |
| 80 | 7 | 19 | **79** |

Fonte: Próprio Autor

Através da Figura 45, apresenta-se a quantidade de implicantes primos gerados à medida que o número de variáveis aumenta. Constata-se com clareza a eficiência do método Clause-Column Table Aprimorado, pois gera menos implicantes que a 1ª fase do método Quine-McCluskey.

**Figura 45** - Gráfico comparativo do número de implicantes gerados pelos métodos Aperfeiçoado e 1ª fase do Quine-McCluskey




**Fonte:** Próprio Autor

Através da Tabela 33, pode-se constatar que o método Clause-Column Table é superior na geração dos implicantes primos quando comparado com o Expander.

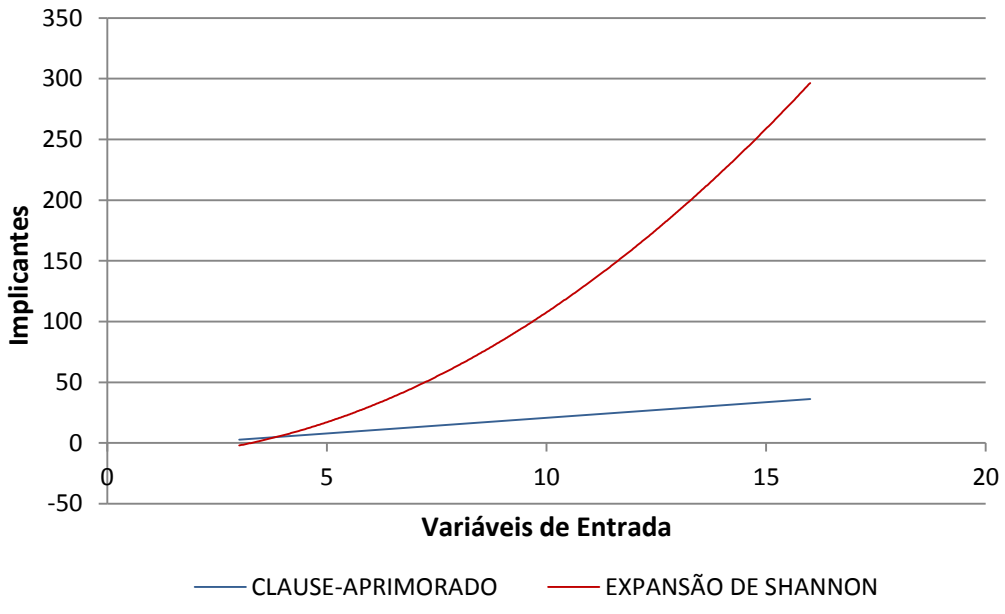
**Tabela 33** – Implicantes primos gerados pelos métodos Clause-Column Table Aprimorado e o programa Expander.

| **Características da função** | | **Quantidade de implicantes** | |
|---|---|---|---|
| Nº de mintermos | Nº de variáveis | Clause-Column Table Aprimorado | **Teorema de Expansão de Shannon** |
| 40 | 6 | 12 | **20** |
| 128 | 8 | 14 | **17** |
| 189 | 11 | 28 | **141** |
| 200 | 12 | 41 | **161** |

**Fonte:** Próprio Autor

O gráfico da Figura 46 apresenta a comparação entre o método Clause-Column Table Aprimorado e o programa Expander. Constata-se novamente que o método Clause-Column-Table Aprimorado gera menos implicantes primos.

**Figura 46** - Gráfico comparativo do número de implicantes gerados pelos métodos Aperfeiçoado e o programa Expander.




**Fonte:** Próprio Autor

# 8 CONCLUSÃO

Este trabalho teve como principal finalidade encontrar métodos eficientes para a geração de implicantes primos no contexto de minimização de funções booleanas. Dentre os métodos encontrados na literatura, o método de Das denominado Clause-Column Table foi estudado e posteriormente aprimorado, eliminando algumas deficiências encontradas no mesmo, originando assim uma nova versão chamada de Clause-Column Table Aprimorado.

Os métodos foram implementados em linguagem de programação C, através de uma técnica tabular e iterativa, que gera um conjunto reduzido de implicantes primos suficientes para obter a solução de custo mínimo.

O algoritmo aprimorado foi comparado com os métodos Clause-Column Table, 1ª fase do Quine-McCluskey e com o Expander que implementa o Teorema de Expansão de Shannon, a fim de apresentar a eficiência na geração dos implicantes primos.

Na comparação do método Clause-Column Table com o Aprimorado utilizou-se como parâmetro o tempo de execução, a quantidade de iterações e o número de implicantes primos gerados. Entretanto na comparação com a 1ª fase do método Quine-McCluskey e Expander utilizou-se como parâmetro somente a geração de implicantes primos, visto que os programas possuem estruturas de dados diferentes.

Pode-se constatar que o número de iterações geradas através do método aprimorado são inferiores as geradas pelo método original de Das. Constatou-se também que o método Clause-Column Table Aprimorado evitou a geração dos termos nulos na maioria dos casos analisados e diminuiu o número de implicantes primos gerados.

Quanto ao tempo de execução verificou-se que o método aprimorado conclui o processo de geração de implicantes primos em tempo aproximado ao original, visto que a estrutura de dados utilizada para implementá-los foi a mesma.

Em relação à geração de implicantes primos constatou-se que o método aprimorado gerou menos implicantes primos que a 1ª fase do Quine-McCluskey e Expander, confirmando assim as previsões iniciais de que o novo algoritmo proposto neste trabalho é eficiente na geração dos implicantes primos.

O problema de cobertura foi formulado como um problema de programação linear inteira 0 e 1. Por conseguinte, constatou-se que o método Clause-Column Table Aprimorado possui uma formulação mais simplificada do que a gerada pelo método Quine-McCluskey.

Como continuação deste trabalho o método Clause-Column Table Aprimorado pode ser expandido para simplificar funções booleanas com múltiplas saídas e também implementá-lo

utilizando técnicas de algoritmos genéticos que foram citadas anteriormente para sanar o problema da aleatoriedade.

# Referências


AKERS, B. Binary Decision Diagrams. **Ieee Transactions On Computers,** New Jersey**,** p. 509-516. jun. 1978.

AMARU, L. et al. A novel data-structure algorithms for efficient logic optimization. In: ANNUAL DESIGN AUTOMATION CONFERENCE ON DESIGN AUTOMATION CONFERENCE- DAC**,** 14., 2014. San Francisco. **Proceedings…** San Francisco: [s.n.], 2014. v. 1, p.1-6.

ASTHANA, R.; VERMA, N.; RATAN, R. Generation of boolean functions using genetic algorithm for cryptographic applications. In: IEEE ADVANCE COMPUTING CONFERENCE- IACC, 4., 2014, Gurgaon. **Conference…** Gurgaon: [s.n.], 2014. p. 1361-1366.

BANSAL, M.; AGARWAL, A. Genetic algorithm for ordering and reduction of BDDs for MIMO circuits. In: INNOVATIVE COMPUTING TECHNOLOGY INTERNATIONAL CONFERENCE ON- INTECH. , 3., 2013, London. **Conference…** London: [s.n.], 2013. p. 411-414.

BAŞÇIFTÇI, F.; KAHRAMANLI, Ş. Fast computation of the prime implicants by exact direct-cover algorithm based on the new partial ordering operation rule. **Advances in Engineering Software**, London, v. 42, n. 6, p. 316–321, jun. 2011.

BESSLICH, B. Anatomy of boolean-function simplification. **Computers And Digital Techniques**, New York, v. 2, n. 1, p.7-12, fev, 1979.

BISWAS, N. N.; SRIKANTH, C.; JACOB, J. Cubical CAMP for minimization of boolean functions. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN, 9., 1996, Bangalore. . **Conference…** Bangalore: [s.n.], 1996. p. 265-269,

BRAYTON, R et al. **Logic minimization algorithms for VLSI synthesis.** Higham: Kluwer Academic, 1984.

BROWN, D. W; TEKTRONIX, I.; BEAVERTON, Or. A State-Machine Synthesizer - SMS. In: DESIGN AUTOMATION CONFERENCE ON**,** 18., 1981, Nashville. **Conference…** Nashville: [s.n.], 1981. p. 301-305.

CARUSO, G. Heuristic algorithm for the minimization of generalized boolean functions. **IEEE…**, Kansas, v. 135, p. 108 -116, 1988.

CHOWDURY, M D et al. A composition technique of multiple switching functions based on BDD. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY- ICCIT, 13., 2010, Dhaka. . **Conference…** Dhaka: [s.n.], p. 337-342. 10 dez. 2010.

DAGHLIAN, J. **Lógica e álgebra de boole.** 4. ed. São Paulo: Atlas, 1995. 152 p.

DAS, S. R.; KHABRA, N. S. Clause-column table approach for generating all the prime implicants of switching functions. **IEEE Computer Society**, Washington, v. 21, n. 11, p. 1239-1246, nov. 1972.

DEHARBE, D. et al. Computing prime implicants. **Computer-aided Design (fmcad),** Portland, v. 1, p. 46-52, out. 2013.

DORIGO, M.; CARO, G. di. Ant colony optimization: a new meta-heuristic. In: CONGRESS ON EVOLUTIONARY COMPUTATION- CEC, 1999, Washington. **Proceedings**... Washington: [s.n.], 1999. p.1470-1477.

EMER, F. R. P. **Formulação matemática do problema de otimização de funções booleanas através do método de expansão de Shannon.** 2002. 91 f. Dissertação (Mestrado em Engenharia Elétrica) – Faculdade de Engenharia, Universidade Estadual Paulista, Ilha Solteira, 2002.

FALLAH, Farzan; LIAO, Stan; DEVADAS, Srinivas. Solving Covering Problems Using LPR-Based Lower Bounds. **Ieee Transactions On Very Large Scale Integration (vlsi) Systems.** Hong Kong, p. 9-17. dez. 2000.

FISER, P.; HLAVIEKA, J. On the use of mutations in boolean minimization. In: EUROMICRO SYMPOSIUM ON DIGITAL SYSTEMS DESIGN, PROCEEDING, 11., 2001, Varsóvia. **Proceeding…** Varsóvia: [s.n.], 2001. p. 300-307.

HALDER, N. et al. Analysis of composition techniques for combinational switching functions using reduced ordered binary decision diagrams (ROBDDs). In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION TECHNOLOGY- ICCIT, 10., 2007, Dhaka. **Conference…** Dhaka: [s.n.] , 2007. p.1-5.

HATA, Y. et al. Multiple-valued logic minimization by genetic algorithms. , In: INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC, 27., 1997, Canadá. **Proceedings.,.** Canadá: [s.n.], 1997. p. 97-102.1997.

HONG, S. J. et al. A Heuristic approach for logic minimization. **Ibm Journal Of Research And Development,** New York, v.1, p. 443-458. set. 1974.

HLAVICKA, J.; FISER, P.. A heuristc method of two-level logic synthesis. In: WORLD MULTICONFERENCE ON SYSTEMICS, CYBERNETICS AND INFORMATICS- SCI, 5., 2001, Orlando. **Conference…** Orlando: [s.n.], 2001. v. 12, p. 283-288.

JAIN, T. K.; KUSHWAHA, D. S.; MISRA, A. K. Optimization of the quine-mccluskey method for the minimization of the boolean expression. In: INTERNATIONAL CONFERENCE ON AUTONOMIC AND AUTONOMOUS SYSTEMS, 4., 2008, Roma. **Conference…** Roma: [s.n.] , 2008. p. 165-168.

KARNAUGH, M. The map method for synthesis of combinational logic circuits. **AIEE Trans**, **American Institute of Electrical Engineers, Part I. Communication and Electronics, Transactions of the,** Lugar de Publicação, v. 72, n. 5. p. 593-598, 1953 LIN, Shun-shii; WEI, Chun-jen. A new approach for minimization of binary decision diagrams. **Electrical And Computer Engineering, Canadian Journal Of,** Canadá, p. 207-214. set. 2005.

LIN, Y.; GENG, R. MLBM: machine-learning-based minimization algorithm for boolean functions. In: IEEE INTERNATIONAL SYMPOSIUM ON, 9., 2009, Seoul. **Symposium**… Seoul: [s.n.] , 2009. p. 1160-1165.

MALIK, A. A.; BRAYTON, R. K.; NEWTON, A. R. Reduced offsets for minimization of binary-valued functions. **Transactions on Computer-Aided Design of Integrated Circuits and Systems, IEEE ,** New York**,** v. 10, n. 4, p. 413-426. 1991.

McCLUSKEY, E. Minimization of boolean function. **The Bell System Techinal Journal**, Kansas, v. 35, p. 1417-1444, 1956.

MIKHTONYUK, S. et al. Boolean function minimization based on Intel architecture. INTERNATIONAL CONFERENCE ON MODERN PROBLEMS OF RADIO ENGINEERING, TELECOMMUNICATIONS AND COMPUTER SCIENCE, , 2008, Lviv-Slavsko. **Proceedings…** Lviv-Slavsko: [s.n., 2008. p. 626-629, 2008.

MINZYUK, V. Integers sorting method for boolean functions minimization. In: INTERNATIONAL CONFERENCE ON MODERN PROBLEMS OF RADIO ENGINEERING TELECOMMUNICATIONS AND COMPUTER SCIENCE- TCSET, 11., Lviv-Slavsko. **Conference…** Lviv-Slavsko: Lviv Polythecnic National University, 2012. p. 61-61.

MOHAMED, S.; PERKOWSKI, M.; JOZWIAK, L. Fast minimization of multi-output boolean functions in sum-of-condition-decoders structures. In: CONFERENCE NEW FRONTIERS OF INFORMATION TECHNOLOGY- EUROMICRO, 23., 1997, Budapeste . **Conference…** Budapeste: [s.n.], 1997. p. 31-38.

PRASAD, P. W.; Chandana; BEG, A.; SINGH, A. K. Effect of quine-mccluskey simplification on boolean space complexity. In: CONFERENCE ON INNOVATIVE TECHNOLOGIES IN INTELLIGENT SYSTEMS AND INDUSTRIAL APPLICATIONS- CITISIA, 2009, Malaysia. **Conference…** Malaysia: [s.n.], 2009. p. 165-170.

QAWASMEH, E. E.; PICHAPPAN, P.; ALFITIANI, A. Development and investigation of a new compression technique using Boolean minimizations. In: INTERNATIONAL CONFERENCE ON MODERN PROBLEMS OF RADIO ENGINEERING TELECOMMUNICATIONS AND COMPUTER SCIENCE- TCSET, 11., 2012, Lviv-Slavsko , 2012. **Conferende…** Lviv-Slavsko: s.n.], 2012. p. 21-24.

QUINE, W**.** The problem of simplifying truth functions. **American Mathematical Monthly,** New York, v. 59, p. 521-531. 1952.

RATHORE, T. S.; JAIN, A. A systematic map method for realizing minimal logic functions of arbitrary number of variables. In: INTERNATIONAL CONFERENCE ON CIRCUITS, SYSTEMS, COMMUNICATION AND INFORMATION TECHNOLOGY APPLICATIONS- CSCITA, 11., 2014, Mumbai. **Conference…** Mumbai: [s.n.], 2014. p. 81-86.

SAFAEI, J.; BEIGY, H. Quine-McCluskey classification. In: INTERNATIONAL CONFERENCE ON COMPUTER SYSTEMS AND APPLICATIONS- AICCSA, 7., 2007, Beaumon. **Conference…** Beaumon: [s.n.], 2007. p. 404-411.

SARKAR, M.; GHOSAL, P.; MOHANTY, S. P. Reversible circuit synthesis using ACO and SA based quine-mccluskey method: circuits and systems (mwscas). 2013 IEEE 56TH In: INTERNATIONAL MIDWEST SYMPOSIUM ON, 56., 2013, Columbus. **Symposium…** Columbus: [s.n.], 2013. v. 1, p. 416-419.

SCHEINMAN, A. H. A method for simplifying boolean functions**. The Bell System Technical Journal**, New York, v. 41, n. 4, p. 1337-1346. July, 1962

SHANNON, C. E. The synthesis of two-terminal switching circuits. **The Bell System Technical Journal,** New York, v. 28, n. 1, p. 59-98, 1948.

SHEIDAEIAN, H.; ZOLFAGHARI, B.; MOZAFFARI, S. P. SPIDERS: swift prime implicant derivation through exhaustive rotation and sort. In: INTERNATIONAL CONFERENCE ON NETWORKING AND INFORMATION TECHNOLOGY- ICNIT, 1., 2010, Manila. **Conference…** Manila: [s.n.], 2010. p. 24-28.

SILVA, A. C. R. da. **Contribuição à minimização e simulação de circuitos lógicos.** 1989. 138 f. Dissertação (Mestrado em Engenharia Elétrica) - Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, Campinas, 1989.

SILVA, A. C. R. da. **Contribuição à síntese de circuitos digitais utilizando programação linear inteira 0 e 1.** 1993. 119 f. Tese (Doutorado)- Curso de Engenharia Elétrica, Universidade Estadual de Campinas, Campinas, 1993.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas digitais:** princípios e aplicações. 10. ed. [S.l.]: Prentice Hall Brasil, 2007. 804 p.

Valachi, A.; AIGNATOAIEI, B. I.; TIMIS, M. G. The comparative study of two analytical methods for detection and elimination of the static hazard in combinational logic circuits. In: INTERNATIONAL CONFERENCE ON SYSTEM THEORY, CONTROL, AND COMPUTING- ICSTCC, 15., 2011, Sinaia. **Conference…** Sinaia: [s.n.], 2011. p. 1- 4.

WU, Y. et al. OBDD minimization based on two-level representation of boolean functions. **Computers, IEEE Transactions on** , Genova, v. 49, n. 12, p. 1371-1379, dec. 2000.