



UNIVERSIDADE ESTADUAL PAULISTA

Programa de Pós-Graduação em Engenharia Elétrica

**AVALIAÇÃO DE ALGORÍTMO DE
PROGRAMAÇÃO MATEMÁTICA NA RESOLUÇÃO
DE PROBLEMAS DE COBERTURA DE FUNÇÕES
BOOLEANAS**



TERCIO ALBERTO DOS SANTOS

Ilha Solteira - SP



UNIVERSIDADE ESTADUAL PAULISTA – UNESP

FACULDADE DE ENGENHARIA – CAMPUS DE ILHA SOLTEIRA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

**AVALIAÇÃO DE ALGORÍTMO DE
PROGRAMAÇÃO MATEMÁTICA NA RESOLUÇÃO
DE PROBLEMAS DE COBERTURA DE FUNÇÕES
BOOLEANAS**

TERCIO ALBERTO DOS SANTOS

Dissertação apresentada à Faculdade de Engenharia de Ilha Solteira / FEIS / UNESP como parte dos requisitos necessários para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

ORIENTADOR: Prof. Dr. ALEXANDRE CÉSAR RODRIGUES DA SILVA

ILHA SOLTEIRA – SP

Agosto – 2003

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação/Serviço Técnico de Biblioteca e Documentação da FEIS/UNESP

Santos, Tercio Alberto dos

S237a Avaliação de algoritmo de programação matemática na resolução de problemas de cobertura de funções booleanas / Tercio Alberto dos Santos . -- Ilha Solteira : [s.n.], 2003
121 p. : il.

Dissertação (mestrado) – Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2003

Orientador: Alexandre César Rodrigues da Silva

Bibliografia: p. 64-65

1. Álgebra booleana. 2. Programação linear. 3. Algoritmos. 4. Simplex (Matemática). 5. Otimização matemática.

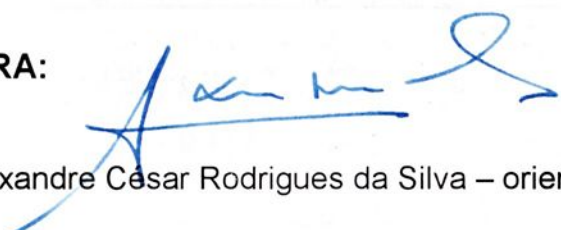
10101004

AVALIAÇÃO DE ALGORITMOS DE PROGRAMAÇÃO MATEMÁTICA NA RESOLUÇÃO DE PROBLEMAS DE COBERTURA DE FUNÇÕES BOOLEANAS

Tércio Alberto dos Santos

DISSERTAÇÃO SUBMETIDA À FACULDADE DE ENGENHARIA - CAMPUS DE
ILHA SOLTEIRA – UNESP – COMO PARTE DOS REQUISITOS NECESSÁRIOS
PARA A OBTENÇÃO DO TÍTULO DE MESTRE EM ENGENHARIA ELÉTRICA

COMISSÃO EXAMINADORA:


Prof. Dr. Alexandre César Rodrigues da Silva – orientador


Prof. Dr. Celso Correia de Souza


Profª Drª Mariângela de Carvalho Bovolato

Ilha Solteira-SP, agosto de 2003.

Proc. 063/04 - NPD 16

1210001440



UNESP - "CAMPUS DE LULA SOLTEIRA"	
SERVIÇO TÉCNICO DE BIBLIOTECA E DOCUMENTAÇÃO	
DATA DE CHEGADA	DATA DE TOMBO
18.03.04	31.03.04
REGISTRADO POR	TOMBO
Ailza	Te. 1440
AQUISIÇÃO	CLASSIFICAÇÃO
Ilustração Autor R\$ 10.00	S 237a

cc
Sys 216 502
Sys 56 187

Dedico este trabalho à minha esposa
Eridan e aos meus filhos Carlos
Eduardo e Tércio Filho, pelo apoio e
compreensão.

AGRADECIMENTOS,

Ao Prof. Dr. Alexandre César Rodrigues da Silva, por sua dedicação como orientador e pela sua atenção durante os momentos de nosso trabalho.

Ao Prof. Dr. Celso Correia de Souza, membro da comissão examinadora, pelas sugestões na correção desse texto.

À Prof^a Dr^a Mariângela de Carvalho Bovolato, membro da comissão examinadora, pela contribuição na formação da estrutura desse texto.

Aos meus professores Marcelo, Mantovani e Rubem, pela contribuição de seus conhecimentos ao ministrarem suas disciplinas.

Ao meu filho Carlos Eduardo da Silva Santos, que muito contribuiu na execução deste trabalho.

Aos Professores César Romero Macedo, Salustiano Ferreira da Luz e à minha amiga Carmem Luzia Azevedo e Silva que contribuíram na revisão gramatical.

Às secretárias da Seção de Pós-Graduação Adelaide Amaral dos Santos Passipieri, Maria de Fátima Sabino e Onilda Naves de Oliveira Akasaki, pela atenção.

Aos servidores administrativos da biblioteca João Josué Barbosa, Cleide Maria da Silva Ferreira, Neide Aparecida Palombo da Silva e Terezinha Alves de Souza pelo atendimento e atenção.

Ao colega e amigo José Marcos Anselmo de Macedo pelo apoio e acima de tudo, pela grande amizade.

Aos colegas e amigos do curso Marcos Anselmo, João Pires, Mirely Baldin, Leandro Oliveira pelo companheirismo nestes anos.

Aos meus familiares esposa e filhos, pelo apoio e compreensão da ausência durante esses anos.

À Fundação do Ensino Superior de Rio Verde "FESURV", pela confiança e apoio abrindo novos horizontes.

A Deus, por tudo.

RESUMO

Este trabalho objetiva estudar alguns algoritmos para a resolução de problemas de cobertura de funções booleanas. De início, estudou-se o algoritmo Branch and Bound, na solução de problemas de cobertura, para comparar com os resultados obtidos através do algoritmo Plano de Corte realizado em trabalhos anteriores. Para formular o problema de cobertura de funções booleanas, utilizaram-se métodos de otimização exata como o Mapa de Karnaugh, método de Quine-McCluskey e o algoritmo Expander, que se utiliza do método de expansão de Shannon, formulando assim, o problema de cobertura como um problema de programação linear inteiro binário 0 e 1. Resolveu-se o problema de programação linear inteiro binário pelo método Simplex, sendo que em alguns casos encontrou-se uma solução otimista (infactível) ou seja, uma solução fracionária para o problema. Como se pretende encontrar uma solução ótima, para isso, empregaram-se os métodos de Plano de Corte e de Branch and Bound. Verificando-se que os custos em cada estudo de caso nestes dois algoritmos foram iguais e os parâmetros: memória, tempo e número de iterações foram praticamente idênticos. Um outro algoritmo foi implementado, o método Dual Simplex e na comparação dos parâmetros comprovaram-se melhores resultados em relação aos dois outros algoritmos, em se tratando de tempo e em números de iterações, visto que a quantidade de memória foi praticamente igual nos três algoritmos implementados.

ABSTRACT

This work has the objective of studying some algorithms for the coverage problems resolution of booleanas functions. At the beginning, it studied the Branch and Bound algorithm, in the coverage problems solution, to compare results obtained through the Cut Plan algorithm accomplished in previous jobs. To formulate the booleanas functions coverage problem in the cousins' generation teasings, it used methods for exact optimization as Karnaugh's Map, method of Quine-McCluskey and the Expander algorithm that is used the expansion method of Shannon, formulating this way, the coverage problem as a problem of whole binary lineal programming 0 and 1. It solved the problem of whole binary lineal programming by the Simplex method, and in some cases met itself an optimistic solution (infactível) in other words, a fractional solution for the problem,, as it intends to find a great solution, for that, it employed the Cut Plan and Branch and Bound methods. Verifying that the costs in each study case in these two algorithms were alike and the parameters: memory, time and number of iterations were practically identical. another algorithm was implemented the Dual Simplex method and in the comparison of the parameters it proved better results regarding the two other algorithms, refering to time and numbers of iterations, since the memory quantity was practically alike in the three implemented algorithms.

SUMÁRIO

RESUMO	5
ABSTRACT	6
CAPÍTULO 1 – INTRODUÇÃO.....	13
1.1 – INTRODUÇÃO.....	13
1.2- REVISÃO BIBLIOGRÁFICA.....	15
1.3 - ORGANIZAÇÃO DO TRABALHO	16
CAPÍTULO 2 – MÉTODO PARA OTIMIZAÇÃO DE FUNÇÕES BOOLEANAS	18
2.1- EXPRESSÕES BOOLEANAS.....	18
2.2 - EXPRESSÕES BOOLEANAS DE CIRCUITOS.....	19
2.3 - MÉTODOS PARA OTIMIZAÇÃO DE FUNÇÕES BOOLEANAS.....	20
2.3.1- MAPA DE KARNAUGH.....	20
CAPÍTULO 3 – PROBLEMAS DE COBERTURA	27
3.1- MÉTODO SIMPLEX.....	27
3.2 - O ALGORITMO DO MÉTODO SIMPLEX.....	27
3.3 - MÉTODO DE PLANO DE CORTE.....	35
3.4 - MÉTODO DE BRANCH AND BOUND.....	38
3.5 – O MÉTODO DUAL SIMPLEX.....	44
CAPÍTULO 4 – APLICAÇÃO DOS MÉTODOS.....	52
4.1 - MÉTODO DUAL	54
4.2 – MÉTODO BRANCH AND BOUND.....	55
CAPÍTULO 5 – COMPARAÇÃO DOS PARÂMETROS.....	59
CAPÍTULO 6 – CONCLUSÃO.....	62
CAPÍTULO 7 – REFERÊNCIAS BIBLIOGRÁFICAS.....	64
APÊNDICE A.....	66
1- CÓDIGO FONTE DO PROGRAMA BRANCH2.....	66
2- CÓDIGO FONTE DO PROGRAMA DUAL.....	74

3- CÓDIGO FONTE DO PROGRAMA GOMO2	78
4- FUNÇÃO AUXILIAR (BRANCH2 E GOMO2).....	84
APÊNDICE B.....	87
1- ARTIGO PUBLICADO NO ICECE 2003.....	87
APÊNDICE C.....	93
1- ESTUDO DE CASOS.....	93

LISTA DE TABELAS

Tabela 2.1: Postulados.....	18
Tabela 2.2: Expressão obtida a partir da tabela verdade $S = A'B'C + A'BC + AB'C$	19
Tabela 2.3: Grupo 0.....	23
Tabela 2.4: Grupo 1.....	23
Tabela 2.5: Grupo 2.....	23
Tabela 2.6: Tabela de cobertura dos Implicantes Primos.....	24
Tabela 3.1: Tabela do quadro simplex	28
Tabela 3.2: Tabela do quadro simplex alternativo	28
Tabela 3.3: Entra a variável x_3 e sai a variável x_9	32
Tabela 3.4: Entra a variável x_1 e sai a variável x_8	33
Tabela 3.5: Apresenta a linha z_0 linearmente dependente da linha x_7	33
Tabela 3.6: Apresenta a função objetivo z_0 é nula.	34
Tabela 3.7: Apresenta a eliminação da função objetivo z_0	34
Tabela 3.8: Quadro da solução otimista.	34
Tabela 3.9: Quadro otimista com acréscimo da restrição de corte	37
Tabela 3.10: Quadro ótimo do método Plano de Corte.....	38
Tabela 3.11: Quadro otimista com acréscimo de uma nova restrição.....	42
Tabela 3.12: Quadro ótimo do subproblema P_1	42
Tabela 3.13: Quadro otimista com acréscimo de uma nova restrição.....	43
Tabela 3.14: Quadro ótimo do subproblema P_2	43
Tabela 3.15: Interpretação de factibilidade do dual no quadro simplex primal.....	47
Tabela 3.16 Apresenta o quadro dual simplex.....	48
Tabela 3.17: Tabela de cobertura dos Implicantes Primos.....	49
Tabela 3.18: Tabela inicial do quadro simplex x_1 entra na base e x_4 sai da base.	51
Tabela 3.19: Tabela inicial do quadro simplex x_2 entra na base e x_5 sai da base.	51
Tabela 3.20: Solução ótima da função booleana.....	51
Tabela 4.1: Tabela de cobertura dos implicantes primos	54
Tabela 4.2: Quadro da solução otimista	55
Tabela 4.3: Quadro da solução otimista acrescentada à restrição S.....	55
Tabela 4.4: Quadro ótimo obtido através da implementação do programa Branch2.....	56
Tabela 4.5: Quadro da solução otimista acrescentada à restrição S.....	56

Tabela 4.6: Quadro ótimo obtido através da implementação do programa Branch2.....56

Tabela 4.7: Quadro da Solução otimista com acréscimo da restrição S53

Tabela 4.8: Quadro da solução otimista com acréscimo da restrição S57

Tabela 4.9: Quadro ótimo obtido através da implementação do programa Gomo257

Tabela 5.1: Comparação dos programas: Gomo2, Branch2 e Dual.59

LISTA DE GRÁFICOS

Gráfico 4.1: Comparação de Tempo entre os programas.	58
Gráfico 4.2: Comparação da Memória entre os programas Branch2, Dual e Gomo 2	58
Gráfico 4.3: Comparação do número de Iterações entre os programas.....	58
Gráfico 5.1: Comparação do tempo dos programas: Branch2, Dual e Gomo2	59
Gráfico 5.2: Comparação das iterações dos programas: Branch2, Dual e Gomo2.....	60
Gráfico 5.3: Comparação da memória dos programas: Branch2, Dual e Gomo2	60

Figura 2.1: Comando Elétrico.....	19
Figura 2.2: Subcubos do Mapa de Karnaugh.....	21
Figura 2.3 Implicantes primos gerados na 1ª fase do método de Quine-McCluskey.	24
Figura 3.1: Formulação da função $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$	29
Figura 3.2: Apresentação do circuito equivalente lógico da função f_2	29
Figura 3.3: Formulação da função $f_3(A, B, C) = \sum m(2, 4, 5, 6, 7) + d(1, 3)$	48
Figura 4.1: Arquivo ex41.min de entrada do programa Pigene.....	53
Figura 4.2: Arquivo ex41.qmc de saída do programa Pigene.....	53
Figura 4.3: Apresentação do mapa de Karnaugh da função f_4	53

CAPÍTULO 1 – INTRODUÇÃO

Resolver problemas de cobertura consiste em encontrar uma solução que satisfaça todas as restrições do problema e que o custo da função objetivo, que representa o custo de implementação da função booleana, seja o menor possível.

A abordagem matemática dada a esse trabalho é muito oportuna, pois muitas ferramentas CAD (computer aided design) para circuitos microeletrônicos modelam alguns problemas de projetos de forma matematicamente precisa; permitindo, dessa forma, a análise do problema e o desenvolvimento de métodos para a solução.

Muitos dos problemas de projetos lógicos para circuitos digitais são de natureza discreta. Torna-se, então, necessário resolver decisões combinatórias e problemas de otimização.

1.1 – Introdução

Um problema de decisão é um problema com solução de valores binários, falso ou verdadeiro. Estes tipos de problemas são relacionados para testar algumas propriedades de um modelo.

Um problema de otimização é um problema cuja solução pode ser medida em termos de função custo (função objetivo), sendo que esta função custo pode atingir valores máximos e mínimos.

Em algumas situações os problemas de otimização podem ser reduzidos a uma sequência de problemas de decisões, tornando a solução bem mais simples.

Vários algoritmos implementam métodos para a solução de problemas de otimização. Um algoritmo é um processo computacional que tem um conjunto de entradas e saídas, tem um número finito de passos bem definidos para o seu desenvolvimento na resolução de um determinado problema.

Algoritmos podem ser descritos em linguagens naturais ou em linguagens de programação (software). O esforço computacional de um algoritmo é medido em termos de tempo de execução e consumo de memória e a complexidade é medida em termos de operações elementares que são repetidas no algoritmo.

Alguns problemas de decisão podem ser resolvidos com complexidade polinomial. Essa classe de problema são muitas das vezes denominados como P ou classe de problemas



tratáveis. Infelizmente essa classe cobre apenas uma pequena quantidade de problemas encontrados na síntese e otimização de circuitos digitais [01].

Muitos dos problemas de síntese e otimização para sistemas digitais são denominados NP-Hard (intratáveis).

Isto tem motivado a busca por algoritmos heurísticos, que são concebidos por terem complexidade polinomial com expoentes pequenos. Entretanto, esses algoritmos não garantem encontrar uma solução ótima global e muitas vezes obtêm soluções locais com relação a algum critério.

Os algoritmos estudados neste trabalho são de natureza muito geral e que podem ser aplicados para resolver problemas de decisão e de otimização.

Na síntese e na otimização digitais muitos problemas podem ser interpretados como a otimização de uma função linear sob restrições lineares.

Na forma mais geral, tem-se uma busca definida por um vetor desconhecido x . Assim, a formulação é denominada de programação linear (PL), como segue:

Min $c^T x$ de modo que

$$Ax \geq b$$

$$x \geq 0$$

onde $x \in \mathbb{R}^{n \times n}$; $c \in \mathbb{R}^n$; $b \in \mathbb{R}^m$ e $A \in \mathbb{R}^{m \times n}$; tem rank cheio.

Problemas lineares podem ser resolvidos de forma eficiente pelo método simplex [02].

Em caso particular, a formulação de programação linear pode ser explicada sob condições de que o não conhecido é limitado aos valores inteiros. Este problema é denominado de programação linear inteira (PLI) e é formulada como segue:

Min $c^T x$ de modo que

$$Ax \geq b$$

$$x \geq 0$$

$$x \in \mathbb{Z}^n$$

A natureza discreta do problema torna-o intratável. O limite interior sobre a solução de um PLI pode ser encontrado relaxando a restrição de inteiros. Isto é especialmente verdade, quando o PLI modela problemas de decisão e as variáveis são restritas a valores binários, isto é, 0 ou 1. Esses PLIs são denominados de PLI – 0/1 ou programa linear binário.

Programas lineares inteiros são resolvidos por uma variedade de algoritmos [03], incluindo o algoritmo Branch and Bound, descrito no Capítulo 3. A solução de um problema

linear obtida relaxando-se a restrição inteira é muitas vezes utilizada como um ponto de partida deste algoritmo.

1.2- Revisão Bibliográfica

A otimização lógica a dois níveis tem suas raízes há muitas décadas atrás e está baseada na teoria clássica sobre a álgebra de chaveamento. Vários textos tratam do assunto [04], [05], [06]. Os primeiros algoritmos para a minimização exata foram propostos por Quine [06] e por McCluskey [04]. Uma grande variedade de métodos foi proposta ao longo dos anos como, por exemplo, o método de Rudell e Sangiovanni [07] denominado Espresso-Exact, o método de Dagenais et ali [08] denominado de McBoole, o método de Coudert e Madre [09].

O método de Quine-McCluskey, por gerar uma solução mínima global, foi utilizado como referência para validar as implementações dos três outros algoritmos. Sua grande deficiência é o uso de uma grande quantidade de memórias na fase de geração dos implicantes primos, que torna o método proibitivo para funções com mais de 15 variáveis.

Este fato tem motivado a busca incessante por métodos eficientes para a geração dos implicantes primos da função. O algoritmo Expander [10] é um bom exemplo. Todos ou, quase todos os implicantes primos que não estejam no conjunto solução são eliminados, simplificando, desse modo, a busca pela solução mínima.

Com o aumento da complexidade dos circuitos digitais, a minimização lógica exata tornou-se impraticável até algumas décadas atrás. Entretanto, com o aumento da velocidade e da capacidade de memória das estações de trabalho, aliado ao desenvolvimento de eficientes métodos de cobertura para uma solução exata, revitalizou-se o interesse por essa área de pesquisa.

Atualmente, tem-se como um dos mais eficientes algoritmos de minimização de funções booleana o programa denominado BOOM – A Heuristic Boolean Minimizer [11], cuja principal característica é a combinação da geração dos implicantes primos em conjunto com a solução do problema de cobertura, o que leva a uma redução no número de implicantes primos gerados.

Utilizando o Mapa de Karnaugh, formulou-se o problema de cobertura (PLI) que representa um problema de programação linear inteira binária e, para a redução do sistema, empregou-se o método de duas fases, encontrando uma solução otimista (não inteira).

1.3 - Organização do Trabalho

Este trabalho trata da resolução do problema de cobertura de funções booleanas formulada como um problema de programação matemática. Implementaram-se três algoritmos de otimização descritos na literatura com o objetivo de avaliar o que melhor soluciona o problema de cobertura dos mintermos. Foram implementados o algoritmo Branch and Bound, o Plano de Corte de Gomory e o Dual Simplex. Utilizou-se como critério de eficiência a quantidade de memória utilizada, o tempo de execução, o número de iterações e a solução obtida.

Fez-se o estudo de casos onde dezenas de funções booleanas foram otimizadas através dos programas implementados (Branch2, Gomo2 e Dual). Obtiveram-se, dessa forma, parâmetros que foram tabulados e serviram de base para a comparação entre os algoritmos.

Neste trabalho, para a implementação dos algoritmos, utilizou-se o MATLAB versão 5.3, face às facilidades oferecidas e a rapidez na programação. Desta forma, os algoritmos puderam ser analisados de acordo com dois critérios: (1) a quantidade de soluções e (2) o esforço computacional necessário para obter a solução.

O problema tratado neste trabalho é a otimização de funções booleanas expressa na forma padrão, soma de produto, e tem como objetivo a atribuição de valores booleanos não variáveis para tornar a função verdade.

Para tornar este texto mais fácil de ser interpretado, apresentam-se, no próximo capítulo, os conceitos lógicos sobre a álgebra de Boole e os métodos clássicos para a minimização de funções booleanas e como o problema em coberturas dos mintermos pode ser formulado como um problema em programação matemática. Utilizaram-se o método de Plano de Corte e o método de Branch and Bound. Estes dois algoritmos foram comparados, verificando-se os custos e analisando-se os parâmetros: número de iterações, quantidade de memórias e tempo gasto.

No Capítulo 2, apresentam-se aspectos sobre a Álgebra de Boole [12]. o Mapa de Karnaugh e o método de Quine-McCluskey.

No Capítulo 3, apresentam-se de forma suficiente os algoritmos do Método Simplex [02] e o algoritmo do quadro Simplex Alternativo [13].

No Capítulo 4, apresentam-se programas computacionais denominados de Gomo2, Branch2 e Dual que foram empregados na resolução de problemas de cobertura de uma função booleana. Com a implementação dos programas, obtiveram-se parâmetros que foram úteis na análise dos custos, tempo de execução, número de iterações e quantidades de memória.

CAPÍTULO 2 – Método Para Otimização de Funções Booleanas

A Álgebra de Boole constitui uma importante ferramenta matemática para projeto de sistemas, ou para definir as expressões dos circuitos digitais de um sistema.

Na Álgebra de Boole trabalha-se com variáveis que podem assumir apenas dois valores (falso ou verdadeiro, zero ou um, etc), que são regidos por três postulados e alguns teoremas. Os postulados são negação, conjunção e disjunção, apresentados na Tabela 2.1 [15].

- Negação:** é conhecida também por Operação Complemento ou Inversão. Utiliza-se a operação negação para afirmar se uma proposição é falsa ou verdadeira, nos circuitos é representada pela porta NOT;
- Conjunção:** operação com função de verdade, indica o produto entre variáveis, geralmente representado pelo símbolo matemático " $\wedge$ " em um circuito é designado por uma porta AND;
- Disjunção:** operação de função verdade utilizada na união das variáveis, é representada pelo símbolo matemático " $\vee$ ", na designação de um circuito digital denotado pela porta OR.

NOT	AND	OR
-	$0 \cdot 0 = 0$	$0 + 0 = 0$
$1 = 0$	$0 \cdot 1 = 0$	$0 + 1 = 1$
-	$1 \cdot 0 = 0$	$1 + 0 = 1$
$0 = 1$	$1 \cdot 1 = 1$	$1 + 1 = 1$

Tabela 2.1: Postulados.

2.1- Expressões Booleanas

A expressão booleana da tabela verdade, apresentada na Tabela 2.2, obtém-se assumindo somente os valores iguais a 1 ou a 0 da saída, porém, não ambos. Estabelece-se que se uma variável de entrada ou de saída estiver ao nível lógico zero, designa-se a variável pelo seu complemento e se estiver ao nível lógico 1, pela forma normal, ou seja:

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

$$A = 1 \Rightarrow A$$

$$S = 0 \Rightarrow \bar{S}$$

$$S = 1 \Rightarrow S$$

Assim, a saída assume valores iguais a 1 nas seguintes combinações das entradas formando a expressão representada na Tabela 2.2. em cada combinação das variáveis A,B,C.

↓
(V)

	A	B	C	S
	0	0	0	0
$A'B'C$ (&)	0	0	1	1
ou	0	1	0	0
$A'BC$ (&)	0	1	1	1
ou	1	0	0	0
$AB'C$ (&)	1	0	1	1
	1	1	0	0
	1	1	1	0

Tabela 2.2: Expressão obtida a partir da tabela verdade $S = A'B'C + A'BC + AB'C$.

Observa-se que combinações de $A'B'C$, $A'BC$ e $AB'C$ na Tabela 2.2, em que as saídas são iguais a 1, são operações "AND". Os diversos estados iguais a 1 que a saída assume é uma operação OR.

Portanto:

$$S = A'B'C + A'BC + AB'C$$

Essa expressão é denominada função booleana e é a representação matemática dos circuitos digitais, portanto, é uma ferramenta muito útil no projeto de circuitos elétricos.

2.2 - Expressões Booleanas de Circuitos

Aos circuitos elétricos pode-se dar o mesmo tratamento matemático dado à tabela verdade, ou seja, à álgebra de Boole, pois acionamentos elétricos utilizados em um comando podem assumir duas condições (1 ou 0), portanto, perfeitamente viável dentro do convencional [15].

A Figura 2.1 apresenta um diagrama de um comando elétrico.

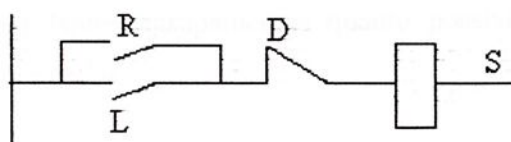


Figura 2.1: Comando Elétrico.

Observando este circuito, verifica-se que, se acionar L e não acionar D tem-se S energizado ou acionado (LD') ou acionado R e não acionando D, tem-se da mesma forma S acionado (RD'). Pode-se caracterizar, assim, que para a saída ou para que o comando execute, no caso S, e seja igual a 1, deve ocorrer a combinação descrita das variáveis de entrada, isto é, tem-se $S = 1$ se:

$$S = LD' + RD'$$

Pela expressão, pode-se observar que foi assumido contato acionado, como nível lógico 1 e não como nível lógico 0. A ocorrência de uma condição "AND" é assumida como uma operação de produto " $\cdot$ ", tratando-se do circuito de contato em série. O fato de existir mais de um caminho elétrico ou combinação de variáveis para S ser acionada (L acionado e D não "ou" R acionado e D não), caracteriza uma operação "OR" ou "+" entre as combinações, tratando-se no circuito de contatos em paralelos.

2.3 - Métodos Para Otimização de Funções Booleanas

2.3.1- Mapa de Karnaugh

O Mapa de Karnaugh [15] é a forma mais conhecida para a simplificação de expressões booleanas. O mapa contém as informações contidas na Tabela Verdade, porém, representada de uma forma particularmente conveniente. Cada mapa de n variáveis consiste de 2^n células, representando todas as possíveis combinações dessas variáveis. O Mapa de Karnaugh torna-se impraticável para funções de 6 ou mais variáveis.

2.3.1.1- Regras de Minimização

A simplificação consiste em se reunir células adjacentes que possuam (0 ou 1), formando termos de subcubos e obedecendo às seguintes regras:

- 1- O número de células reunido deve ser o maior possível, mesmo que para isso uma mesma célula deva pertencer a dois subcubos ou conjuntos diferentes;
- 2- O número de células reunidas em um subconjunto deve ser sempre potência de 2, ou seja, 1, 2, 4, 8, 16, ...;
- 3- Uma mesma célula pode pertencer a subcubos diferentes para satisfazer o item 1, mas não pode fazer agrupamentos desnecessários;
- 4- Deve-se formar tantos agrupamentos quanto possível até que todas as células tenham sido agrupadas.

A expressão da saída se obtém dos subcubos formados. Cada subcubo é uma função AND entre as variáveis que não mudam o nível lógico dentro do subcubo. Os diversos subcubos obtidos do mapa formam uma função OR [10].

Um produto que contém todas as n variáveis de uma função como fatores, complementada ou não, é chamado de mintermo. Sua característica particular é que assume o valor 1 para somente uma combinação de valores das variáveis. A soma de todos os mintermos derivados das linhas da tabela onde a função assume o valor 1, assumirá os valores 0 ou 1 da mesma forma que a função. Portanto, esta soma é realmente uma representação algébrica da função. Uma expressão deste tipo é chamada de soma canônica de produtos.

Uma soma que contém todas as n variáveis de uma função como fatores, complementadas ou não, é chamada de maxtermos. O produto de todos os maxtermos derivados das linhas da tabela onde a função assume o valor 0 assumirá os valores 0 ou 1 da mesma forma que a função. Portanto, este produto também é uma representação algébrica da função e é chamado de produto canônico de somas.

Como exemplo de aplicação do Mapa de Karnaugh na simplificação de funções booleanas, considere o mapa que minimize na função. Pode-se notar que os mintermos $AB'CD$, $ABCD$, $A'B'CD$ e $A'B'CD$ geraram o implicante primo CD , para termos que diferem em apenas uma literal podem ser unidos, eliminando-se a literal que se altera. Esse fato está baseado na propriedade da álgebra de Boole em que $AB + AB' = A$.

Exemplo 2.1

A função $f_1(A, B, C, D) = \sum m(3, 4, 5, 7, 8, 15) + d(2, 11, 14)$ apresentado na Figura 2.2.

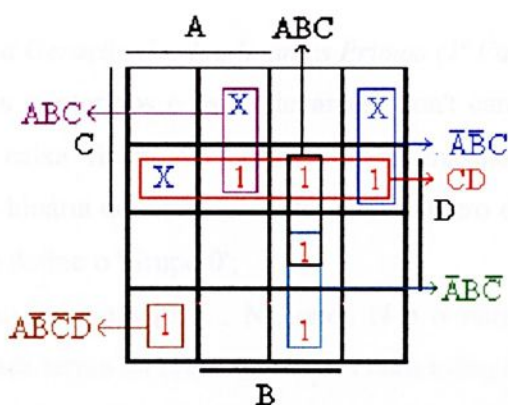


Figura 2.2: Subcubos do Mapa de Karnaugh.

Os subconjuntos são representados literalmente de acordo com as variáveis que os compõem, se a variável está presente escreve-se ela própria, caso contrário, representa-se pelo seu complemento e a saída é a união de todos os subconjuntos do mapa como mostra a equação, $S = CD + A'BC' + AB'C'D$ obtida através do Exemplo 2.1.

Um conjunto de 2^m células, cada uma adjacente a m células do conjunto é chamado de subcubo de ordem m , e diz-se que o subcubo cobre estas células. Todo subcubo cuja correspondência ("1" lógico) obriga a função a ter valor "1", ou seja, implica a função dada é um implicante da função. A cada implicante da função corresponde um produto de literais.

Um implicante que corresponde a um subcubo não contido em nenhum subcubo maior é chamado de implicante primo. Um implicante primo, além de implicar a função não pode implicar nenhum outro produto de menor número de literais (subcubo maior) que também implique a função dada.

Em uma tabela de implicantes primos cujos mintermos contém uma única marca, os implicantes primos que cobrem estes mintermos são denominados implicantes primos essenciais e aparecem obrigatoriamente na expressão mínima da função.

2.3.2 - O Método Quine-McCluskey

O método de Quine-McCluskey é usado para simplificar as funções booleanas no sentido de minimizar os custos dos circuitos digitais. A idéia fundamental do método é a aplicação repetida da propriedade $AB + AB' = A(B + B') = A$ (lei de absorção).

O método de Quine-McCluskey consiste em duas fases:

- a) Geração de todos os implicantes primos;
- b) Cobertura mínima dos mintermos.

2.3.2.1 - Algoritmo para a Geração dos Implicantes Primos (1ª Fase)

- 1- Separam-se os mintermos e os irrelevantes (don't care) em caixas, sendo que em uma mesma caixa ficam os termos com o mesmo número de dígitos '1' na representação binária do termo produto. Este número de '1s' é o índice da caixa e essa operação define o 'Grupo 0';
- 2- Para todo i no intervalo $[0, ..., N]$, onde N é o número de variáveis da função, compara-se cada termo da caixa de índice i com todos os termos da caixa de índice $i + 1$. Os termos são combinados de acordo com a propriedade $AB + AB' = A(B + B') = A$, isto é, dois termos são combináveis se pertencem a caixas adjacentes e diferem

apenas em um dígito na sua representação binária. Além disso, o termo da caixa de índice superior ($i + 1$) deve ser sempre maior que o termo da caixa de índice inferior (i). Marcam-se todos os termos que foram combinados pelo menos uma vez. Esta operação gera um novo "Grupo".

- 3- Combina-se da mesma forma os termos gerados no passo anterior;
- 4- O processo descrito no passo três continua até que não haja mais combinações possíveis. Os termos não marcados constituem no conjunto de todos os implicantes primos da função. Se repetirem implicantes primos não marcados, utiliza-se apenas um implicante primo de cada termo que repete.

Exemplo 2.2

Como exemplo da aplicação do método de Quine-McCluskey considere a função representada por $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$.

A primeira fase do método consiste na geração dos implicantes primos, cujas fases estão apresentadas nas Tabelas 2.3, 2.4, 2.5.

Geração dos implicantes primos.

cx	dec	bin	marca
1	2	0010	✓
2	6	0110	✓
2	10	1010	✓
3	7	0111	✓
3	11	1011	✓
3	14	1110	✓
4	15	1111	✓

Tabela 2.3: Grupo 0.

cx	dec	bin	marca
1	2 - 6	0X10	✓
1	2 - 10	X010	✓
2	6 - 7	011X	✓
2	6 - 14	X110	✓
3	10 - 11	101X	✓
3	10 - 14	1X10	✓
3	7 - 15	X111	✓
3	11 - 15	1X11	✓
3	14 - 15	111X	✓

Tabela 2.4: Grupo 1.

cx	dec	bin	marca
1	2 - 6 - 10 - 14	XX10	
2	6 - 7 - 14 - 15	X11X	
2	10 - 11 - 14 - 15	1X1X	

Tabela 2.5: Grupo 2.

Os termos não marcados constituem-se conjuntos dos implicantes primos. Os implicantes primos obtidos foram:

$$\text{Implicantes primos} \begin{cases} XX10 \\ X11X \\ 1X1X \end{cases}$$

Neste estágio a função pode ser representada por $f_2(A, B, C, D) = CD' + BC + AC$ ou $f_2(A, B, C, D) = XX10 + X11X + 1X1X$.

A primeira fase do método de Quine-McCluskey corresponde à geração de todos os implicantes primos da função. A Figura 2.3 apresenta no mapa de Karnaugh os implicantes primos gerados.

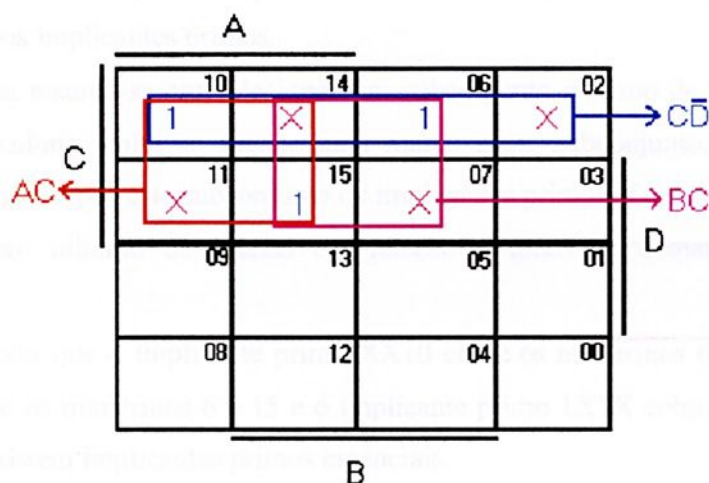


Figura 2.3: Implicantes primos (AC, $\bar{C}D$ e BC) gerados na 1ª fase do método de Quine-McCluskey.

Pode-se notar pelo mapa da Figura 2.3 que não existem implicantes primos essenciais, pois todos os mintermos são cobertos por dois implicantes primos.

A 2ª fase do método Quine-McCluskey é utilizada para a seleção dos implicantes primos que cobrem os mintermos das funções, utilizando-se a chamada tabela de cobertura dos mintermos.

2.3.2.2 - Tabela de Cobertura dos Mintermos (2ª Fase)

A Tabela 2.6, apresenta a cobertura dos mintermos para a função $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$ formada com base no conjunto de implicantes primos que foram gerados na primeira fase do método.

Implicantes	Mintermos		
	6	10	15
XX10	X	X	
X11X	X		X
1X1X		x	X

Tabela 2.6: Tabela de cobertura dos Implicantes Primos.

Após a obtenção do conjunto de todos os implicantes primos da função, monta-se uma tabela na qual as linhas são os implicantes primos e as colunas são os mintermos da função. Os

estados irrelevantes (don't care) não aparecem nessa tabela, uma vez que não precisam ser cobertos pela expressão mínima da função. A tabela é montada colocando-se uma marca (x) nas intersecções entre os implicantes primos e os mintermos, indicando quais mintermos são cobertos por um dos implicantes primos.

O problema resume-se em selecionar um subconjunto mínimo de linhas da tabela de modo que cada coluna tenha ao menos uma marca neste subconjunto, ou seja, todos os mintermos são cobertos por este subconjunto de implicantes primos. Tal subconjunto deve ser o que possui número mínimo de literais em relação a todos os demais que possam ser selecionados.

Pode-se notar que o implicante primo XX10 cobre os mintermos 6 e 10, o implicante primo X11X cobre os mintermos 6 e 15 e o implicante primo 1X1X cobre os mintermos 10 e 15, ou seja, não existem implicantes primos essenciais.

Esta situação é denominada de mapa cíclico. Um dos métodos empregados é o método de Peterson [03], que é empregado para reduzir a tabela de implicantes primos (tabela de cobertura dos mintermos).

No método de Peterson, a lista de todos possíveis conjuntos irredundantes de implicantes primos que cobrem os mintermos podem ser obtidos da seguinte forma, as letras representam os implicantes primos e são interpretadas como variáveis booleanas, que assumem valor "1" se o implicante primo é selecionado e "0" se não é selecionado.

Rotulou-se XX10 como x, X11X como y e 1X1X como w.

Pela Tabela 2.6 pode-se verificar o x ou y deve ser selecionado para cobrir o mintermo 6. Desta forma pode-se escrever:

$$x + y = 1.$$

De forma similar, x ou w pode ser selecionado para cobrir o mintermo 10. Então tem-se:

$$(x + y). (x + w) = 1.$$

Através desse raciocínio, forma-se a expressão completa:

$$(x + y). (x + w). (y + w) = 1.$$

Aplicando-se a lei distributiva tem-se:

$$(x + xw + xy + yw)(y + w) = 1$$

$$xy + xw + xyw + xw + xy + xyw + yw + yw = xy + xw + xyw + yw = 1$$

$$xy + xw + xyw + yw = 1.$$

Para satisfazer a equação, pelo menos um dos 4 produtos deve ser 1.

O custo total é menor se $xy = 1$ ou se $xw = 1$ ou se $yw = 1$. Desta forma, têm-se três possíveis soluções de mesmo custo, ou seja.

Solução 1: $XX10 + X11X$;

Solução 2: $X11X + 1X1X$;

Solução 3: $XX10 + 1X1X$.

Todas as três soluções têm custo igual a 6.

Neste trabalho o problema de mapa cíclico foi resolvido através de um problema de programação matemática.

Com auxílio da tabela de cobertura dos mintermos, representada na Tabela 2.6, transforma-se o problema de cobertura em um problema de programação linear inteira binária $\{0, 1\}$.

Tem-se o PL:

Min $3(XX10 + X11X + 1X1X)$

s.a.

$$XX10 + X11X \geq 1$$

$$XX10 + 1X1X \geq 1$$

$$X11X + 1X1X \geq 1$$

$$XX10, X1X1, X11X \in \{0, 1\}.$$

São apresentadas a seguir algumas tabelas de cobertura dos mintermos cuja obtenção da solução mínima foi obtida através de uma formulação do problema como um problema de programação linear inteira 0 e 1.

O objetivo é mostrar para o leitor como formular o problema de cobertura como um problema de programação linear inteira.

Problemas de cobertura como estes estão resolvido no Capítulo 3 a seguir, empregando-se o método Simplex [02] e o Dual simplex [16], pois trata-se de problemas de programação linear inteira binário. No caso de uma solução otimista, ou seja, um resultado fracionário, emprega-se o método: Branch and Bound ou o Plano de Corte de Gomory.

Verificou-se no programa Dual Simplex que sempre obteve uma solução ótima, não havendo, portanto, a necessidade do emprego do Branch and Bound ou Plano de Corte.

CAPÍTULO 3 – Problemas de Cobertura

No Capítulo 2, foram descritos dois métodos de minimização de funções booleanas. Foi dada ênfase no método de Quine-McCluskey, pois a segunda fase do método, a fase de cobertura dos mintermos, corresponde a um problema de programação linear inteira 0 e 1.

Para se formular a cobertura dos mintermos como um PLI é necessária a obtenção de todos os implicantes primos para garantir uma solução de custo mínimo. A fase de formação de todos os implicantes primos é muito trabalhosa computacionalmente, pois o número de implicantes primos cresce exponencialmente com o aumento das variáveis [13].

Pesquisas procuraram não gerar todos os implicantes primos, simplificando, dessa forma, a formulação do problema de cobertura [11]; entretanto, não se pode garantir que a solução obtida é a de custo mínimo global.

Neste trabalho, não se teve a preocupação com a formulação do problema de cobertura dos mintermos mas sim na solução do PLI.

Toda cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1 [13]. Estes problemas foram solucionados empregando o método Simplex [02] e método Dual Simplex [16].

O método simplex, em alguns casos, em se tratando de problemas de programação linear, encontram-se soluções otimizadas, ou seja, fracionárias, e para tornar a solução inteira, utiliza-se um dos seguintes métodos: O método Plano de Corte e/ou método de Branch and Bound.

3.1- Método Simplex

Dado um problema de programação linear inteira, no caso, de cobertura mínima de uma função booleana, pode-se encontrar a solução ótima do sistema usando o método Simplex ou do método Simplex Alternativo [13].

3.2 - O Algoritmo do Método Simplex

Pode-se encontrar a solução ótima do sistema, ou seja, um problema de programação linear inteiro usando-se o método simplex conforme Tabela 3.1.



	Z	X_B	X_N	RHS
Z	1	0	$C_B B^{-1} N - C_N$	$C_B B^{-1} b$
X_B	0	I	$B^{-1} N$	$B^{-1} b$

Tabela 3.1: Tabela do quadro simplex.

Neste quadro, existem todas as informações para identificar se a solução factível básica (SFB) é ótima ou ilimitada.

Onde: $Z \rightarrow$ função objetivo;

$X_B \rightarrow$ variáveis básicas;

$X_N \rightarrow$ variáveis não básicas;

$C_B \rightarrow$ coeficientes da função objetivo;

$C_N \rightarrow$ coeficientes das restrições;

$N \rightarrow$ matriz formada pelos coeficientes das restrições;

$B^{-1} \rightarrow$ inversa da matriz da base;

$b \rightarrow$ termos independentes;

$I \rightarrow$ matriz identidade

3.2.1 - Quadro Simplex Alternativo

O quadro simplex alternativo é de tamanho reduzido quando comparado com o quadro simplex empregado por Bazaraa [02]. Foram eliminadas do quadro alternativo as colunas das variáveis básicas que formam uma matriz identidade, assim, facilitando o pivoteamento. A Tabela 3.2 apresenta o quadro simplex alternativo.

	X_N	RHS
X_0	$C_B B^{-1} N - C_N$	$C_B B^{-1} b$
X_B	$B^{-1} N$	$B^{-1} b$

Tabela 3.2: Tabela do quadro simplex alternativo.

Onde: x_0 é a função objetivo.

Pelo fato do quadro simplex alternativo não trabalhar com as variáveis básicas, seus custos operacionais são menores que a do quadro simplex.

Exemplo 3.1

A cobertura da função booleana $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$, apresentada no Mapa de Karnaugh da Figura 3.1, a seguir pode ser formulado como um problema de programação linear inteira 0 ou 1.

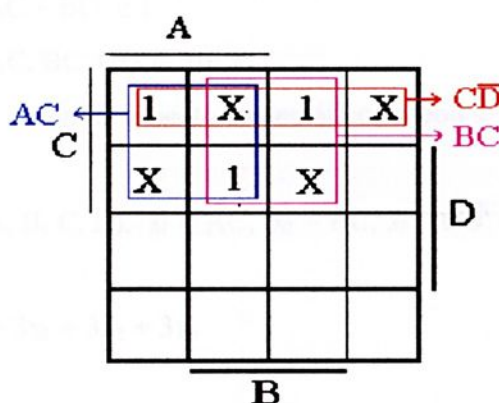


Figura 3.1: Formulação da função $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$.

Sabe-se que toda cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1 [17], [18]. Na resolução do Exemplo 3.1, inicialmente aplica-se o método simplex alternativo o qual resulta numa solução otimista. Para encontrar a solução de custo mínimo global, é necessário utilizar outros métodos como, por exemplo, o Plano de Corte, ou o Branch and Bound.

Dado f_2 , através da aplicação da segunda fase do método de Quine-McCluskey, encontraam-se os seguintes implicants primos: $1X1X$, $X11X$ e $XX10$, a função equivalente a f_2 pode ser representada pela soma de seus implicants primos, ou seja, $f_2 = AC + BC + CD'$ que corresponde no circuito digital apresentado na Figura 3.2.

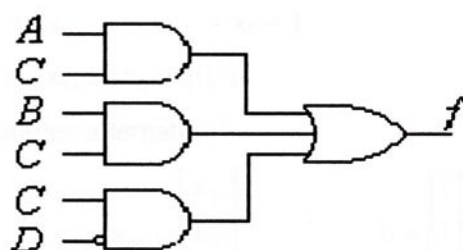


Figura 3.2: Apresentação do circuito equivalente lógico da função f_2 .

O problema de cobertura da função booleana $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$ é dado por:

$$\text{Min } f_2(A, B, C, D) = 3AC + 3BC + 3CD'$$

s.a.

$$BC + CD' \geq 1$$

$$AC + CD' \geq 1$$

$$AC + BC \geq 1$$

$$AC, BC, CD' \in \{0, 1\}$$

Para facilitar o desenvolvimento matemático trocou-se a nomenclatura das variáveis, como segue::

$$x_0 = f_2(A, B, C, D), \quad x_1 = AC, \quad x_2 = BC, \quad x_3 = CD'$$

Tem-se:

$$\text{Min } x_0 = 3x_1 + 3x_2 + 3x_3$$

s.a.

$$x_1 + x_2 \geq 1$$

$$x_1 + x_3 \geq 1$$

$$x_2 + x_3 \geq 1$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

Transformando o sistema na forma padrão, e introduzindo-se as variáveis de folgas x_4, x_5, x_6

Obteve-se:

$$\text{Min } x_0 = 3x_1 + 3x_2 + 3x_3$$

s.a.

$$x_1 + x_2 - x_4 = 1$$

$$x_1 + x_3 - x_5 = 1$$

$$x_2 + x_3 - x_6 = 1$$

$$x_1, x_2, x_3, x_4, x_5, x_6 \in \{0, 1\}$$

Monta-se o quadro simplex alternativo:

$$x_B = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad x_N = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$B = -I = B^{-1} \quad N = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$C_B B^{-1} b = 0$$

$$B^{-1} = -b$$

$$B^{-1} N = -N$$

$$-b = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$-N = \begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

Para resolver este sistema emprega-se o método das duas fases da programação linear.

Portanto, acrescentam-se as variáveis artificiais x_7, x_8, x_9 .

Assim, tem-se o seguinte:

$$\text{Min } x_0 = 3x_1 + 3x_2 + 3x_3$$

s.a.

$$x_1 + x_2 - x_4 + x_7 = 1$$

$$x_1 + x_3 - x_5 + x_8 = 1$$

$$x_2 + x_3 - x_6 + x_9 = 1$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9 \in \{0, 1\}$$

Formulação do quadro inicial:

$$A = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x_B = \begin{bmatrix} x_7 \\ x_8 \\ x_9 \end{bmatrix}$$

$$x_N = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix}$$

Custo relativo da função objetivo I:

$$\bar{C}'_B = C'_B B^{-1} N - C'_N$$

$$C'_B = [1 \ 1 \ 1]$$

$$C'_N = [0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$\bar{C}'_B = [1 \ 1 \ 1] \cdot \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \end{bmatrix} - [0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$\bar{C}'_B = [2 \ 2 \ 2 \ -1 \ -1 \ -1]$$

Custo relativo da função objetivo II:

$$\bar{C}_B = C_B B^{-1} N - C_N$$

$$C_B = [0 \ 0 \ 0]$$

$$C_N = [3 \ 3 \ 3 \ 0 \ 0 \ 0]$$

$$\bar{C}_B = [0 \ 0 \ 0] \cdot \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 & 0 & -1 \end{bmatrix} - [3 \ 3 \ 3 \ 0 \ 0 \ 0]$$

$$\bar{C}_B = [-3 \ -3 \ -3 \ 0 \ 0 \ 0]$$

Função objetivo da fase I:

$$\bar{C}'_B = C'_B B^{-1} b = C'_B b \Rightarrow \bar{C}'_B = [1 \ 1 \ 1] \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = 3$$

Função objetivo da fase II:

$$\bar{C}_B = C_B B^{-1} b = 0$$

$$\text{Variáveis não básicas: } B^{-1} N = N \quad \text{e} \quad \text{RHS } B^{-1} b = b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Com estes dados, monta-se o quadro escolhendo como pivô a variável que entra e a que sai da base, em seguida faz-se o pivoteamento como apresentam as Tabela 3.3 à Tabela 3.8. Este processo é feito para todas as iterações necessárias para encontrar a solução, se a solução encontrada não for inteira, então, continuando o processo emprega-se outro método para chegar à solução ótima.

↓

	RHS	-x ₁	-x ₂	-x ₃	-x ₄	-x ₅	-x ₆
z ₀	3	2	2	2	-1	-1	-1
x ₀	0	-3	-3	-3	0	0	0
x ₇	1	1	1	1	-1	0	0
x ₈	1	1	0	0	0	-1	0
← x ₉	1	0	1	1	0	0	-1

Tabela 3.3: Entra a variável x₃ e sai a variável x₉.

Após encontrar o elemento pivô em cada tabela, os cálculos serão realizados com todas as colunas da tabela em relação à coluna do elemento pivô. Como exemplo, os cálculos da primeira coluna da Tabela 3.3 serão efetuados, e as demais colunas seguem o mesmo processo, como a seguir:

- (o primeiro elemento da coluna RHS) menos (o primeiro elemento da coluna onde se encontra o elemento pivô) vezes (o elemento da coluna RHS da linha do pivô) divididas (pelo elemento pivô).
- cálculo do primeiro elemento da coluna RHS $3 - (2 \times 1)/1 = 1$;
- cálculo do segundo elemento da coluna RHS $0 - [(-3) \times 1]/1 = 3$;
- cálculo do terceiro elemento da coluna RHS $1 - (1 \times 1)/1 = 0$;
- cálculo do quarto elemento da coluna RHS $1 - (0 \times 1)/1 = 1$;
- o quinto elemento pertence à linha do pivô, ele é calculado pela divisão de todos os elementos da linha do pivô, pelo elemento pivô.

	RHS	$-x_1$	$-x_2$	$-x_9$	$-x_4$	$-x_5$	$-x_6$
z_0	1	2	0		-1	-1	1
x_0	3	-3	0		0	0	-3
x_7	0	1	0		-1	0	1
x_8	1	1	0		0	-1	0
x_3	1	0	1		0	0	-1

Tabela 3.4: Entra a variável x_1 e sai a variável x_8 .

	RHS	$-x_8$	$-x_2$	$-x_9$	$-x_4$	$-x_5$	$-x_6$
z_0	-1		0		-1	1	1
x_0	6		0		0	-3	-3
x_7	-1		0		-1	1	1
x_1	1		0		0	1	0
x_3	1		1		0	-1	-1

Tabela 3.5: Apresenta a linha z_0 linearmente dependente da linha x_7 .

Multiplicando-se a linha x_7 por (-1) e o resultado somado à linha onde está localizado z_0 , a linha da função objetivo auxiliar, será eliminada.

	RHS	$-x_7$	$-x_2$	$-x_9$	$-x_4$	$-x_5$	$-x_6$
z_0	0		0		0	0	0
x_0	6		3		0	0	-3
x_1	1		1		-1	0	0
x_8	-1		-2		0	-1	1
x_3	1		1		0	0	-1

Tabela 3.6: Apresenta a função objetivo z_0 é nula.

Eliminando-se a linha da função objetivo auxiliar z_0 , tem-se:

	RHS	$-x_7$	$-x_2$	$-x_9$	$-x_4$	$-x_5$	$-x_6$
x_0	6		3		0	0	-3
x_1	1		1		-1	0	0
x_8	-1		-2		0	-1	1
x_3	1		1		0	0	-1

Tabela 3.7: Apresenta a eliminação da função objetivo z_0 .

Como na Tabela 3.7 o quadro não é ótimo, faz-se novamente o pivoteamento e assim, obtém-se a Tabela 3.8 que é a solução otimista do problema.

	RHS	$-x_7$	$-x_8$	$-x_9$	$-x_4$	$-x_5$	$-x_6$
x_0	$9/2$				$-3/2$	$-3/2$	$-3/2$
x_1	$1/2$				$-1/2$	$-1/2$	$1/2$
x_2	$1/2$				$-1/2$	$1/2$	$-1/2$
x_3	$1/2$				$1/2$	$-1/2$	$1/2$

Tabela 3.8: Quadro da solução otimista.

$$\text{Solução otimista do problema original: } \left\{ \begin{array}{l} \text{Função Objetivo} = x_0 = 9/2 \\ x_1 = 1/2 \\ x_2 = 1/2 \\ x_3 = 1/2 \end{array} \right.$$

Pode-se verificar que a solução otimista não é inteira, tornando-se infactível para o problema considerado. Assim, será necessário utilizar o método Plano de Corte ou o método Branch and Bound para encontrar a solução factível do Exemplo 3.1.

3.3 - Método de Plano de Corte

O Exemplo 3.1 foi resolvido pelo método simplex do quadro alternativo, porém encontrou-se um quadro otimista, cuja solução não satisfaz, pois se pretende encontrar uma solução inteira binária. Para encontrar esta solução 0 e 1, parte-se do quadro inicial otimista empregando-se o Método de Plano de Corte fracionário de Gomory.

3.3.1- Corte fundamental do Gomory

$$\sum_{j \in R} ([h]y_{ij} - [hy_{ij}])x_j \geq [h]y_{i0} - [hy_{i0}] \quad (1)$$

Usando o corte fundamental de várias maneiras podem ser obtidos diferentes tipos de cortes e diferentes tipos de algoritmos de plano de corte.

3.3.2 - Método de Corte Fracionário de Gomory

Este método é obtido adotando-se $h = 1$ em (1). Assim, com $h = 1$ tem-se:

$$\sum_{j \in R} (y_{ij} - [y_{ij}])x_j \geq y_{i0} - [y_{i0}]$$

Usando a notação: $y_{ij} = [y_{ij}] + f_{ij}$

(1) Considere a seguinte forma:

$$\sum_{j \in R} f_{ij}x_j \geq f_{i0} \quad i = 1, 2, 3 \dots, m \quad (2)$$

Multiplicando-se por -1 , obtêm-se: $-\sum_{j \in R} f_{ij}x_j \leq -f_{i0}$ Assim, introduz-se a variável de folga S .

$$\text{Tem-se : } -\sum_{j \in R} f_{ij}x_j + S \leq -f_{i0} \quad \text{ou} \quad S = -f_{i0} + \sum_{j \in R} f_{i0}x_j$$

3.3.3 - Algoritmo Plano de Corte de Gomory

3.3.3.1- Algoritmo Básico

Passo 1: Fase inicial

Resolver o PL correspondente do PLI e ir ao passo 2.

Passo 2: Teste da otimalidade:

Se a solução for inteira, pare, será encontrada a solução ótima. Caso contrário ir ao passo 3.

Passo 3: Gerar o corte e fazer o pivoteamento:

Escolher uma fila r para gerar o corte tal que $f_{r0} > 0$, gerar o corte e ainda usar o corte gerado como a última linha do quadro simplex. Reotimizar o quadro simplex usando um algoritmo Dual Simplex e depois voltar ao passo 2.

Observação: Existem vários critérios para escolher a fila r com $f_{r0} > 0$ para gerar o corte. Pode-se usar das seguintes regras heurísticas (critérios):

a) Escolher a primeira fila (linha) com $f_{r0} > 0$;

b) Escolher a fila (linha) com maior parte fracionária da coluna RHS: $F_{r0} = \max\{f_{r0}\}$;

c) Usar o critério: $\frac{f_{r0}}{\sum_{j \in R} f_{rj}} = \max \left\{ \frac{f_{r0}}{\sum_{j \in R} f_{ij}} \right\}$;

d) Usar o critério: $\frac{f_{r0}}{f_{rk}} = \max \left\{ \frac{f_{io}}{f_{ik}} \right\}$ para algum $k \in R$

Assim, pode-se dar seqüência na resolução da função f_3 para obter a solução ótima, ou seja, a solução binária 0 e 1.

$$\text{Min } x_0 = 3x_1 + 3x_2 + 3x_3$$

s.a.

$$x_1 + x_2 \geq 1$$

$$x_1 + x_3 \geq 1$$

$$x_2 + x_3 \geq 1$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

Passo 1: Resolver o PL

Resolveu-se o PL usando o método primal simplex, verificou-se que o quadro ótimo do PL não apresenta solução inteira.

Passo 2: Teste da otimalidade

A solução não é inteira, foi encontrada uma solução ótima. Neste caso ir ao passo 3.

Passo 3: Usa-se o critério (b) para gerar o corte de Gomory. A fila que gera o corte é aquela que tem o maior $f_{r0} \rightarrow f_{r0}$.

Fila do x_1 é escolhida $r = 1$.

Observação: No caso do Exemplo 3.1, pode-se escolher qualquer uma das linhas, pois todos os valores da coluna do RHS são iguais.

Tem-se

$$\frac{1}{2}x_4 + \frac{1}{2}x_5 + \frac{1}{2}x_6 \geq \frac{1}{2}$$

Multiplicando esta restrição por (-1) .

$$-\frac{1}{2}x_4 - \frac{1}{2}x_5 - \frac{1}{2}x_6 \leq -\frac{1}{2}$$

Introduz a variável de folga S_1 .

$$-\frac{1}{2}x_4 - \frac{1}{2}x_5 - \frac{1}{2}x_6 + S_1 = -\frac{1}{2}$$

Isolando-se S_1 , tem-se:

$$S_1 = -\frac{1}{2} + \frac{1}{2}x_4 + \frac{1}{2}x_5 + \frac{1}{2}x_6$$

Coloca-se esta restrição na última linha do quadro otimista inicial apresentado na Tabela 3.9.

	RHS	- x_4	- x_5	- x_6
x_0	9/2	-3/2	-3/2	-3/2
x_1	1/2	-1/2	-1/2	1/2
x_2	1/2	-1/2	1/2	-1/2
x_3	1/2	1/2	-1/2	1/2
S_1	-1/2	-1/2	-1/2	-1/2

Tabela 3.9: Quadro otimista com acréscimo da restrição de corte.

Escolhe-se a variável que entra e a que sai da base e faz-se o pivoteamento; este processo se repete em quantas iterações forem necessárias. Assim, encontra-se a solução ótima do problema de cobertura da função booleana $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$. A Tabela 3.10 apresenta o quadro ótimo.

	RHS	- S ₄	- x ₅	- x ₆
x ₀	6	-3	0	0
x ₁	1	-1	0	1
x ₂	1	-1	1	0
x ₃	0	1	-1	-1
x ₄	1	-2	1	1

Tabela 3.10: Quadro ótimo do método Plano de Corte.

$$\text{Solução ótima} \left\{ \begin{array}{l} \text{Função Objetivo} = x_0 = 6 \\ x_1 = 1 \\ x_2 = 1 \\ x_3 = 0 \end{array} \right.$$

Como $x_1 = AC$, $x_2 = BC$, $x_3 = CD'$.

Tem-se: $AC = 1$, $BC = 1$, $CD' = 0$ e a função objetivo F.O. = 6.

Uma outra forma de se obter uma solução factível, dada uma solução otimista, é através do método de Branch and Bound

3.4 - Método de Branch and Bound

Branch and Bound (B & B) é um método geral de pesquisa e que neste trabalho foi empregado na minimização de uma função $f(x, \text{função})$, onde x é restrito a alguma possível região.

Seja o problema de programação linear.

$$\text{Min } z(x) = \sum_{j=1}^n c_j x_j$$

s.a.

$$\sum_{j=1}^n a_{ij} x_j \leq b_i \quad i = 1, 2, 3, \dots, m \quad (3)$$

$$x_j \geq 0 \quad j = 1, 2, 3, \dots, n$$

$$x_j \text{ inteiro para } j = 1, 2, \dots, I \quad (I \leq n)$$

O problema (3) pode ser resolvido usando o algoritmo de Branch and Bound. A idéia básica da metodologia de Branch and Bound é a estratégia de dividir e conquistar. A pesquisa de B&B de uma forma geral enumera esquemas para resolver problemas e obter solução direta.

Algoritmos que não utilizam este artifício têm um desempenho inferior. Este fato é baseado que, em geral, somente um pequeno número das soluções possíveis necessitam ser enumeradas. As possíveis restantes são eliminadas pelas aplicações de limites que refletem as restrições. Ela pode também ser vista como pesquisa estruturada em espaço contendo todas soluções possíveis no sentido de que, no pior caso, todas soluções possíveis devem ser enumeradas.

O nome "Branch and Bound" surge de duas operações básicas:

- 1- Branching: consiste em dividir as soluções possíveis em subconjuntos;
- 2- Bounding: consiste em eliminar estes subconjuntos conhecidos que não contêm uma solução.

Como o problema original é complexo para resolver diretamente, então, ele é dividido em subproblemas menores, sucessivamente até que esses subproblemas sejam completamente resolvidos.

A divisão (branching) é realizada partindo da região factível de um problema em sub-regiões menores. A conquista ou sondagem (fothoming) é realizada encontrando um limitante (bounding) da melhor solução (solução ótima) de um subproblema e verificando que nesse subproblema não pode existir a solução ótima global.

Assim, existem três conceitos básicos no algoritmo de Branch and Bound:

Branching ----- Dividindo, separando;

Bounding ----- Limitando;

Fothoming ----- Sondando (Eliminando).

3.4.1- Branching

Quando a solução de um problema de programação linear correspondente não é inteira, então, deve-se escolher uma variável para programação *branching*, esta variável possui um valor fracionário x_r . O problema corrente é dividido em outros dois subproblemas e, estes são formados adicionando as seguintes restrições:

- $x_r \leq [x_r^*]$
- $x_r \geq [x_r^*] + 1$

Onde, $[x_r^*]$ é o próximo inteiro menor ou igual a x_r e $[x_r^*] + 1$ é o próximo inteiro maior ou igual a x_r .

Este procedimento se repete para cada x_r encontrado no problema original.

Exemplo 3.2: Se P_k é subproblema escolhido a variável x_z para *branching* com valor corrente igual a $x_z^* = 7/2$ então se divide P_k em dois subproblemas:

$$P_{k+1} \left\{ \begin{array}{l} P_k \\ + \\ x_z \leq 3 \end{array} \right. \quad e \quad P_{k+2} \left\{ \begin{array}{l} P_k \\ + \\ x_z \geq 4 \end{array} \right.$$

3.4.2 - Bounding

Para um subproblema necessita-se obter a introdução de um limitante tão bom quanto possível da melhor solução inteira que existe em um subproblema. A forma mais prática de obter um limitante é resolvendo um problema relaxado, isto é, o PL correspondente do subproblema. Assim, se a resolução de um PL correspondente produz uma função objetivo de δ , então, pode-se afirmar que a melhor solução inteira que existe nesse subproblema, que será chamado de z , deve satisfazer: $z \geq \delta$ (problema de minimização).

3.4.3 - Fathoming

Um problema pode ser conquistado (sondado) e eliminado para futuras avaliações, no caso em que as soluções do PL correspondentes forem inteiras, caso em que o limitante do subproblema foi maior que a incumbente ou caso em que a solução do PL é infactível.

3.4.4 - Testes de Sondagens

Um subproblema pode ser sondado se satisfaz um dos seguintes testes de sondagens:

Teste 1: Se o limitante δ satisfaz $\delta \geq z^*$ em que z^* é a incumbente (problema de minimização).

Teste 2: Se o PL correspondente é infactível.

Teste 3: Se a solução do PL correspondente é inteira. Neste caso se a solução inteira é melhor que a incumbente então, deve-se atualizar a incumbente e aplicar novamente o teste 1.

Observações:

- 1- No algoritmo é encontrado um limitante resolvendo o PL relaxado. Uma outra possibilidade é usar uma metodologia chamada relaxação lagrangeana;
- 2- Para encontrar soluções alternativas ótimas deve-se fazer as seguintes modificações:
 - 2.1- Trocar o limitante maior ou igual a z^* por limitante maior que z^* ;
 - 2.2- Cada vez que é encontrada uma solução inteira com maior ou igual a incumbente corrente deve-se armazenar todas essas incumbentes;

2.3- Se aparecer uma incumbente de melhor qualidade então se deve atualizar o armazenador do incumbente;

3- Um algoritmo de Branch and Bound eficiente deve usar um algoritmo dual simplex para resolver o PL relaxado correspondente a cada subproblema.

Para resolver o Exemplo 3.1 pelo método de Branch and Bound parte-se do quadro otimista do problema original (PO) resolvido pelo método simplex.

Usando $x_1 = 1/2$ pode-se separar o problema PO em dois outros subproblemas:

$$P_1 = \left\{ \begin{array}{l} \text{Min } x_0 = 3x_1 + 3x_2 + 3x_3 \\ \text{s.a.} \\ x_1 + x_2 \geq 1 \\ x_1 + x_3 \geq 1 \\ x_2 + x_3 \geq 1 \\ x_1 \leq 0 \\ x_1, x_2, x_3 \in \{0, 1\} \end{array} \right. \quad e \quad P_2 = \left\{ \begin{array}{l} \text{Min } x_0 = 3x_1 + 3x_2 + 3x_3 \\ \text{s.a.} \\ x_1 + x_2 \geq 1 \\ x_1 + x_3 \geq 1 \\ x_2 + x_3 \geq 1 \\ x_1 \geq 1 \\ x_1, x_2, x_3 \in \{0, 1\} \end{array} \right.$$

Resolução do subproblema P_1 :

Monta-se o quadro simplex alternativo, incluindo-se uma nova restrição: $x_1 \leq 0$.

Nesta nova restrição acrescenta-se a variável de folga $x_1 + x_7 = 0$.

$$x_B = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad x_N = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad a_B = [1 \ 0 \ 0] \quad a_N = [0 \ 0 \ 0]$$

$$\bar{C}_B = a_N - a_B B^{-1} N$$

$$\bar{C}_B = [0 \ 0 \ 0] - [1 \ 0 \ 0] \cdot \begin{bmatrix} -1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 \end{bmatrix} \Rightarrow \bar{C}_B = [1/2 \ 1/2 \ -1/2]$$

$$\bar{b} = b_4 - a_b B^{-1} b \Rightarrow \bar{b} = 0 - [1 \ 0 \ 0] \cdot \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix} \therefore \bar{b} = -1/2$$

Com obtenção dos dados forma-se uma nova restrição a ser acrescentada no quadro otimista apresentada na Tabela 3.11.

	RHS	- x ₄	- x ₅	- x ₆
x ₀	9/2	-3/2	-3/2	-3/2
x ₁	1/2	-1/2	-1/2	1/2
x ₂	1/2	-1/2	1/2	-1/2
x ₃	1/2	-1/2	-1/2	1/2
x ₇	-1/2	1/2	1/2	-1/2

Tabela 3.11: Quadro otimista com acréscimo de uma nova restrição.

Escolhe-se a variável que entra e a variável que sai da base para encontrar o pivô, faz-se o pivoteamento, este processo se repete em quantas iterações forem necessárias. Assim, encontra-se a solução ótima do problema de cobertura mínima da função booleana $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$ apresentada na Tabela 3.12.

	RHS	- x ₄	- x ₅	- x ₇
x ₀	6	0	0	-3
x ₁	0	0	0	1
x ₂	1	-1	-1	1
x ₃	1	0	-1	-1
x ₆	1	-1	-1	-2

Tabela 3.12: Quadro ótimo do subproblema P₁.

Solução ótima:

Função Objetivo = x₀ = 6

x₁ = 0

x₂ = 1

x₃ = 1

Tem-se como solução ótima do problema de programação linear FO = 6 e conjunto solução formada pela terna ordenada $S = \{(0, 1, 1)\}$

Resolução do subproblema P₂:

Monta-se o quadro simplex alternativo, incluindo-se uma nova restrição $x_1 \geq 1$

Multiplica-se por (-1) e acrescenta-se a variável de fo.

Tem-se:

$$x_1 \geq 1 \quad (-1) \quad \Rightarrow \quad -x_1 \leq -1 \quad \therefore \quad -x_1 + x_7 = -1$$

$$B = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad x_n = \begin{bmatrix} x_4 \\ x_5 \\ x_6 \end{bmatrix} \quad a_B = [-1 \ 0 \ 0] \quad a_N = [0 \ 0 \ 0]$$

$$\bar{C}_B = a_N - a_B B^{-1} N$$

$$\bar{C}_B = [0 \ 0 \ 0] - [-1 \ 0 \ 0] \cdot \begin{bmatrix} -1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 \end{bmatrix} \Rightarrow \bar{C}_B = [-1/2 \ -1/2 \ 1/2]$$

$$\bar{b} = b_4 - a_b B^{-1} b \quad \bar{b} = -1/2$$

Com a obtenção dos dados forma-se uma nova restrição a ser acrescida no quadro otimista apresentado na Tabela 3.13.

	RHS	- x ₄	- x ₅	- x ₆
x ₀	9/2	-3/2	-3/2	-3/2
x ₁	1/2	-1/2	-1/2	1/2
x ₂	1/2	-1/2	1/2	-1/2
x ₃	1/2	-1/2	-1/2	1/2
x ₇	-1/2	-1/2	-1/2	1/2

Tabela 3.13: Quadro otimista com acréscimo de uma nova restrição.

Escolhe-se a variável que entra e a variável que sai da base; faz-se o pivoteamento. Este processo se repete tantas vezes quantas iterações forem necessárias. Assim, encontra-se a solução ótima do problema de cobertura mínima da função booleana $f_2(A, B, C, D) = \sum m(6, 10, 15) + d(2, 7, 11, 14)$ apresentada na Tabela 3.14.

	RHS	- x ₄	- x ₅	- x ₇
x ₀	6	0	-3	-3
x ₁	1	0	-1	0
x ₂	0	-1	1	-1
x ₃	1	0	-1	0
x ₆	1	1	-2	-2

Tabela 3.14: Quadro ótimo do subproblema P₂.

$$\text{Solução ótima: } \begin{cases} \text{Função Objetivo} = x_0 = 6 \\ x_1 = 1 \\ x_2 = 0 \\ x_3 = 1 \end{cases}$$

Pode-se notar que tanto o Plano de Corte de Gomory como o de Branch and Bound leva de uma solução otimista (infactível) para uma solução ótima global.

3.5 – O Método Dual Simplex

Cada vez que é resolvido um problema de programação linear (PL), simultaneamente, é resolvido outro PL chamado dual. Assim, associado a cada problema de PL que chamaremos de problema primal existe um outro problema dual.

a) Dual de um problema de PL na forma canônica

Seja o problema primal na forma canônica:

$$(P) \quad \begin{cases} \text{Min } z = cx \\ s.a. \\ Ax \geq b \\ x \geq 0 \end{cases}$$

então se define o problema dual de (P) como:

$$(D) \quad \begin{cases} \text{Max } v = wb \\ s.a. \\ wA \leq c \\ w \geq 0 \end{cases}$$

A cada restrição de (P) está associada uma variável de (D) e a cada variável de (P) está associada uma restrição de (D). Se (P) tem m restrições e n variáveis, então (D) tem m variável w e n restrições cada uma delas identificada por um c_i .

b) Dual de um problema primal na forma padrão.

Seja o problema primal na forma padrão

$$(P) \quad \begin{cases} \text{Min } z = cx \\ s.a. \\ Ax \geq b \\ x \geq 0 \end{cases}$$

Define-se o problema dual de (P) da seguinte forma:

$$(D) \quad \left\{ \begin{array}{l} \text{Max } v(w) = wb \\ \text{s.a.} \\ wA \leq c \\ w \text{ irrestrito} \end{array} \right.$$

Obs.: Ambas definições são equivalentes.

c) O Dual do Problema Dual

O dual de um problema de PL é outro problema de PL. Assim, é possível demonstrar que o dual de um problema dual é o próprio problema original. Os problemas primal e dual assim têm definições relativas. Qualquer deles pode ser considerado “problema primal” ou “problema dual”.

Problema Primal:

$$(P) \quad \left\{ \begin{array}{l} \text{Min } z(x) = cx \\ \text{s.a.} \\ Ax \geq b \\ x \geq 0 \end{array} \right.$$

Problema dual:

$$(D) \quad \left\{ \begin{array}{l} \text{Max } v(w) = wb \\ \text{s.a.} \\ wA \geq c \\ w \geq 0 \end{array} \right.$$

Dual do problema dual:

$$\text{Transformando} \quad (D) \quad \left\{ \begin{array}{l} -\{\min v'(w) = (-b')w'\} \\ \text{s.a.} \\ (-A')w' \geq (-c') \\ w' \geq 0 \end{array} \right.$$

E que a transposta de um escalar é o mesmo escalar.

Aplicando o dual:

$$\begin{cases} -\{\max z'(x) = x'(-c') \\ \text{s.a.} \\ x'(-A') \leq (-b') \\ x' \geq 0 \end{cases}$$

Onde x' é um vetor fila de variáveis duais em relação a D.

Transformando a relação anterior, tem-se:

$$\begin{cases} \min z(x) = cx \\ \text{s.a.} \\ Ax \geq b \\ x \geq 0 \end{cases}$$

Obs.: No processo de transformação foi ampliada a seguinte equivalência:

$$\max f(x) = - [\min -f(x)].$$

3.6.1 -Teorema Fundamental da Dualidade

Em relação aos problemas primal e dual, exatamente uma das seguintes afirmações é verdadeira:

- a) Ambos possuem solução ótima x^* e w^* com $cx^* = w^*b$.
- b) Um problema tem valor de função objetivo ilimitado e, nesse caso, o outro problema é infactível.

c) Ambos os problemas são infactíveis. Assim o teorema da dualidade mostra que dualidade não é uma propriedade completamente simétrica. Assim, uma forma equivalente de expressar a propriedade de dualidade é assim:

$$(P) \text{ é ótimo} \Leftrightarrow (D) \text{ é ótimo}$$

$$(P) \text{ (D) ilimitado} \Rightarrow (D) \text{ (P) infactível}$$

$$(P) \text{ (D) infactível} \Rightarrow (D) \text{ (P) ilimitado ou infactível}$$

$$(P) \text{ (D) infactível} \Leftrightarrow (D) \text{ (P) ilimitado na forma homogênea}$$

O método dual simplex resolve o problema dual diretamente no quadro primal simplex e indiretamente resolve o problema primal original [16]. O processo é iniciado a partir de uma SFB para o problema dual e em cada iteração se obtém uma SFB para o problema dual que é maior ou igual ao quadro anterior. O processo termina quando se verifica a otimalidade do problema dual. Se esse quadro ótimo é encontrado, então, tem-se o quadro ótimo para o problema dual e também para o problema primal.

$$(P) \quad \begin{cases} \text{Min } z(x) = cx \\ \text{s.a.} \\ Ax \geq b \\ x \geq 0 \end{cases}$$

(4)

Se B é uma base do primal e não é necessariamente factível; o primal (4) mostra o quadro simplex para esta base:

- Se $\bar{b}_i \geq 0 \ \forall_i = 1, \dots, m \Rightarrow$ o quadro primal é factível $\bar{b} = B^{-1} b \geq 0$;
- Por outro lado, tem-se que: $(z_j - c_j) \leq 0 \ \forall_j = 1, 2, \dots, n, n + 1, \dots, (n + m)$ então o quadro é ótimo.

Interpretação da factibilidade do dual no quadro simplex do primal, representado na Tabela 3.15.

		Variáveis de Folga						RHS
	z	x ₁	x ₂	...	x _n	x _{n+1}	x _{n+m}	
z	1	z ₁ - c ₁	z ₁ - c ₁	...	z _n - c _n	z _{n+1} - c _{n+1}	z _{n+m} - c _{n+m}	c _B $\bar{b}$
x _{B1}	0	y ₁₁	y ₁₂		y _{1n}	y _{1,n+1}	y _{1,n+m}	$\bar{b}_1$
x _{B2}	0	y ₂₁	y ₂₂		y _{2n}	y _{2,n+1}	y _{2,n+m}	$\bar{b}_2$
⋮	⋮	⋮	⋮		⋮	⋮	⋮	⋮
x _{Bm}	0	y ₃₁	y ₃₂		y _{3n}	y _{3,n+1}	y _{3,n+m}	$\bar{b}_m$

Tabela 3.15. Interpretação de factibilidade do dual no quadro simplex primal.

Seja Tabela 3.15, e seja B uma base não necessariamente factível.

- O quadro é factível se: $\bar{b} = B^{-1}b \geq 0$
- O quadro é ótimo se : $(z_j - c_j) \leq 0 \quad j = 1, 2, \dots, n + m$

Seja w $C_B B^{-1}$:

(*) Para $j = 1, 2, \dots, n \Rightarrow (z_j - c_j) = C_B B^{-1} a_j - C_j = w a_j - C_j$

Se a Tabela 3.15. é ótimo, então para $j = 1, \dots, n$ tem-se que:

$w a_j - c_j$ para $j = 1, \dots, n$.

A Tabela 3.16 representa o dual simplex.

	z	x ₁	...	x _j	...	x _k	...	x _n	...	RHS
z	1	z ₁ - c _{1b} ...		z _j - c _j ...		z _k - c _k ...		z _n - c _n		c _B $\bar{b}$
x _{B1}	0	y ₁₁	...	y _{1j}	...	y _{1k}	...	y _{1n}		$\bar{b}_1$
x _{B2}	0	y ₂₁	...	y _{2j}	...	y _{2k}	...	y _{2n}		$\bar{b}_2$
⋮	⋮	⋮	...	⋮	...	⋮	...	⋮		⋮
x _{Br}	0	y _{r1}	...	y _{rj}	...	y _{rk}	...	y _{rn}		$\bar{b}_m$
⋮	⋮	⋮	...	⋮	...	⋮	...	⋮		⋮
x _{Bm}	0	y _{m1}	...	y _{mj}	...	y _{mk}	...	y _{mn}		$\bar{b}_m$

Tabela 3.16 Apresenta o quadro dual simplex.

- Se tableau dual factível $\Rightarrow z_j - c_j \leq 0$ (problema de minimização).
- Se o tableau é primal factível $\Rightarrow$ Solução ótima ($\bar{b}_i \geq 0, \forall i$).

Considere algum $\bar{b}_r < 0$. A idéia é buscar transformar os $\bar{b}_i \geq 0$, para tornar o primal factível mantendo a otimalidade do primal (factibilidade do dual).

Exemplo 3.2

A cobertura da função booleana $f_3(A, B, C) = \sum m(2, 4, 5, 6, 7) + d(1, 3)$, apresentada no Mapa de Karnaugh da Figura 3.3 pode ser formulada como um problema de programação linear inteira binária.

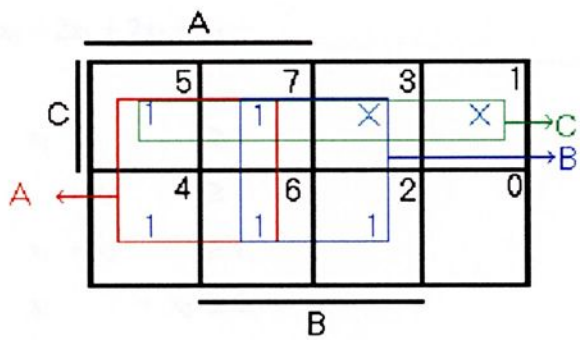


Figura 3.3: Formulação da função $f_3(A, B, C) = \sum m(2, 4, 5, 6, 7) + d(1, 3)$.

A Tabela 3.17 apresenta a cobertura dos mintermos para a função $f_3(A, B, C) = \sum m(2, 4, 5, 6, 7) + d(1, 3)$ formada com base no conjunto de implicantes primos que foram gerados na primeira fase do método.

Implicantes	Mintermos				
	2	4	5	6	7
1XX		X	X	X	
X1X	X			X	X
XX1			X		X

Tabela 3.17: Tabela de cobertura dos Implicantes Primos.

Com auxílio da tabela de cobertura dos mintermos, representada na Tabela 3.17, transforma-se o problema de cobertura em um problema de programação linear inteira binária $\{0, 1\}$.

Tem-se o PL:

$$\text{Min } 2(1XX + X1X + XX1)$$

s.a.

$$1XX \geq 1$$

$$X1X \geq 1$$

$$1XX + X1X \geq 1$$

$$1XX + XX1 \geq 1$$

$$X1X + XX1 \geq 1$$

$$1XX, X1X, XX1 \in \{0, 1\}.$$

Tornando-se o PL numa linguagem matemática com incógnitas x_i , tem-se:

$$x_0 = \text{F.O.}, \quad x_1 = 1XX, \quad x_2 = X1X \text{ e } x_3 = XX1.$$

Escreve-se o PL:

$$\text{Min } x_0 = 2x_1 + 2x_2 + x_3$$

s.a.

$$x_1 \geq 1$$

$$+ x_2 \geq 1$$

$$x_1 + x_2 \geq 1$$

$$x_1 + x_3 \geq 1$$

$$+ x_2 + x_3 \geq 1$$

$$x_1, x_2, x_3 \in \{0, 1\}$$

Multiplicam-se as restrições por (-1) e coloca-se o problema na forma padrão, e acrescentam-se as variáveis de folgas o PL apresenta-se na seguinte forma:

$$\text{Min } x_0 = 2x_1 + 2x_2 + x_3$$

s.a.

$$-x_1 \quad \quad \quad + x_4 \quad \quad \quad = -1$$

$$\quad -x_2 \quad \quad \quad + x_5 \quad \quad \quad = -1$$

$$-x_1 - x_2 \quad \quad \quad + x_6 \quad \quad \quad = -1$$

$$-x_1 \quad \quad -x_3 \quad \quad \quad + x_7 \quad \quad \quad = -1$$

$$\quad -x_2 - x_3 \quad \quad \quad + x_8 = -1$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \in \{0, 1\}.$$

Formulação do quadro inicial:

$$A = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$b = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$C_B = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$C_N = [2 \ 2 \ 1]$$

$$N = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ -1 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

$$\bar{b} = B^{-1}b \Rightarrow \bar{b} = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix}$$

$$FO = C_B B^{-1}b \Rightarrow FO = 0 \quad (z_j - c_j) = C_B B^{-1}N - C_N$$

$$(z_j - c_j) = -C_N \Rightarrow (z_j - c_j) = [-2 \ -2 \ -1]$$

Com estes dados, pode-se montar o quadro inicial simplex, e resolver o PL como mostram as Tabelas: 3.18., 3.19 e 3.20.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	rhs
x_0	1	-2	-2	-1	0	0	0	0	0	0
x_4	0	-1	0	0	1	0	0	0	0	-1
x_5	0	0	0	0	0	1	0	0	0	-1
x_6	0	-1	-1	-1	0	0	1	0	0	-1
x_7	0	-1	-1	-1	0	0	0	1	0	-1
x_8	0	0	0	-1	0	0	0	0	1	-1

Tabela 3.18: Tabela inicial do quadro simplex x_1 entra na base e x_4 sai da base.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	rhs
x_0	1	0	-2	-1	-2	0	0	0	0	2
x_1	0	1	0	0	-1	0	0	0	0	1
x_5	0	0	-1	0	0	1	0	0	0	-1
x_6	0	0	0	-1	-1	0	0	1	0	0
x_7	0	0	0	-1	1	0	0	1	0	0
x_8	0	0	-1	-1	0	0	0	0	1	-1

Tabela 3.19: Tabela inicial do quadro simplex x_2 entra na base e x_5 sai da base.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	rhs
x_0	1	0	0	-1	-2	-2	0	0	0	4
x_1	0	1	0	0	-1	0	0	0	0	1
x_5	0	0	1	0	0	-1	0	0	0	1
x_6	0	0	0	-1	-1	0	0	1	0	0
x_7	0	0	0	-1	1	0	0	1	0	0
x_8	0	0	0	-1	0	-1	0	0	1	0

Tabela 3.20: Solução ótima da função booleana $f_3(A, B, C) = \Sigma m(2, 4, 5, 6) + d(1, 3)$.

$$\text{Solução ótima} \left\{ \begin{array}{l} \text{Função Objetivo} = x_0 = 4 \\ x_1 = 1 \\ x_2 = 1 \\ x_3 = 0 \end{array} \right.$$

Tem-se como solução ótima do problema de programação linear da função booleana $f_3(A, B, C) = \Sigma m(2, 4, 5, 6, 7) + d(1, 3)$ FO = 4 e o conjunto solução formado pela terna ordenada $S = \{(1, 1, 0)\}$.

Os exercícios resolvidos no Capítulo 3 serviram como modelo para a formulação dos programas computacionais Branch2, Gomo2 e o Dual, onde foram empregados; no próximo capítulo, como exemplo, para comparar custos e os parâmetros tempo, memórias e números de iterações.

CAPÍTULO 4 – Aplicação dos Métodos

Resolveu-se no Capítulo 3, o exemplo 3.1, no qual aplicou-se o método Simplex, obtendo-se como resultado uma solução otimista. Neste caso, recorreram-se a outros métodos como: Plano de Corte de Gomory, Branch and Bound e o método Dual para encontrar a solução ótima. Na formação do problema de coberturas utilizou-se Pigene [10], que implementa a primeira fase do método de Quine-McCluskey. De posse de todos os implicantes primos, o problema de cobertura foi formulado como um problema de programação linear inteira binária.

Neste capítulo, empregaram-se recursos computacionais, implementando-se programas para a execução dos algoritmos Simplex, Plano de Corte de Gomory, Branch and Bound e Dual Simplex.

Dezenas de casos de funções booleanas foram estudados com objetivo de melhorar o desempenho dos algoritmos implementados e aqui foram desenvolvidos: o Branch2, Gomo2 e Dual.

Observou-se a quantidade de memórias, o tempo de execução, o número de iterações e o custo da solução obtida.

No exemplo 4.1, apresentaram-se em detalhes as tabelas geradas pelos algoritmos e os resultados obtidos, apresentados de forma gráfica.

Em todos os outros casos estudados, apresenta-se somente o resultado tabulado, para no final apresentar gráficos com os dados compilados.

Exemplo 4.1:

Considere a função booleana $f_4(A, B, C, D) = \sum m(6, 10, 12, 15) + d(2, 7, 11, 14)$. Na Figura 4.1, o arquivo de entrada do programa Pigene, contendo os mintermos e os irrelevantes da função, a partir dos quais geram-se todos os implicantes primos, e na Figura 4.2 apresenta-se o arquivo de saída contendo todos os implicantes primos gerados.

A Figura 4.3 apresenta no Mapa de Karnaugh e os implicantes primos gerados. Pode-se verificar que a cobertura apresentada não é mínima.

De posse de todos os implicantes primos, obtém-se, a partir da tabela de cobertura apresentada na Tabela 4.1, um problema de programação linear inteira 0 e 1.



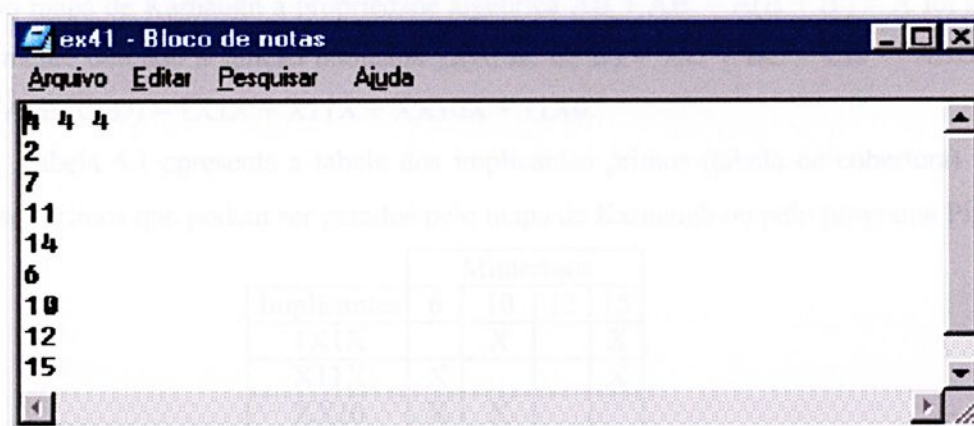


Figura 4.1: Arquivo ex41.min de entrada do programa Pigene.

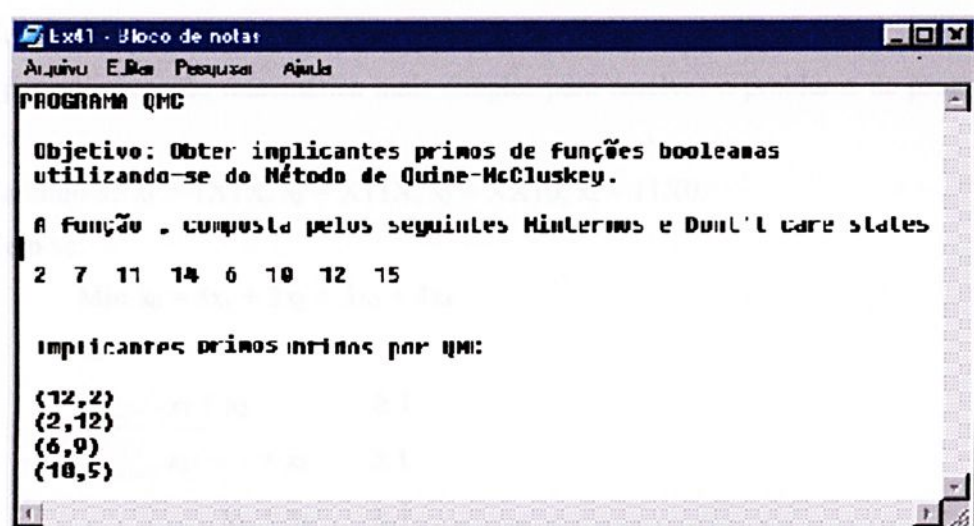


Figura 4.2: Arquivo ex41.qmc de saída do programa Pigene.

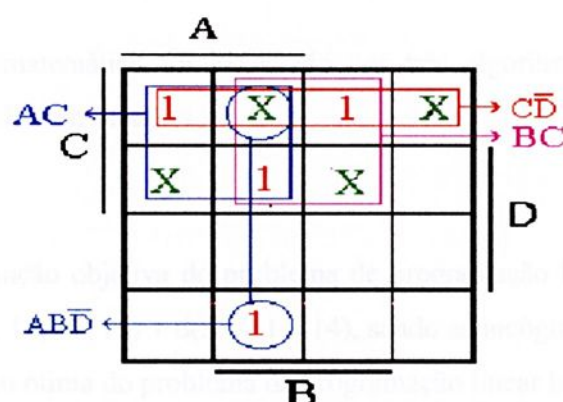


Figura 4.3: Apresentação do mapa de Karnaugh da função f_4 .

No mapa de Karnaugh a propriedade algébrica $AB + AB' = A(B + B') = A$ foi aplicada implicitamente obtendo a função booleana $f_4(A, B, C, D) = AC + BC + CD' + ABD'$ ou da forma $f_4(A, B, C, D) = 1X1X + X11X + XX10X + 11X0$.

A Tabela 4.1 apresenta a tabela dos implicantes primos (tabela de cobertura) para os implicantes primos que podem ser gerados pelo mapa de Karnaugh ou pelo programa Pigene.

Implicantes	Mintermos			
	6	10	12	15
1X1X		X		X
X11X	X			X
XX10	X	X		
11X10			X	

Tabela 4.1: Tabela de Cobertura dos Implicantes Primos.

Trocando-se a notação da função booleana $f_4(A, B, C, D) = \sum m(6, 10, 12, 15) + d(2, 7, 11, 14)$, por uma notação matemática mais simples para resolver o problema de programação linear.

Fazendo-se $x_1 = 1X1X$, $x_2 = X11X$, $x_3 = XX10$, $x_4 = 11X0$.

Tem-se:

$$\text{Min } x_0 = 3x_1 + 3x_2 + 3x_3 + 4x_4$$

s.a.

$$x_1 + x_2 \geq 1$$

$$x_1 + x_3 \geq 1$$

$$x_2 + x_3 \geq 1$$

$$x_4 \geq 1$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

Este problema matemático foi executado nos três algoritmos implementados, cujos passos foram apresentados em detalhes.

4.1 - Método Dual

FO = 10 é a função objetiva do problema de programação linear da função booleana $f_4(A, B, C, D) = \sum m(6, 10, 12, 15) + d(2, 7, 11, 14)$, sendo as incógnitas $x_2 = 1$, $x_3 = 1$, $x_7 = 1$ e $x_4 = 1$. Assim, a solução ótima do problema de programação linear binária da função booleana é dada pela hepta ordenada: $\{(0, 1, 1, 1, 0, 0, 1)\}$.

4.2 – Método Branch and Bound

Resolveu-se este problema utilizando programa Branch2 o qual apresentou uma solução ótima. Tendo a FO o mesmo custo encontrado na execução do Dual.

O programa implementado Branch2 gera duas novas restrições, pois o problema foi dividido em dois subproblemas, assim encontrando o mesmo custo, mas com subconjuntos soluções diferentes.

A Tabela 4.2 apresenta o quadro otimista obtida através da implementação do programa do algoritmo Simplex Alternativo.

8,5	-1,5	-1,5	-1,5	-4,0
0,5	-0,5	-0,5	0,5	0
0,5	-0,5	0,5	-0,5	0
0,5	0,5	-0,5	-0,5	0
1,0	0	0	0	-1,0

Tabela 4.2: Quadro da Solução otimista.

Resolvendo-se P_1 .

Para encontrar uma solução ótima acrescentou-se uma restrição S no quadro otimista da Tabela 4.2, apresentada na Tabela 4.3 e executou-se o programa Branch2.

A restrição a ser acrescentada no quadro otimista é uma restrição menor ou igual e fazendo a atualização devida tem o que se segue:

$$S = -0,5 \ -0,5 \ -0,5 \ 0,5 \ 0$$

8,5	-1,5	-1,5	-1,5	-4,0
0,5	-0,5	-0,5	0,5	0
0,5	-0,5	0,5	-0,5	0
0,5	0,5	-0,5	-0,5	0
1,0	0	0	0	-1,0
-0,5	-0,5	-0,5	0,5	0

Tabela 4.3: Quadro da Solução otimista acrescentada a restrição S.

A Tabela 4.4 apresenta uma candidata à solução ótima do problema de cobertura mínima da função booleana $f_4(A, B, C, D) = \sum m(6, 10, 12, 15) + d(2, 7, 11, 14)$, o valor da função objetivo é $FO = 10$ e o conjunto solução é $\{(0, 1, 1, 1, 1)\}$.

10	-3	-3	-3	-4
0	0	0	1	0
1	-1	0	-1	0
1	0	-1	-1	0
1	0	0	0	-1
1	-1	-1	2	0

Tabela 4.4: Quadro ótimo obtido através da implementação do programa Branch2.

Resolvendo-se P_2

Acrescenta-se também uma restrição $S = [-0,5 \ 0,5 \ 0,5 \ -0,5 \ 0]$ maior ou igual como mostra a Tabela 4.5

8,5	-1,5	-1,5	-1,5	-4,0
0,5	-0,5	-0,5	0,5	0
0,5	-0,5	0,5	-0,5	0
0,5	0,5	-0,5	-0,5	0
1,0	0	0	0	-1,0
-0,5	0,5	0,5	-0,5	0

Tabela 4.5: Quadro da Solução otimista acrescentada a restrição S.

A Tabela 4.6 apresenta uma candidata à solução ótima do problema de cobertura mínima da função booleana $f_4(A, B, C, D) = \sum m(6, 10, 12, 15) + d(2, 7, 11, 14)$, o valor da função objetivo é $FO = 10$

10	-3	-3	-3	-4
0	0	0	1	0
1	-1	0	-1	0
1	0	-1	-1	0
1	0	0	0	-1
1	-1	-1	2	0

Tabela 4.6: Quadro ótimo obtido através da implementação do programa Branch2.

Sendo o resultado das Tabelas 4.4 e 4.6 iguais tem-se $FO = 10$ e o conjunto solução é $\{(0, 1, 1, 1, 1)\}$

4.3 - Método Plano de Corte

Resolveu-se este problema utilizando também o programa Gomo2, o qual apresentou uma solução ótima.

A Tabela 4.2 apresenta o quadro otimista obtida através da implementação do programa do algoritmo Simplex Alternativo.

Para encontrar uma solução ótima acrescentou-se uma restrição S no quadro otimista da Tabela 4.6, apresentada na Tabela 4.8, e implementou-se o programa Gomo2.

Restrição a ser acrescentada no quadro otimista.

$S = -0.5000 \quad -0.5000 \quad -0.5000 \quad -0.5000 \quad 0$

8,5	-1,5	-1,5	-1,5	-4,0
0,5	-0,5	-0,5	0,5	0
0,5	-0,5	0,5	-0,5	0
0,5	0,5	-0,5	-0,5	0
1,0	0	0	0	-1,0
-0,5	-0,5	-0,5	-0,5	0

Tabela 4.7: Quadro da Solução otimista com acréscimo da restrição S.

A Tabela 4.9 apresenta uma candidata à solução ótima do problema de cobertura mínima da função booleana $f_4(A, B, C, D) = \sum m(6, 10, 12, 15) + d(2, 7, 11, 14)$, o valor da função objetivo é FO = 10 e o conjunto solução é $\{(1, 0, 1, 1, 1, 1)\}$.

10	-3	-3	-3	-4
1	0	0	1	0
0	-1	0	-1	0
1	0	-1	-1	0
1	0	0	0	-1
1	-1	-1	2	0

Tabela 4.8: Quadro ótimo obtido através da implementação do programa Gomo2.

Os três métodos utilizados foram comparados, e o resultado obtido e aparece na Tabela 4.10 e nos gráficos 4.1, 4.2, 4.3, analisando o comportamento dos programas Branch2, Gomo2 e o Dual, quanto aos custos, memória, número de iterações e tempo computacional.

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,37	4	2772	10
Dual	0,22	3	1780	10
Gomo2	1,7	4	2448	10

Tabela 4.9: Comparação dos programas Branch2, Dual e Gomo2.

Através dos parâmetros encontrados, observa-se que os custos são iguais nos três programas.

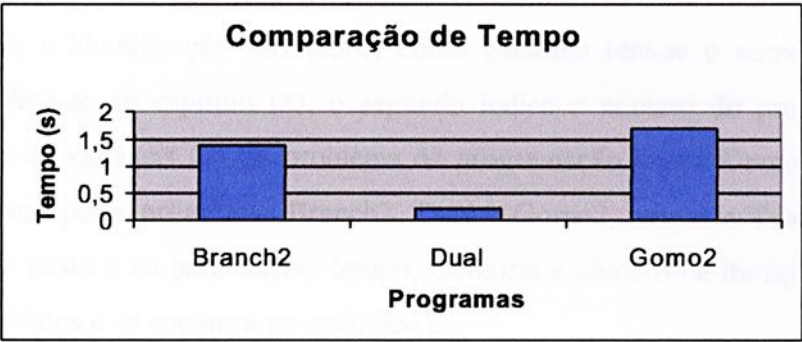


Gráfico 4.1: Comparação de Tempo entre os programas.

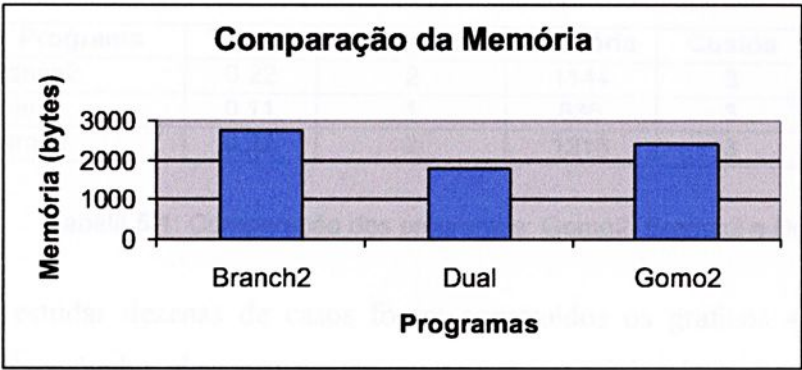


Gráfico 4.2: Comparação da Memória entre os programas Branch2, Dual e Gomo 2.

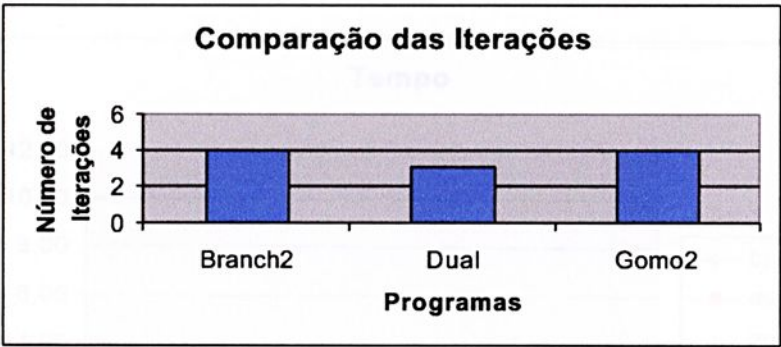


Gráfico 4.3: Comparação do número de Iterações entre os programas-.

Foram apresentadas as tabelas comparativas entre os algoritmos implementados, a partir das quais geraram-se gráficos, onde se pôde avaliar o desempenho dos programas Branch2, Gomo2 e Dual.

CAPÍTULO 5 – Comparação dos Parâmetros

Para a identificação dos casos, como exemplo tem-se o número 5.1-2. O primeiro número refere-se ao capítulo (5), o segundo indica o número do problema (1) o último à quantidade de variáveis (2) do problema de programação linear. Como exemplo de um caso implementado pelos programas Branch2, Dual e Gomo2, tem-se a Tabela 5.1, onde se pôde comparar o custo e os parâmetros: tempo, memória e número de iterações. Dezenas de casos foram resolvidos e se encontra no apêndice C.

5.1-. 2 $f(A, B, C) = \sum m(5, 9) + d(1, 7, 13, 15)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	2	1144	3
Dual	0,11	1	836	3
Gomo2	0,22	2	1216	3

Tabela 5.1: Comparação dos programas: Gomo2, Branch2 e Dual.

Após estudar dezenas de casos foram construídos os gráficos 4.4, 4.5 e 4.6 para mostrar o desempenho dos programas quanto à memória, iteração e tempo, não há comparação de custos, pois em todos os problemas, os resultados nos três programas foram os mesmos.

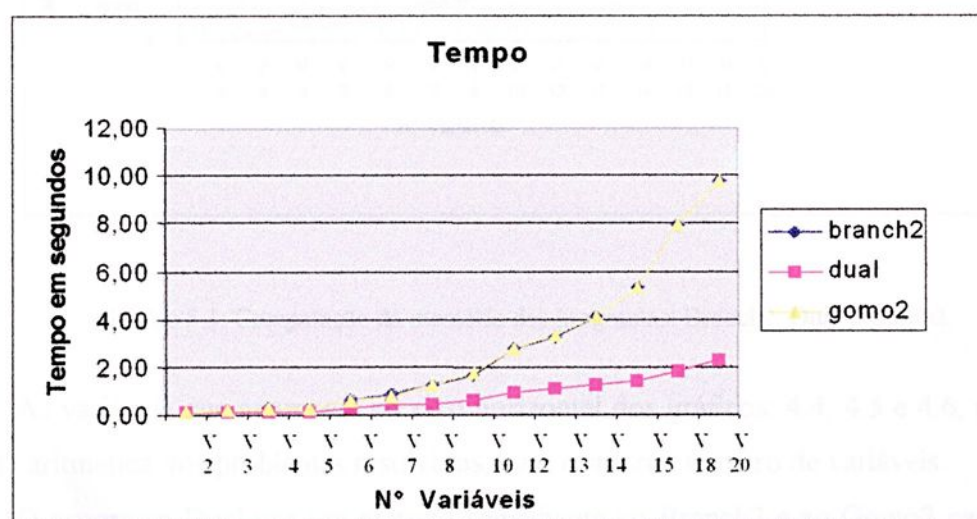


Gráfico 5.1: Comparação do tempo dos programas: Branch2, Dual e Gomo2.

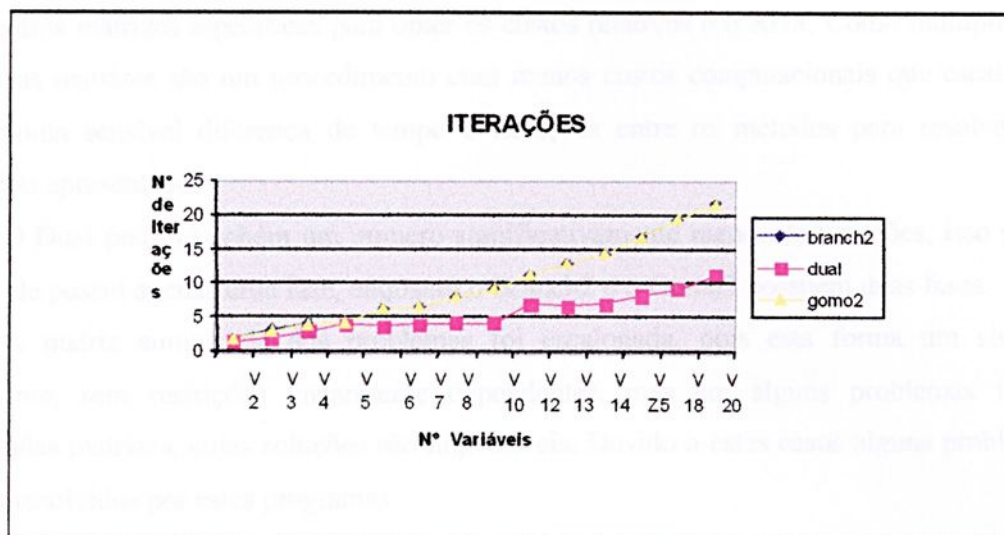


Gráfico 5.2: Comparação das iterações dos programas: Branch2, Dual e Gomo2.

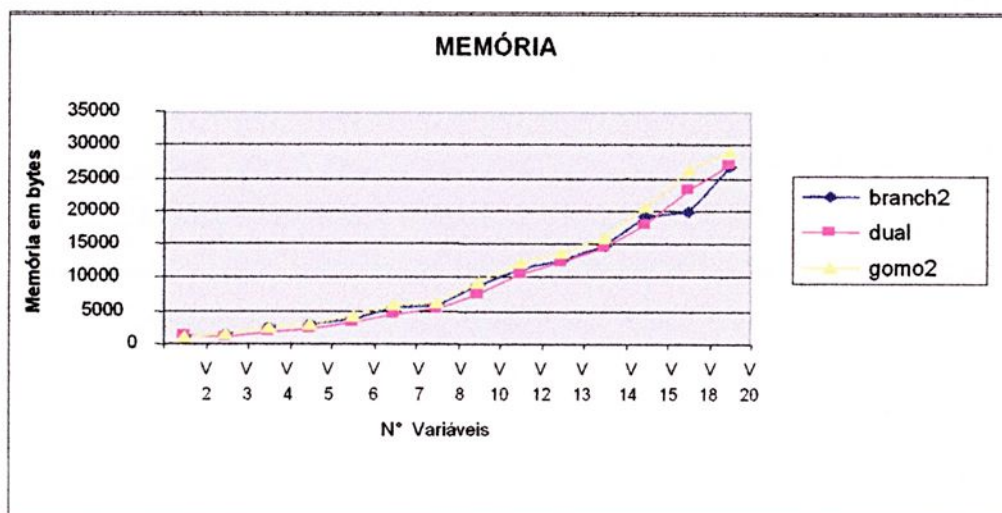


Gráfico 5.3: Comparação da memória dos programas: Branch2, Dual e Gomo2.

As variáveis que aparecem no eixo horizontal dos gráficos: 4.4, 4.5 e 4.6, representam a média aritmética dos problemas resolvidos, com o mesmo número de variáveis.

O programa Dual usa um método semelhante ao Branch2 e ao Gomo2 para a escolha do pivô, sua principal diferença está na atualização da tabela (custo relativo, RHS), enquanto o Branch2 e o Gomo2 utiliza um método semelhante a um escalonamento de matrizes para pivotar a tabela (matriz), o Dual executa esta tarefa apenas invertendo a base e multiplicando

esta a outras matrizes específicas para obter os custos relativos e o RHS. Como multiplicar e inverter as matrizes são um procedimento com menos custos computacionais que escalonar, nota-se uma sensível diferença de tempo e iterações entre os métodos para resolver os problemas apresentados.

O Dual possui também um número significativamente menor de iterações, isso se dá porque ele possui apenas uma fase, enquanto o Branch2 e o Gomo2 possuem duas fases.

A matriz aumentada dos problemas foi escalonada, pois esta forma um sistema equivalente, sem restrições linearmente dependentes, mas em alguns problemas foram encontradas matrizes, cujas soluções são impossíveis. Devido a estes casos alguns problemas não são resolvidos por estes programas.



CAPÍTULO 6 – Conclusão

Este estudo, longe está de ser considerado concluído. Entretanto, após ter-se apresentado conceitos, levantada a problematização atinente ao assunto, e feitas às abordagens que se julgou serem necessárias e suficientes para essa etapa da pesquisa, dentro dos critérios de sistematização de trabalho, condizentes com a bibliografia disponível, passa-se a apresentar algumas conclusões, certos de que os resultados podem ser considerados satisfatórios.

Resolveram-se problemas de cobertura, partindo-se de métodos eficientes para a geração de implicantes primos de uma função booleana. Observou-se que, a cobertura da função pode ser formulada como um problema de programação linear 0 e 1.

Na formulação de problemas de cobertura, utilizou-se o método de Quine-McCluskey na simplificação de funções booleanas no sentido de minimizar os custos dos circuitos. A idéia fundamental do método é a aplicação repetida da propriedade $AB + AB' = A(B + B') = A$, sendo que, o método de Quine-McCluskey consiste da geração de todos os implicantes primos e cobertura mínima dos mintermos. O mapa de Karnaugh é outro método de simplificação de funções booleanas, o qual contém as mesmas informações contidas na Tabela Verdade. O algoritmo PIGENE e o método de Expansão são métodos utilizados na geração de implicantes primos para a formulação de problemas de cobertura através de programas computacionais, oriundos do método de Quine-McCluskey.

No início deste trabalho de pesquisa, tinha-se como objetivo somente comparar o desempenho computacional dos métodos Branch and Bound e Plano de Corte para definir qual melhor se aplicam no caso específico para o problema de cobertura de uma função booleana.

Conforme apresentado no Capítulo 4, o desempenho de ambos foi muito próximo, motivo pelo qual buscou-se outro método de otimização. Optou-se por implementar o método Dual Simplex e comparar este último com os outros dois. Os resultados foram surpreendentes com relação ao número de iterações e conseqüentemente em tempo de execução.

A resolução do problema de cobertura mínima da função booleana no Capítulo 3, serviu de base para a elaboração dos programas computacionais Branch2, Gomo2 e Dual que foram desenvolvidos no MATLAB 5.3.

Com a implementação dos programas, foram resolvidas dezenas de estudos de casos para obter parâmetros como, custos, memórias, tempo e iterações. Estes parâmetros foram

destacados nas tabelas para cada programa. Observando as tabelas e os gráficos, pôde-se notar que o programa Dual levou vantagem em relação a Branch2 e Gomo2, no número de iterações e tempo gasto, sendo que a quantidade de memória nos três programas foram praticamente iguais.

Este trabalho deu origem ao artigo Branch and Bound Aplicado no Problema de Cobertura de Funções Booleanas apresentado e publicado pela International Conference on Engineering and Computer Education, realizado no período de 16 a 19 de março de 2003, na cidade de São Vicente, Estado de São Paulo.

Outros algoritmos de programação matemática deverão ser implementados em pesquisa que pretendo realizar no futuro, para que possam ser comparados com os resultados obtidos neste trabalho. Como exemplo, tem-se o Algoritmo Implícito Zero-Um de Balas, que é resolvido sem o uso de tabelas e de pivoteamento. Objetiva-se também implementar o Algoritmo Genético, que deverá, igualmente, ser submetido a parâmetros de comparação.



CAPÍTULO 7 – Referências Bibliográficas

- [01] DE MICHELE, Giovanni "Synthese and Optimization of Digital Circuits", McGraw-Hill, Inc, New York, 1994.
- [02] BAZARAA, M. S., Jarvis, J. J. and Sherali, H. D- "*Linear Programming and Network Flows*" - Second Edition - John Wiley & Sons, Inc. - New York - 1993.
- [03] HILL, F. e G. Peterson, *Switching Theory and Logical Design*, Wiley, New York, 1981.
- [04] McCLUSKEY, E.J. "*Minimization of Boolean functions*, *The Bell System Technical Journal*, 35, No 5, November 1956. pp. 1417-1444
- [05] KOHAVI, Z. *Switching and Finite Automata Theory*, McGraw-Hill, 1970
- [06] QUINE, W. *The Problem of Simplifying Truth Functions*, *American Mathematical Monthly*, Vol. 59, 1952, pp. 521-531
- [07] RUDELL and A. Sangiovanni-Vincentelli, *Multiple - valued Minimization for PLA Optimization*, *IEEE Transactions on CAD/ ICAS*, Vol. CAD-6, No. 5, September 1987, pp 727-750.
- [08] DAGENAIS, M - Agarwal V. and Rumin, N. McBOOLE: *A New Procedure for Exact Logic Minimization*, *IEEE Transactions on CAD/ICAS*, Vol. CAD-5, No. 1, January 1986, pp. 229-223
- [09] COUDERT, O. - Madre, J. C.: "*Implicit and incremental computation of primes and essential primes of Boolean functions*, In *Proc. of the Design Automation Conf.* (Anaheim,CA, June 1992), pp. 36-39.
- [10] EMER, F.R.P, "Formulação Matemática do Problema de Otimização de Funções Booleanas Através do Método de Expansão de Shannon", *Dissertação de Mestrado Faculdade de Engenharia Elétrica - UNESP - Ilha Solteira - Outubro de 2002.*
- [11] HLAVICKA, J. and Fišer, *BOOM- a Heuristic Boolean Minimizer*. *Proc. ICCAD-2001*, San Jose. Cal. (USA)



- [12] MENDELSON, E "Álgebra Booleana e Circuitos de Chaveamento", Coleção Schaum, Editora McGraw-Hill. , 1977.
- [13] GARFINKEL, R. S. and Nehauser, G. L. "*Integer Programming*", Library of Congress Cataloging in Publication Data - United States - 1976.
- [14] SANTOS, T.A., Silva, A.C.R e Santos, C.E.S. "*Aplicação do Método de Branch and Bound Aplicado na Cobertura de Funções Booleanas*", 3rd International Conference on Engineering and Computer Education, ICECE, March 16 - 19, 2003, São Vicente-S.P. BRAZIL.
- [15] NATALE, F. "Automação Industrial", Série Brasileira de Tecnologia, Nobel: Siemens S.A.,1989.
- [16] LÁZARO, R.A.R., Rendón, R.A.G. "Optimización en Sistemas Eléctricos I" Universidade Estadual Paulista – UNESP- Brasil
- [17] NEMHAUGER, G. and Wolseyl, Interger and Combinatorial Optimization, Wiley, New York, 1988.
- [18] PAPARDIMITRIOV, C. and Steiglitzk, Combinatorial Optimization, Prestice-Hall, Englewood Cliffs, 1982.



Apêndice A

1- Código Fonte do Programa Branch2

```
%Tabela do Quadro Alternativo ou Metodo de Gomory
clear
clc

%-----
%O PL e resolvido pelo método de duas fases
[i,j]=size(A);           %i3 numero de linhas de A=numero de

%PREPARANDO O A E b PARA O QUADRO ALTERNATIVO
A(:,j+1)=ones(i,1);
C=rref(A);
b=C(:,j+1);
A=C(:,1:j);

%Fórmulas de preparação
[i,j]=size(A);           %i, j números de linhas e colunas
respectivamente da matriz A

%b=ones(i,1);           %b e a coluna das restricoes
B=eye(i);               %B e a base representado pela matriz
identidade
AI=[A,-eye(i),eye(i)];  %AI coeficientes das restricoes acrescida da
matriz
%identidade

[i1,j1]=size(AI);       %i1, j1 n° de linhas e colunas
respectivamente da matrix AI
N=[A,-eye(i)];          %N e a matriz formada pelos coeficientes das
restricoes
%acrescida das variaveis de folga
[i2,j2]=size(N);        %i2, j2 n° de linhas e colunas
respectivamente da matrix N
cn=zeros(1,j2);         %cn custos nao basicos formada pelos
coeficientes das
%variaveis nao basicas

%Custo relativo da função objetivo I
cb=ones(1,i);           %cb coeficientes das variaveis baseicas
Cr=cb*inv(B)*N-cn;       %Cr custos relativos da funcao objetivo I
FO=cb*inv(B)*b;          %FO valor da funacao objetivo I

%Custo relativo da função objetivo II
cb2=zeros(1,i);         %cb coeficientes das variaveis baseicas da
fase II
cn2=[c,zeros(1,i)];     %cn custos nao basicos formada pelos
coeficientes das
%variaveis nao basicas da fase II
Cr2=cb2*inv(B)*N-cn2;   %Cr custos relativos da funcao objetivo II
FO2=cb2*inv(B)*b;       %FO valor da funacao objetivo II

%Criando a matriz M2
[i5,j5]=size(N);        %[i5,j5] tamanho de N
M2=zeros(i5+2,j5+1);    %M2 matriz nula para auxiliar a formacao da
tabela
```

```

M2(1,2:j5+1)=Cr;           %M2(1,2:j5+1) acrescida da linha Cr
M2(2,2:j5+1)=Cr2;          %M2(2,2:j5+1) acrescida da linha Cr2
M2(3:i5+2,2:j5+1)=N;       %M2(3:i5+2,2:j5+1) acrescida da matriz N
M2(3:i5+2,1)=b;            %M2(3:i5+2,1) acrescida de b
M2(1,1)=FO;                %M2(1,1) acrescida da funcao objetivo FO da
fase I
M2(2,1)=FO2;               %M2(2,1) acrescida da funcao objetivo FO2 da
fase II

```

```
%teste das iterações
```

```

[i3,j3]=size(M2);          %i3, j3 n° de linhas e colunas
respectivamente da matrix M2
[c1,c2]=size(M2(1,2:i3));
v=M2(1,2:i3);
f=find(v<=0);
f=v(1,f);
[c3,c4]=size(f);
ite=0;
aux_ite1=0;                %Variável de decisão de aux_teste
while c4~=c2
%for v=1:3

```

```

[i3,j3]=size(M2);          %i3, j3 n° de linhas e colunas
respectivamente da matrix M2

```

```
%Encontrando o maior valor da FO II
```

```

g=max(M2(1,2:j3));
for k=2:j3                  %O n° de repetições do laço é o n° da coluna
pivot k
    y=M2(1,k);
    if g==y
        break
    end
end

```

```
%Encontrando quem sai da base
```

```

d=M2(3:i3,1)./M2(3:i3,k);  %d coluna do pivot
w=find(d>0);
w=d(w);
if w==inf
    aux_ite1=2;
    aux_teste21;
    break
elseif d<=0
    %disp('ILIMITADO 2')
    aux_ite1=2;
    aux_teste21;
    break
else
    l=d(find(d>0));
    l=min(l);
    l=find(l==d);
    l=min(l);
    l=l+2;
    pp=[l k];               %endereço do elemento pivot
end

```

```
%Fazendo o pivoteamento
```



```

for p2=1:j3
    if k~=p2
        for p1=1:i3
            if l~=p1
                M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(l,p2)/M2(l,k);
            end
        end
    end
end
end

```

```

M2(1,:)=M2(1,:)/M2(1,k);
M2=[M2(:,1:k-1),M2(:,k+1:j3)];

```

```

%Verificando se A é a matriz identidade

```

```

i4=0;
[i3,j3]=size(M2);
if i==j
    Q=ones(i,j);
    Q1=A*Q;

```

```

else
    Q1=1;
    Q=2;

```

```

end
    if Q1==Q

```

```

%-----
%verificando se a primeira linha é nula

```

```

    ln1=0;
    for ln=1:j3
        if M2(1,ln)==0
            ln1=ln1+1;
        else
            break
        end
    end
    if ln1==j3
        M3=M2(2:i3,:);
        [i4,j4]=size(M3);
    end

```

```

%-----
%Verificando se existe alguma linha linearmente dependente

```

```

    else
        for ti=1:i3
            for ti2=1:i3
                if ti==ti2
                    break
                elseif M2(ti,:)==M2(ti2,:)
                    M3=[M2(1:ti2-1,:);M2(ti2+1:i3,:)];
                    [i4,j4]=size(M3);
                end
            end
        end
    end

```

```

    if i4==i3-1
        M2=M3;
    end
    [i3,j3]=size(M2);
    [c1,c2]=size(M2(1,2:j3));

```

```

v=M2(1,2:j3);
f=find(v<=0);
f=v(1,f);
[c3,c4]=size(f);
ite=ite+1;
end

%%Aqui termina o quadro alternativo (Simplex)
%-----
_e=M2(:,1);
e1=fix(M2(:,1));

if e==e1
    %Fazendo outro pivoteamento para quando o RHS for negativo
    [i6,j6]=size(M2);
    [i8,j8]=find(M2(2:i6,1)<0);           %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m1=size(m);
    [i8,j8]=find(M2(1,2:j6)>0);           %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m2=size(m);

    while (m1~=0) | (m2~=0)                %Acontecer isso até solução
ser ótima

        if m1~=0
            %Condição das iterações
            [i6,j6]=size(M2);
            [c1,c2]=size(M2(2:i6,1));
            v=M2(2:i6,1);
            f=find(v>=0);
            f=v(f,1);
            [c3,c4]=size(f);
            while c1~=c3                    %PIVOTEMENTO DA SEGUNDA
FASE (DUAL)
                m=min(M2(2:i6));           %m menor valor de "RHS"
                [l,j7]=find(M2(2:i6,1)<(m+0.00001)); %l(linha do pivo) linha
do menor valor de "RHS"
                l=l+1;                     %para compensar a linha
de custos
                l=max(l);
                %Escolhendo a coluna do pivo
                C=M2(1,2:j6) ./M2(1,2:j6); %C teste da razão mínima
                C1=find(C>0);               %C1 valores positivos de "C" (vetor
posicao)
                C2=C(1,C1);                 %C2 vetor dos valores positivos de
"C"
                C3=min(C2);                 %C3 menor valor positivo de "C"
                C1=find(C==C3);             %C1 posicao do menor valor de "d"
(coluna
                %que entra na base)
                C1=max(C1);                 %Qd tem dois valores iguais
escolhemos o de maio indice
                k=C1+1;
                pp=[l k];
                %fazendo o pivoteamento uma iteração a mais
                for p2=1:j6
                    if k~=p2
                        for p1=1:i6

```

```

        if l~=p1
            M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(l,p2)/M2(l,k);
        end
    end
end
end

M2(1,:)=M2(1,:)/M2(1,k);
M2=[M2(:,1:k-1),M2(:,k+1:j6)];

%condição das iterações
[i6,j6]=size(M2);
[c1,c2]=size(M2(2:i6,1));
v=M2(2:i6,1);
f=find(v>=0);
f=v(f,1);
[c3,c4]=size(f);
ite=ite+1;
end
end

% -----

[i8,j8]=find(M2(1,2:j6)>0); %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m2=size(m);

    if m2~=0
        %Condição das iterações
        [c1,c2]=size(M2(1,2:j6));
        v=M2(1,2:j6);
        f=find(v<=0);
        f=v(1,f);
        [c3,c4]=size(f);

        while c2~=c4 %PIVOTEAMENTO DA SEGUNDA FASE NORMAL
            [u1,u2]=find(M2(1,2:j6)>0);
            u2=u2+1;
            v1=M2(1,u2);
            v1=max(v1);
            v1=find(v1==M2(1,2:j6));
            v1=min(v1);
            k=v1+1;
            %Escolhendo a coluna do pivo
            C=M2(2:i6,1)./M2(2:i6,k); %C teste da razão mínima
            C1=find(C>0); %C1 valores positivos de "C" (vetor
posicao) %C2 vetor dos valores positivos de
"C"
            C2=C(C1,1);

            if C2==inf
                aux_itel=1;
                aux_teste21;
                break
            end

            C3=min(C2); %C3 menor valor positivo de "C"
            C1=find(C==C3); %C1 posicao do menor valor de "d"
(coluna %que entra na base)

```



```

        C1=max(C1); %Qd tem dois valores iguais
escolhemos o de maio indice
        l=C1+1;
        pp=[l k];

        %fazendo o pivoteamento uma iteração a mais
        for p2=1:j6
            if k~=p2
                for p1=1:i6
                    if l~=p1
                        M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(l,p2)/M2(l,k);
                    end
                end
            end
        end

        M2(l,:)=M2(l,+)/M2(l,k);
        M2=[M2(:,1:k-1),M2(:,k+1:j6)];
        [i6,j6]=size(M2);
        [c1,c2]=size(M2(1,2:j6));
        v=M2(1,2:j6);
        f=find(v<=0);
        f=v(1,f);
        [c3,c4]=size(f);
    end
end
[i6,j6]=size(M2);
[i8,j8]=find(M2(2:i6,1)<0); %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m1=size(m);
    [i8,j8]=find(M2(1,2:j6)>0); %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m2=size(m);
end

else
    %Preparação para Branch and Bound, identificando as variáveis básicas

    H=M2(2:i3,2:j3);
    H=-inv(H);
    [i4,j4]=size(H);
    [i5,j5]=size(N);
    a=[]; %vetor dos índices das variáveis básicas
    aux=0;
    q=j4; %quantidade de linhas da matriz identidade
    r=j5; %numero de colunas de AI
    rhs=M2(2:i3,1);
    for q1=1:q;
        for r1=1:r;
            if N(:,r1)-H(:,q1)==0;
                x=r1;
                aux=aux+1;
                w=sprintf('X%d = %d',x,rhs(aux,1));
                a(1,aux)=r1;
            end
        end
    end
end
end
M2;

```



```

M3(i1,:)=M3(i1,+)/-M2(i1,j2);
M3(:,j2)=M3(:,j2)/M2(i1,j2);
M3(i1,j2)=pii;
M5=M3;
%* * * * *
%Adicionando uma restrição de <=
%i=1; %variável para decisão de escolha de qual variável
básica trabalhar
ba=0;
k1=find(i==a);
ab=[];
ab(1,k1)=1;
ab1=zeros(1,i4-k1);
ab=[ab ab1];
an=zeros(1,j5-j4); %N° de variáveis não básicas j5 colunas de N e j4
colunas de B
%Atualizando a nova restrição
Cb1=an-ab*(-inv(H)); %H é igual a inv(B)*N
b1=ba-ab*rhs; %rhs é igual a inv(B)*b
nl=[b1 Cb1];
M2(i3+1,:)=nl;

%Resolvendo novamente o problema
%Localizando a linha do pivot
rhsm=min(M2(2:i3+1,1));
[i1,j1]=find(rhsm==M2(2:i3+1,1));
i1=i1+1;

%Localizando a coluna do pivot
d=M2(1,2:j3)./M2(i1,2:j3);
[lp,cp]=find(d>0);
av=d(1,cp);
av=min(av);
[i2,j2]=find(av==d);
j2=max(j2);
j2=j2+1;
pp=[i1 j2];

pi=M2(i+1,j2);
pii=inv(pi);
M3=M2;
for p2=1:j3
    if j2~=p2
        for p1=1:i3
            if i1~=p1
                M3(p1,p2)=M3(p1,p2)-M3(p1,j2)*M3(i1,p2)/M3(i1,j2);
            end
        end
    end
end
ite=ite+1; %Terminou outra iteração
if M2(i1,j2)<0
    M2(i1,j2)=-M2(i1,j2);
end

M3(i1,:)=M3(i1,+)/-M2(i1,j2);
M3(:,j2)=M3(:,j2)/M2(i1,j2);
M3(i1,j2)=pii;

```



```

M6=M3;
if M5(1,1)<M6(1,1)
    M3=M5;
    M3
else
    M3
end
end
M2
ite

```

2- Código Fonte do Programa Dual

```

%Aluno: Tercio A. Santos
%Resolvendo o dual no quadro primal
clear
clc
% -----
% -----
[i3,j1]=size(A); %i3 numero de linhas de A=numero de
%PREPARANDO O A E b PARA O QUADRO DUAL
A(:,j1+1)=ones(i3,1);
C=rref(A);
b=C(:,j1+1);
A=C(:,1:j1);
b=-b;
A=-A;
%Adicionando as variáveis de folga:
[i3,j1]=size(A); %i3 numero de linhas de A=numero de

%restricoes tambem igual a ordem da bse igual a ordem da identidade

AI=[A,eye(i3)]; %AI a matriz A acrescida da identidade I
%Determinando as variáveis da base:
B=eye(i3); %B matriz identidade
cb=zeros(1,i3); %cb coeficientes de custos basicos (dado
%o numero de variavel de folga)

%Determinar as variáveis não básica:
ci=zeros(1,i3); %Variavel que auxilia o calculo de CC
CC=[c,ci]; %Coeficientes de custos
N=A; %Matriz dos coeficientes das restricoes
%(variaveis nao basicas)
cn=c; %cn Coeficientes de custos nao basicos

%1 Preparacao para iniciar o simplex

%Determinar os custos relativos cr:
Cr=cb*inv(B)*N-cn; %Cr coeficientes dos custos relativos
% -----
bt=inv(B)*b; %Cálculo do b traço (rhs)
a=size(bt); %a vetor da ordem de bt [1 c]
a1=find(bt>=0); %a1 vetor posicao dos elementos positivos de
bt
a2=bt(a1,1); %a2 vetor formado pelos valores positivos de
bt
f=size(a2); %f tamanho de a2
Ite=0; %Ite iteracao inicial
% -----

ac=size(Cr);

```

```

alc=find(Cr<=0);
a2c=Cr(1,alc);
fc=size(a2c);
% -----

%Procedimento para preparacao do simplex
%Comparar a quantidade de linhas de "a" e "f", se o tamanho de a=f
%indica que todos os "bt" sao positivos "a(1,1)~=f(1,1)"
while (a(1,1)~=f(1,1)) | ac(1,2)~=fc(1,2)

    while a(1,1)~=f(1,1)

        Yr=inv(B)*N;                %Yr e o "N" atualizado com a base "B"

        %Determinando quem sai da base:
        m=min(bt);                  %m menor valor de "bt"
        [i,j]=find(bt<(m+0.00001)); %i linha do menor valor de "bt"
        s=Yr(min(i,:),:);           %s e a linha de "Yr" que sai da base
        i=min(i);                   %min(i) menor indice entre dois
valores                             %iguais da linha "i"

        %Determinando quem entra na base:
        is=size(s);                 %is vetor das dimensões de s
        i5=find(s>=0);              %i5 vetor com as componentes negativas
de s

        h=s(1,i5);                 %h vetor com os termos positivos de s
        ih=size(h);                %ih tamanho de "h"

        d=Cr./s;                    %d teste da razão mínima
        il=find(d>0);               %il valores positivos de "d" (vetor
posicao)

        x=d(1,il);                  %x vetor dos valores positivos de "d"
        n=min(x);                   %n menor valor positivo de "x"
        il=find(d==n);              %il posicao do menor valor de "d" (coluna
que entra na base)
        il=min(il);
        % -----
        %Nova Base
        e=AI(:,min(il));            %e coluna que entra na base
        A1=B(:,1:i-1);              %A1 colunas que antecede a coluna que
entra na base
        A2=B(:,i+1:i3);             %A2 colunas que estao apos a coluna que
entra na base
        B=[A1,e,A2];                %a nova base

        %Novo Cb
        s1=CC(:,min(il));            %s1 menor custo positivo
        cb1=cb(1,1:i-1);            %cb1 colunas que antecede o menor custo
positivo
        cb2=cb(1,i+1:i3);           %cb2 colunas que estao apos o menor
custo positivo
        cb=[cb1,s1,cb2];            %novo "cb" custo basico

        %Novo N
        n1=AI(:,j1+min(i));          %
        N1=N(:,1:i1-1);
        N2=N(:,i1+1:j1);
        N=[N1,n1,N2];

        %Determinando os coeficientes de custos não-básicos cn

```

```

CN=CC(1,j1+min(i));
cn1=cn(1,1:i1-1);
cn2=cn(1,i1+1:j1);
cn=[cn1,CN,cn2];

%Condição das iterações
Cr=cb*Yr-cn;

bt=inv(B)*b;           %Cálculo do b traço
a=size(bt);           %a vetor da ordem de bt [1 c]
a1=find(bt>=0);       %%a1 vetor posicao dos elementos
positivos de bt       %a2 vetor formado pelos valores
a2=bt(a1,1);
positivos de bt       %f tamanho de a2
f=size(a2);           %numero de iteracoes
Ite=Ite+1;
end

%-----
%Determinar os custos relativos cr:
Cr=cb*Yr-cn;
ac=size(Cr);
alc=find(Cr<=0);
a2c=Cr(1,alc);
fc=size(a2c);

while ac(1,2)~=fc(1,2)

%Determinando os Yr:
Yr=inv(B)*N;

%Determinando quem entra na base:
m=max(Cr);
[i,j]=find(Cr>(m-0.00001));
k=min(j);
s=Yr(:,k);
b1=inv(B)*b;           %Cálculo do b barrado
%Determinando quem sai da base:
is=size(s);           %vetor das dimensões de s
i5=find(s<0);         %vetor com as componentes negativas de

s
h=s(i5,1);
ih=size(h);
if is==ih;
    disp('ILIMITADO')
else
    d=b1./s;           %Teste da razão mínima
    i1=find(d>0);
    x=d(i1,:);
    n=min(x);
    j2=find(n);
    i1=find(d==n);
    i1=min(i1);
end
%Nova Base
e=AI(:,k);
A1=B(:,1:i1-1);
A2=B(:,i1+1:i3);
B=[A1,e,A2];
%Novo Cb
s1=c(:,k);
cb1=cb(1,1:i1-1);

```



```

cb2=cb(1,i1+1:i3);
cb=[cb1,s1,cb2];

%Novo N
n1=AI(:,j1+min(i1));
N1=N(:,1:k-1);
N2=N(:,k+1:j1);
N=[N1,n1,N2];

%Determinando os coeficientes de custos não-básicos cn
cn1=cn(1,1:k-1);
cn2=cn(1,k+1:j1);
CN=CC(1,j1+min(i1));
cn=[cn1,CN,cn2];

%Condição das iterações
Cr=cb*Yr-cn;
ac=size(Cr);
alc=find(Cr<=0);
a2c=Cr(1,alc);
fc=size(a2c);
Ite=Ite+1;
end
%Atualizando as condições do primeiro While

Cr=cb*Yr-cn; %Cr coeficientes dos custos relativos
bt=inv(B)*b; %Cálculo do b traço (rhs)
a=size(bt); %a vetor da ordem de bt [1 c]
a1=find(bt>=0); %a1 vetor posicao dos elementos positivos
de bt %a2 vetor formado pelos valores positivos
a2=bt(a1,1);
de bt %f tamanho de a2
f=size(a2);

ac=size(Cr);
alc=find(Cr<=0);
a2c=Cr(1,alc);
fc=size(a2c);
end

Ite
b_traco=inv(B)*b; %vetor rhs
FO=cb*bt %funcao objetivo

%descrição das variáveis
aux=0;
q=i3; %quantidade de linhas da matriz identidade
r=i3+j1; %numero de colunas de AI
for q1=1:q;
    for r1=1:r;
        if AI(:,r1)-B(:,q1)==0;
            x=r1;
            aux=aux+1;
            w=sprintf('X%d = %d',x,b_traco(aux,1));
            disp(w)
        end
    end
end
end

```

3- Código Fonte do Programa Gomo2

```
%Tabela do Quadro Alternativo ou Metodo de Gomory
clear
clc

%-----
%O PL e resolvido pelo método de duas fases
[i,j]=size(A);          %i3 numero de linhas de A=numero de

%PREPARANDO O A E b PARA O QUADRO ALTERNATIVO
A(:,j+1)=ones(i,1);
C=rref(A);
b=C(:,j+1);
A=C(:,1:j);

%Fórmulas de preparação
[i,j]=size(A);          %i, j números de linhas e colunas
respectivamente da matriz A

%b=ones(i,1);           %b e a coluna das restricoes
B=eye(i);               %B e a base representado pela matriz
identidade
AI=[A,-eye(i),eye(i)];  %AI coeficientes das restricoes acrescida da
matriz
%identidade

[i1,j1]=size(AI);       %i1, j1 n° de linhas e colunas
respectivamente da matrix AI
N=[A,-eye(i)];           %N e a matriz formada pelos coeficientes das
restricoes
%acrescidada das variaveis de folga
[i2,j2]=size(N);        %i2, j2 n° de linhas e colunas
respectivamente da matrix N
cn=zeros(1,j2);         %cn custos nao basicos formada pelos
coeficientes das
%variaveis nao basicas

%Custo relativo da função objetivo I
cb=ones(1,i);            %cb coeficientes das variaveis baseicas
Cr=cb*inv(B)*N-cn;        %Cr custos relativos da funcao objetivo I
FO=cb*inv(B)*b;          %FO valor da funacao objetivo I

%Custo relativo da função objetivo II
cb2=zeros(1,i);          %cb coeficientes das variaveis baseicas da
fase II
cn2=[c,zeros(1,i)];      %cn custos nao basicos formada pelos
coeficientes das
%variaveis nao basicas da fase II
Cr2=cb2*inv(B)*N-cn2;    %Cr custos relativos da funcao objetivo II
FO2=cb2*inv(B)*b;        %FO valor da funacao objetivo II

%Criando a matriz M2
[i5,j5]=size(N);         %[i5,j5] tamanho de N
M2=zeros(i5+2,j5+1);     %M2 matriz nula para auxiliar a formacao da
tabela
M2(1,2:j5+1)=Cr;         %M2(1,2:j5+1) acrescida da linha Cr
M2(2,2:j5+1)=Cr2;        %M2(2,2:j5+1) acrescida da linha Cr2
```

```

M2(3:i5+2,2:j5+1)=N;      %M2(3:i5+2,2:j5+1) acrescida da matriz N
M2(3:i5+2,1)=b;           %M2(3:i5+2,1) acrescida de b
M2(1,1)=FO;               %M2(1,1) acrescida da funcao objetivo FO da
fase I                    %M2(2,1) acrescida da funcao objetivo FO2 da
M2(2,1)=FO2;              fase II

```

```
%teste das iterações
```

```

[i3,j3]=size(M2);          %i3, j3 n° de linhas e colunas
respectivamente da matrix M2
[c1,c2]=size(M2(1,2:i3));
v=M2(1,2:i3);
f=find(v<=0);
f=v(1,f);
[c3,c4]=size(f);
ite=0;
aux_itel=0;                %Variável de decisão de aux_teste
while c4~=c2

    [i3,j3]=size(M2);      %i3, j3 n° de linhas e colunas
    respectivamente da matrix M2

    %Encontrando o maior valor da FO II
    g=max(M2(1,2:j3));
    for k=2:j3              %O n° de repetições do laço é o n° da coluna
        pivot k
        y=M2(1,k);
        if g==y
            break
        end
    end

    %Encontrando quem sai da base
    d=M2(3:i3,1)./M2(3:i3,k); %d coluna do pivot
    w=find(d>0);
    w=d(w);
    if w==inf
        aux_itel=2;
        aux_teste21;
        break
    elseif d<=0
        aux_itel=2;
        aux_teste21;
        break
    else
        l=d(find(d>0));
        l=min(l);
        l=find(l==d);
        l=min(l);
        l=l+2;
        pp=[l k];           %endereço do elemento pivot
    end

    %Fazendo o pivoteamento
    for p2=1:j3
        if k~=p2
            for p1=1:i3
                if l~=p1

```



```

        M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(1,p2)/M2(1,k);
    end
end
end
end

M2(1,:)=M2(1,:)/M2(1,k);
M2=[M2(:,1:k-1),M2(:,k+1:j3)];

%Verificando se A é a matriz identidade
i4=0;
[i3,j3]=size(M2);
if i==j
    Q=ones(i,j);
    Q1=A*Q;

else
    Q1=1;
    Q=2;
end
if Q1==Q
%-----
%verificando se a primeira linha é nula
    ln1=0;
    for ln=1:j3
        if M2(1,ln)==0
            ln1=ln1+1;
        else
            break
        end
    end
    if ln1==j3
        M3=M2(2:i3,:);
        [i4,j4]=size(M3);
    end
%-----
%Verificando se existe alguma linha linearmente dependente
    else
        for ti=1:i3
            for ti2=1:i3
                if ti==ti2
                    break
                elseif M2(ti,:)==M2(ti2,:)
                    disp('LINHAS LINEARMENTE DEPENDENTES FORAM
ENCONTRADAS!!!!')
                    M3=[M2(1:ti2-1,:);M2(ti2+1:i3,:)];
                    [i4,j4]=size(M3);
                end
            end
        end
    end

    if i4==i3-1
        M2=M3;
    end
    [i3,j3]=size(M2);
    [c1,c2]=size(M2(1,2:j3));
    v=M2(1,2:j3);
    f=find(v<=0);

```

```

f=v(1,f);
[c3,c4]=size(f);
ite=ite+1;
end

%%Aqui termina o quadro alternativo (Simplex)
%-----

e=M2(:,1);
e1=fix(M2(:,1));

if e==e1
    %Fazendo outro pivoteamento para quando o RHS for negativo
    [i6,j6]=size(M2);
    [i8,j8]=find(M2(2:i6,1)<0);           %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m1=size(m);
    [i8,j8]=find(M2(1,2:j6)>0);           %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m2=size(m);

    while (m1~=0) | (m2~=0)               %Acontecer isso até solução
ser ótima

        if m1~=0
            %Condição das iterações
            [i6,j6]=size(M2);
            [c1,c2]=size(M2(2:i6,1));
            v=M2(2:i6,1);
            f=find(v>=0);
            f=v(f,1);
            [c3,c4]=size(f);
            while c1~=c3                   %PIVOTEMENTO DA SEGUNDA
FASE (DUAL)
                m=min(M2(2:i6));           %m menor valor de "RHS"
                [l,j7]=find(M2(2:i6,1)<(m+0.00001)); %l(linha do pivô) linha
do menor valor de "RHS"
                l=l+1;                     %para compensar a linha
de custos
                l=max(l);
                %Escolhendo a coluna do pivô
                C=M2(1,2:j6)./M2(1,2:j6);   %C teste da razão mínima
                C1=find(C>0);               %C1 valores positivos de "C" (vetor
posicao)
                C2=C(1,C1);                %C2 vetor dos valores positivos de
"C"
                C3=min(C2);                %C3 menor valor positivo de "C"
                C1=find(C==C3);             %C1 posicao do menor valor de "d"
(coluna
                %que entra na base)
                C1=max(C1);                %Qd tem dois valores iguais
escolhemos o de maior indice
                k=C1+1;
                pp=[l k];
                %fazendo o pivoteamento uma iteração a mais
                for p2=1:j6
                    if k~=p2
                        for p1=1:i6
                            if l~=p1

```

```

        M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(1,p2)/M2(1,k);
    end
end
end
end

M2(1,:)=M2(1,:)/M2(1,k);
M2=[M2(:,1:k-1),M2(:,k+1:j6)];

%condição das iterações
[i6,j6]=size(M2);
[c1,c2]=size(M2(2:i6,1));
v=M2(2:i6,1);
f=find(v>=0);
f=v(f,1);
[c3,c4]=size(f);
ite=ite+1;
end
end

% -----
[i8,j8]=find(M2(1,2:j6)>0); %i8 linha do termo negativo
do RHS
m=M2(i8,j8);
m2=size(m);

if m2~=0
    %Condição das iterações
    [i6,j6]=size(M2);
    [c1,c2]=size(M2(1,2:j6));
    v=M2(1,2:j6);
    f=find(v<=0);
    f=v(1,f);
    [c3,c4]=size(f);

    while c2~=c4 %PIVOTEAMENTO DA SEGUNDA FASE NORMAL
        [u1,u2]=find(M2(1,2:j6)>0);
        u2=u2+1;
        v1=M2(1,u2);
        v1=max(v1);
        v1=find(v1==M2(1,2:j6));
        v1=min(v1);
        k=v1+1;
        %Escolhendo a coluna do pivo
        C=M2(2:i6,1)./M2(2:i6,k); %C teste da razão mínima
        C1=find(C>0); %C1 valores positivos de "C" (vetor
posicao)
        C2=C(C1,1); %C2 vetor dos valores positivos de
"C"

        if C2==inf
            aux_itel=1;
            aux_teste21;
            break
        end

        C3=min(C2); %C3 menor valor positivo de "C"
        C1=find(C==C3); %C1 posicao do menor valor de "d"
(coluna
%que entra na base)

```



```

        C1=max(C1);                                %Qd tem dois valores iguais
escolhemos o de maio indice
        l=C1+1;
        pp=[l k];

        %fazendo o pivoteamento uma iteração a mais
        for p2=1:j6
            if k~=p2
                for p1=1:i6
                    if l~=p1
                        M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(l,p2)/M2(l,k);
                    end
                end
            end
        end

        M2(l,:)=M2(l,:)/M2(l,k);
        M2=[M2(:,1:k-1),M2(:,k+1:j6)];
        [i6,j6]=size(M2);
        [c1,c2]=size(M2(1,2:j6));
        v=M2(1,2:j6);
        f=find(v<=0);
        f=v(1,f);
        [c3,c4]=size(f);

    end
end
[i6,j6]=size(M2);
[i8,j8]=find(M2(2:i6,1)<0);                    %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m1=size(m);
    [i8,j8]=find(M2(1,2:j6)>0);                %i8 linha do termo negativo
do RHS
    m=M2(i8,j8);
    m2=size(m);
end

else

%PLANO DE CORTE DE GOMORY

[i,j]=size(M2);                                %[i,j] ordem da matriz M2
rhs=M2(2:i,1);
rhsi=fix(rhs);
rhsf=rhs-rhsi;
mf=max(rhsf);
h=find(mf==rhsf);
h=min(h);
S=[];                                           %S linha formada por elementos nulos,
local
    %onde devera ser acrescida a linha do plano de corte

    for n=1:j
        if M2(h+1,n)<0
            S(1,n)=M2(h+1,n)-0.9;
            S(1,n)=-fix(S(1,n));
            S(1,n)=S(1,n)+M2(h+1,n);
        elseif M2(h+1,n)>=0
            S(1,n)=fix(M2(h+1,n));
            S(1,n)=M2(h+1,n)-S(1,n);
        end
    end
end

```

```

end
S=-S; %S linha que devera ser acrescida na
matriz M2 %que e a solucao do metodo de duas fases

M2(i+1,:)=S;

%Localizando a linha do pivot
rhsm=min(M2(2:i+1,1)); %rhsm menor valor do elemento de rhs
[i1,j1]=find(rhsm==M2(2:i+1,1));
i1=i1+1;

%Localizando a coluna do pivot
d=M2(i1,2:j)./M2(1,2:j); %d coluna do pivot
[t1 t2]=find(d>0);
rm=d(t1,t2);
rm=min(rm);
rm=rm(1,1);
[i2,j2]=find(rm==d);
j2=min(j2+1);
pp=[i1 j2];
pi=M2(i+1,j2);
pii=inv(pi);
M3=M2;
for p2=1:j
    if j2~=p2
        for p1=1:i+1
            if i1~=p1
                M3(p1,p2)=M3(p1,p2)-M3(p1,j2)*M3(i1,p2)/M3(i1,j2);
            end
        end
    end
end
ite=ite+1;
if M2(i1,j2)<0
    M2(i1,j2)=-M2(i1,j2);
end

M3(i1,:)=M3(i1,:)/-M2(i1,j2);
M3(:,j2)=M3(:,j2)/M2(i1,j2);
M3(i1,j2)=pii;
end
M3=M2;
M3
ite
disp('GOMO2')

```

4- Função Auxiliar (Branch2 e Gomo2)

%Programa auxiliar para contornar o problema de ILIMITADO

```

%-----
%
if aux_itel==1 %Variável de decisão
    var=1;
elseif aux_itel==2
    var=2;
end
[i3,j3]=size(M2);
M4=zeros(i+var,j3+i); %
M4(:,1)=M2(:,1);
M4(1:var,2:i+1)=zeros(1:var,1:i);

```

```

M4(var+1:i+var,2:i+1)=eye(i);
M4(:,i+2:j3+i)=M2(:,2:j3);
M2=M4;

while c2~=c4
%Inserindo a matriz identidade na matriz M2

[i3,j3]=size(M2);
%Encontrando o maior valor da FO II
g=max(M2(1,2:j3));
for k=2:j3 %O n° de repetições do laço é o n° da coluna
pivot k
    y=M2(1,k);
    if g==y
        break
    end
end
%encontrando quem sai da base
q1=find(M2(3:i3,k)>0);
q2=M2(q1+2,k);
[q4,q5]=size(q1);
if q4==1
    l=q1+2;
    pp=[1 k];
else
    l=max(q1);
    l=l+2;
    pp=[1 k];
end
% -----
% -----

%Fazendo o pivoteamento
for p2=1:j3
    if k~=p2
        for p1=1:i3
            if l~=p1
                M2(p1,p2)=M2(p1,p2)-M2(p1,k)*M2(l,p2)/M2(l,k);
            end
        end
    end
end
end

M2(1,:)=M2(1,:)/M2(1,k);
M2=[M2(:,1:k-1),M2(:,k+1:j3)];

%Verificando se A é a matriz identidade
i4=0;
[i3,j3]=size(M2);

if i==j
    Q=ones(i,j);
    Q1=A*Q;
else
    Q1=1;
    Q=2;
end

    if Q1==Q
%verificando se a primeira linha é nula

```



```

ln1=0;
for ln=1:j3
    if M2(1,ln)==0
        ln1=ln1+1;
    else
        break
    end
end
if ln1==j3
    M3=M2(2:i3,:);
    [i4,j4]=size(M3);
end
% -----
%Verificando se existe alguma linha linearmente dependente
else
    for ti=1:i3
        for ti2=1:i3
            if ti==ti2
                break
            elseif M2(ti,:)==M2(ti2,:)
                disp('linhas linearmente dependentes foram
encontradas!!!!')
                M3=[M2(1:ti2-1,:);M2(ti2+1:i3,:)];
                [i4,j4]=size(M3);
            end
        end
    end
end

if i4==i3-1
    M2=M3;
end
[i3,j3]=size(M2);
[c1,c2]=size(M2(1,2:j3));
v=M2(1,2:j3);
f=find(v<=0);
f=v(1,f);
[c3,c4]=size(f);
ite=ite+1;
%Tirando a matriz identidade da M2
if c2==c4
    M4=zeros(i3,j3-i);
    M4(:,1)=M2(:,1);
    M4(:,2:j3-i)=M2(:,i+2:j3);
    M2=M4;
    L1=i+2;
    if L1==i3
        [L1,K1]=size(M2);
        M4=[];
        M4(:, :)=M2(2:L1, :);
        M2=M4;
        L1=0;
    end
end
end
end

```


BRANCH AND BOUND APLICADO NO PROBLEMA DE COBERTURA DE FUNÇÕES BOOLEANAS

Tercio Alberto dos Santos¹, Alexandre César Rodrigues da Silva² e Carlos Eduardo da Silva Santos³

Resumo — Este trabalho apresenta um método para a obtenção de cobertura mínima para uma função booleana empregando o algoritmo Branch and Bound modificado. A partir da formulação do problema de cobertura de funções booleanas como um problema de programação linear inteira 0 e 1, cuja formulação é apresentada a seguir, a solução ótima global é obtida.

MIN CT. X de modo que

$$AX \geq b$$

$$X \geq 0$$

Onde: $X \in R^n$; $C \in R^n$; $b \in R^n$ e $A \in R^{n \times n}$ com posto completo. Vários métodos para a geração de implicantes primos podem ser utilizados para a geração da formulação da cobertura. Empregou-se neste trabalho o algoritmo Expander que utiliza-se do método de expansão de Shannon.

Uma grande variedade de funções booleanas foram estudadas, cujos resultados comprovaram a eficiência computacional deste tipo de abordagem. O algoritmo implementado em linguagem de programação está sendo comparado com outros algoritmos já implementados.

Palavras chave — Programação Matemática, Otimização de Funções Booleanas, Branch and Bound, Plano de Corte, Dual Simplex.

INTRODUÇÃO

A otimização lógica a dois níveis tem suas raízes há muitas décadas atrás e está baseada na teoria clássica sobre a álgebra de chaveamento. Vários textos tratam do assunto [1], [2], [3]. Os primeiros algoritmos para a minimização exata foram propostos por Quine [4] e por McCluskey [2]. Uma grande variedade de métodos foram propostos ao longo dos anos como, por exemplo, o método de Rudell e Sangiovanni [5] denominado Espresso-Exact, o método de Dagenais et al [6] denominado de McBoole, o método de Coudert e Madre [7] e muitos outros.

Com o aumento da complexidade dos circuitos digitais a minimização lógica exata tornou-se impraticável até algumas décadas atrás. Com o aumento da velocidade e da capacidade de memória das estações de trabalho, aliado ao desenvolvimento de eficientes métodos de cobertura para uma solução exata, revitalizou-se o interesse por essa área de pesquisa.

Atualmente, tem-se como um dos mais eficientes algoritmos de minimização de funções booleana o programa denominado BOOM (A

Heuristic Boolean Minimizer) [8], cuja principal característica é a combinação da geração dos implicantes primos em conjunto com a solução do problema de cobertura, o que leva a uma redução no número de implicantes primos gerados.

Este trabalho trata sobre a resolução do problema de cobertura de funções booleanas formulada como um problema de programação matemática. Implementou-se três algoritmos descritos na literatura com o objetivo de avaliar o que melhor soluciona o problema de cobertura. Foram implementados o algoritmo Branch and Bound, o Plano de Corte de Gomory e o Dual Simplex. Utilizou-se como critério de eficiência a quantidade de memória utilizada, o tempo de execução, a quantidade de iterações e a solução obtida.

PROBLEMA DE COBERTURA

Toda cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1, dessa forma todos os avanços da área da programação matemática podem ser utilizados com o objetivo de solucioná-la. Muitos métodos clássicos podem ser empregados, porém nesse trabalho implementou-se três métodos, o Branch and Bound, o Plano de Corte de Gomory e o Dual Simplex Canalizado [9].

O método Simplex é a base para o Plano de Corte de Gomory e para o Branch and Bound. O Dual Simplex Canalizado é uma forma alternativa para solucionar o problema de cobertura sem a necessidade de se utilizar o método Simplex.

Resolver o problema de cobertura consiste em encontrar uma solução que satisfaça todas as restrições do problema e que o custo da função objetivo, que representa o custo de implementação da função booleana, seja o menor possível.

Na grande maioria dos exemplos tratados neste trabalho obteve-se solução inteira no conjunto binário 0 e 1. Contudo, em alguns casos encontrou-se uma solução otimista (casos cíclicos). A identificação de casos cíclicos tornou-se possível, pois a solução é fracionária. Para esses casos empregou-se o método Branch and Bound ou Plano de Corte de Gomory para encontrar a solução ótima global do problema.

O algoritmo de Branch and Bound é um algoritmo enumerativo, cuja estrutura de resolução baseia-se na construção de uma árvore onde os nós representam os problemas candidatos e os ramos representam as novas restrições que devem ser

consideradas. Por intermédio dessa árvore, todas as soluções inteiras da região viável do problema são enumeradas de modo implícito ou explícito o que garante que todas as soluções ótimas serão encontradas.

O algoritmo de Plano de Corte [10] é um dos principais algoritmos de programação linear inteira, o seu uso é pós otimização de PL internamente. Nesse algoritmo de uma iteração para outra se resolve um PL com uma restrição adicional. Assim, os PL's sucessivos são diferentes em uma restrição adicional. Portanto, a estratégia mais evidente é usar dual simplex que aproveita o quadro ótimo de um PL para resolver o PL seguinte com a simples adição no quadro da nova restrição corretamente atualizada para a base atual.

O algoritmo dual simplex canalizado é o mais adequado para reotimizar subproblemas de programação linear (PL) gerados pelos algoritmos de Branch and Bound na resolução de um problema de programação linear inteira.

No algoritmo de Branch and Bound cada problema (PL) é diferente de seu antecessor em uma restrição adicional da seguinte forma:

$$x_j \geq k + 1$$

ou

$$x_j \leq k$$

Sendo k um número inteiro.

Embora as restrições do tipo β_1 e β_2 podem ser adicionadas como uma restrição padrão e usar um algoritmo dual simplex padrão, essa estratégia aumenta sem necessidade o tamanho do quadro e da base. Como as restrições β_1 e β_2 são restrições sobre uma única variável então a melhor estratégia é usar um algoritmo dual simplex canalizado que leva em conta esse tipo de restrições somente de maneira implícita.

Vários métodos podem ser empregados para a formulação do problema de cobertura. Neste trabalho utilizou-se o algoritmo Expander [11] que implementa Método de Expansão de Shannon.

O MÉTODO DE EXPANSÃO DE SHANNON

A geração de implicantes primos pelo Método de Expansão de Shannon é uma técnica iterativa que permite obter implicantes primos de uma dada função booleana através de simples operações sobre o conjunto de mintermos e/ou estados "don't care", representados em notação decimal, que descreve a função original. Tal método pode ser empregado em função com um grande número de variáveis. Todos ou, quase todos os implicantes primos que não estariam no conjunto solução são eliminados, simplificando, desse modo, a busca pela solução mínima.

Como exemplo considere a função booleana $f(A,B,C,D) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11,$

14). Para a formulação do problema de cobertura utilizou-se o algoritmo Expander cujo arquivo de entrada e de saída estão apresentados na Figura 1 e na Figura 2, respectivamente. A tabela de cobertura proposta por McCluskey, consiste de colunas contendo todos os mintermos da função e de linhas que representam todos os implicantes primos gerados. A Tabela I apresenta a tabela de cobertura de McCluskey cujos implicantes primos foram gerados pelo algoritmo Expander.

Com o auxílio da Tabela I, o problema de cobertura dos mintermos transforma-se em um problema de programação linear inteiro binário 0 e 1.

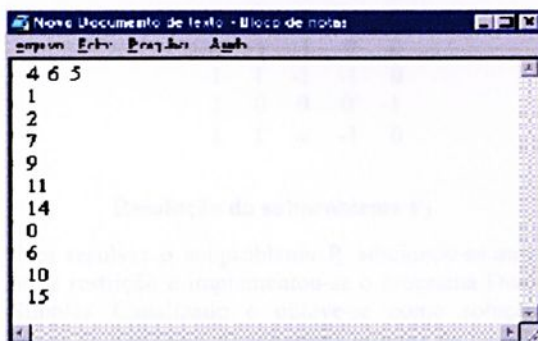


FIGURA. 1

TELA DO ARQUIVO DE ENTRADA DO PROGRAMA EXPANDER.



FIGURA. 2

TELA DO ARQUIVO DE SAÍDA DO PROGRAMA EXPANDER.

TABELA I
TABELA DE COBERTURA DE MCCLUSKEY CUJOS
IMPLICANTES PRIMOS FORAM GERADOS PELO ALGORITMO
EXPANDER

Implicantes	Mintermos				
	0	6	8	10	15
XX10		x		x	
XIX				x	x
X11X		x			x
X00X	x		x		

Optou-se por tratar o problema de cobertura como um problema matemático. Dessa forma,

empregou-se uma notação matemática mais simples.

Fazendo-se: $x_1 = XX10$, $x_2 = 1X1X$, $x_3 = X11X$ e $x_4 = X00X$ o problema apresentado na Tabela I é formulado como um problema de programação linear inteira 0 e 1 como descrito a seguir [12]:

$$\text{Min } x_0 = 3x_1 + 3x_2 + 3x_3 + 3x_4$$

s.a.

$$x_1 + x_2 \geq 1$$

$$x_1 + x_3 \geq 1$$

$$+ x_2 + x_3 \geq 1$$

$$+ x_4 \geq 1$$

$$x_1, x_2, x_3, x_4 \in \{0, 1\}$$

Na resolução deste problema de programação linear usou-se uma linguagem de programação [13] MATLAB 5.3, que foi implementado em duas etapas:

a) Fez-se um programa para resolver o problema pelo método Simplex e obteve-se uma solução otimista, apresentado na Tabela II.

TABELA II

SOLUÇÃO OTIMISTA PELO MÉTODO SIMPLEX

	7.5000	-1.5000	-1.5000	-1.5000	-
3.0000	0.5000	-0.5000	-0.5000	0.5000	0
	0.5000	-0.5000	0.5000	-0.5000	0
	0.5000	0.5000	-0.5000	-0.5000	0
1.0000	1.0000	0	0	0	-

O valor da função objetivo é igual a 7.5000, os valores da coluna abaixo do valor da função objetivo são os valores das variáveis, ou seja, $\{(0.5000, 0.5000, 0.5000, 1.0000)\}$;

b) A solução da primeira etapa é otimista, então, implementou-se o programa (Branch2) empregando o dual simplex canalizado, assim, dando sequência na resolução do método Branch and Bound.

No desenvolvimento do problema tem-se dois subproblemas P_1 e P_2 .

Resolução do subproblema P_1

Para resolver o subproblema P_1 acrescentou-se uma restrição representada na Tabela III.

TABELA III

SOLUÇÃO OTIMISTA COM ACRÉSCIMO DA RESTRIÇÃO

	7.5000	-1.5000	-1.5000	-1.5000	-
3.0000	0.5000	-0.5000	-0.5000	0.5000	0
	0.5000	-0.5000	0.5000	-0.5000	0
	0.5000	0.5000	-0.5000	-0.5000	0

1.0000	0	0	0	-1.0000
-0.5000	-0.5000	0.5000	0.5000	0

Implementou-se o programa Dual Simplex Canalizado e encontrou-se como solução ótima global da função booleana apresentada na Tabela IV.

O valor da função objetivo é igual a 9 e os valores da coluna abaixo do valor da função objetivo são os valores das variáveis, ou seja, o conjunto verdade $\{(1, 0, 1, 1, 1)\}$.

TABELA IV

SOLUÇÃO ÓTIMA DO SUBPROBLEMA P_1

9	0	-3	-3	-3
1	0	-1	0	0
0	-1	1	0	0
1	1	-1	-1	0
1	0	0	0	-1
1	1	-2	-1	0

Resolução do subproblema P_2

Para resolver o subproblema P_2 adicionou-se uma nova restrição e implementou-se o programa Dual Simplex Canalizado e obteve-se como solução ótima do problema de cobertura mínima da função booleana $f(A,B,C,D) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$ representado na Tabela V.

TABELA V

SOLUÇÃO ÓTIMA DO SUBPROBLEMA P_2

9	-3	-3	-3	-3
0	0	0	1	0
1	-1	0	-1	0
1	0	-1	-1	0
1	0	0	0	-1
1	-1	-1	2	0

A solução ótima do subproblema P_2 é o valor da F.O = 9 (função objetivo) e os valores das variáveis é o conjunto verdade $\{(0, 1, 1, 1, 1)\}$.

Observa-se que em ambos os subproblemas P_1 e P_2 o custo da função booleana $f(A,B,C,D) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$ é F.O. = 9.

Outros algoritmos foram implementados neste Exemplo na linguagem de programação MATLAB 5.3 e o valor obtido em ambos o caso foi o mesmo para função booleana $f(A,B,C,D) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$.

Implementação do Algoritmo Dual do Simplex

Partindo-se do quadro otimista implementa-se o algoritmo Dual Simplex na linguagem de programação (MATLAB 5.3), e obtém-se como resultado:

FO = 9, $X_2 = 1$, $X_3 = 1$, $X_7 = 1$ e $X_4 = 1$

Assim, a solução ótima global do problema de cobertura da função booleana $f(A,B,C,D) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$ é dada por: custo

mínimo igual a 9, e o conjunto verdade {(0, 1, 1, 1, 0, 0, 1)}, embora em ambos casos os conjuntos verdades sejam diferentes, porém, o custo é o mesmo.

Implementação do Algoritmo Plano de Corte ou Algoritmo de Gomory.

Partindo-se do quadro otimista da Tabela II encontrado pelo algoritmo Simplex, e acrescenta-se uma restrição de corte tem-se a Tabela VI, para depois implementar o algoritmo de Gomory (Gomo2) na linguagem de programação MATLAB 5.3.

TABELA VI
ACRÉSCIMO DE UMA NOVA RESTRIÇÃO NO QUADRO OTIMISTA

	7.5000	-1.5000	-1.5000	-1.5000	-
3.0000	0.5000	-0.5000	-0.5000	0.5000	0
	0.5000	-0.5000	0.5000	-0.5000	0
	0.5000	0.5000	-0.5000	-0.5000	0
	1.0000	0	0	0	-
1.0000	-0.5000	0.5000	-0.5000	-0.5000	0

Implementando o algoritmo de Gomory (Gomo2) tem-se a apresentação da Tabela VII a qual apresenta a solução ótima do problema.

TABELA VII
SOLUÇÃO ÓTIMA PELO MÉTODO DE GOMORY

9	-3	0	0	-3
1	-1	0	1	0
1	-1	1	0	0
0	1	-1	-1	0
1	0	0	0	-1
1	-2	1	1	0

A função objetivo FO = 9 e os números que aparecem na coluna abaixo do número 9 são os valores das variáveis, ou seja, o conjunto verdade {(1, 1, 0, 1, 1)}.

Comparando-se as três implementações obteve-se o conjunto de soluções diferentes, porém com o mesmo custo, ou seja, em todos os casos a solução mínima global da função booleana $f(A,B,C,D) = \Sigma m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$ tem o custo igual a 9. Assim, obteve-se parâmetros para os três algoritmos implementados para relacionar custos, números de iterações, tempo computacional e quantidades de memórias, apresentados na Tabela VIII. Os resultados obtidos são:

TABELA VIII
APRESENTAÇÃO DA ANÁLISE DOS ALGORITMOS BRANCH2, GO DUAL

Programa	Custo	Iteração	Tempo(seg)	Mem.(bytes)
Branch2	9	6	2,56	2.772
Gomo2	9	5	2,09	2.448
Dual	9	3	0,77	1.780

As Figuras 3, 4 e 5, respectivamente, apresentam a relação gráfica das iterações, tempo gasto em segundos e memória em bytes dos três algoritmos implementados.

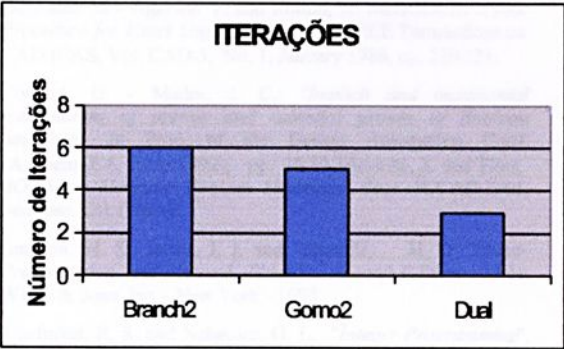


FIGURA. 3
APRESENTA AS ITERAÇÕES GASTAS PELOS PROGRAMAS BRANC2, GOMO2 E DUAL.

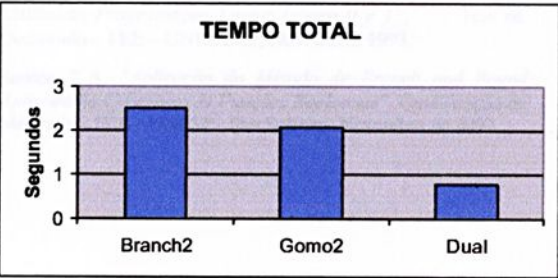


FIGURA. 4
APRESENTA O TEMPO GASTO PELOS PROGRAMAS BRANC2, GOMO2 E DUAL.

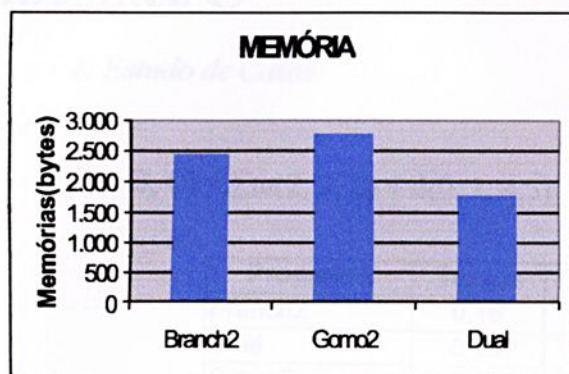


FIGURA. 5
APRESENTA A QUANTIA DE MEMÓRIAS GASTA PELOS
PROGRAMAS BRANC2, GOMO2 E DUAL.

CONCLUSÃO

Os circuitos elétricos são representados matematicamente pelas funções booleanas e neste trabalho revolveu-se a função booleana $f(A,B,C,D) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$ como exemplo. Através do algoritmo Expander, que se utiliza o método de expansão de Shannon, gerou-se os implicantes primos as funções e o problema de cobertura foi modelado como um problema de programação matemática.

Visto que todo problema de cobertura mínima de uma função booleana é solução de um problema de programação linear inteira 0 e 1, implementou-se o algoritmo de Branch and Bound e encontrou-se a solução ótima global do problema de cobertura mínima.

Comparando-se com a implementação de outros algoritmos como Plano de Corte de Gomory e Dual Simplex observou-se que os três algoritmos implementados encontram o mesmo custo. A análise da Tabela II mostra os parâmetros entre os programas Branch2, Gomo2 e Dual quanto ao número de iterações, quantidade de memórias e o tempo em segundos gastos na execução de cada programa. Chega-se a conclusão que entre os três programas implementados o Dual leva uma ligeira vantagem, pois tem um número menor de iterações, o tempo gasto para executar o programa é menor comparando com os outros dois e a quantidade de memórias é bem menor.

Muito outros casos de cobertura estão sendo avaliados com o objetivo de validar os algoritmos implementados.

AGRADECIMENTOS

Os autores agradecem a Direção da Fundação do Ensino Superior de Rio Verde "FESURV" e a

CAPES, pelo apoio irrestrito dado as pesquisas realizadas no desenvolvimento deste trabalho.

REFERÊNCIAS

- Hill, F. e G. Peterson, Switching Theory and Logical Design, Wiley, New York, 1981.
- McCluskey, E.J. "Minimization of Boolean functions, The Bell System Technical Journal, 35, No 5, November 1956. pp. 1417-1444.
- Kohavi, Z. Switching and Finite Automata Theory, McGraw-Hill, 1970.
- Quine, W. The Problem of Simplifying Truth Functions, American Mathematical Monthly, Vol. 59, 1952, pp. 521-531.
- Rudell and A. Sangiovanni-Vincentelli, Multiple - valued Minimization for PLA Optimization, IEEE Transactions on CAD/ICAS, Vol. CAD-6, No. 5, September 1987, pp 727-750.
- Dagenais, M - Agarwal V. and Rumin, N. McBOOLE: A New Procedure for Exact Logic Minimization, IEEE Transactions on CAD/ICAS, Vol. CAD-5, No. 1, January 1986, pp. 229-223.
- Coudert, O. - Madre, J. C.: "Implicit and incremental computation of primes and essential primes of Boolean functions, In Proc. of the Design Automation Conf. (Anaheim, CA, June 1992), pp. 36-39 Hlavicka, J. and Fišer, BOOM- a Heuristic Boolean Minimizer. Proc. ICCAD-2001, San Jose. Cal. (USA).
- Bazaraa, M. S., Jarvis, J. J. and Sherali, H. D. "Linear Programming and Network Flows" - Second Edition - John Wiley & Sons, Inc. - New York - 1993.
- Garfinkel, R. S. and Nehauser, G. L. "Integer Programming", Library of Congress Cataloging in Publication Data - United States - 1976.
- Emer, F.R.P, "Formulação Matemática do Problema de Otimização de Funções Booleanas Através do Método de Expansão de Shannon", Dissertação de Mestrado - FEE - UNESP - Ilha Solteira- Outubro de 2002.
- Silva, A.C.R., "Contribuição à Síntese de Circuitos Digitais Utilizando Programação Linear Inteira 0 e 1", Tese de Doutorado - FEE - UNICAMP, Novembro, 1993.
- Santos, T.A. "Aplicação do Método de Branch and Bound Aplicado na Cobertura de Funções Booleanas", Qualificação de Mestrado - FEE - UNESP - Ilha Solteira- Novembro de 2002.

Apêndice C

1. Estudo de Casos

5.2 - 2

$$f(A, B, C) = \sum m(2, 3, 6) + d(0, 1, 4, 5)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	2	1336	4
Dual	0,13	2	1996	4
Gomo2	0,17	2	1336	4

Tabela 5.2: Comparação dos programas: Gomo2, Branch2 e Dual.

5.3 - 3

$$f(A, B, C) = \sum m(2, 4, 5, 7) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	1992	6
Dual	0,22	3	1548	6
Gomo2	0,22	3	1992	6

Tabela 5.3: Comparação dos programas: Gomo2, Branch2 e Dual.

5.4 - 3

$$f(A, B, C) = \sum m(1, 6, 7) + d(0, 2, 4, 5)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	1656	4
Dual	0,12	2	1188	4
Gomo2	0,22	3	1784	4

Tabela 5.4: Comparação dos programas: Gomo2, Branch2 e Dual.

5.5 - 3

$$f(A, B, C) = \sum m(1, 5, 7) + d(0, 2, 3, 4)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	2	1656	4
Dual	0,12	2	1188	4
Gomo2	0,17	2	1656	4

Tabela 5.5: Comparação dos programas: Gomo2, Branch2 e Dual.

5.6 - 3

$$f(A, B, C) = \sum m(1, 2, 6) + d(0, 3, 4, 5)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	2	1656	4
Dual	0,12	2	1188	4
Gomo2	0,17	2	1656	4

Tabela 5.6: Comparação dos programas: Gomo2, Branch2 e Dual.

5.7 - 3

$$f(A, B, C) = \sum m(1, 2, 7) + d(0, 3, 4, 5)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	2	1656	4
Dual	0,13	2	1188	4
Gomo2	0,16	2	1656	4

Tabela 5.7: Comparação dos programas: Gomo2, Branch2 e Dual.

5.8 - 3

$$f(A, B, C) = \sum m(3, 4, 6) + d(0, 1, 2, 7)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,17	2	1656	4
Dual	0,11	2	1188	4
Gomo2	0,11	2	1656	4

Tabela 5.8: Comparação dos programas: Gomo2, Branch2 e Dual.

5.9 - 3

$$f(A, B, C) = \sum m(1, 4, 5) + d(0, 2, 3, 6)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	3	1640	2
Dual	0,11	1	1204	2
Gomo2	0,22	3	1768	2

Tabela 5.9: Comparação dos programas: Gomo2, Branch2 e Dual.

5.10 - 3

$$f(A, B, C) = \sum m(1, 4, 7) + d(0, 2, 3, 6)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	3	1584	6
Dual	0,16	3	1188	6
Gomo2	0,22	3	1584	6

Tabela 5.10: Comparação dos programas: Gomo2, Branch2 e Dual.

5.11 - 3

$$f(A, B, C) = \sum m(4, 8, 12, 13, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,38	4	2000	8
Dual	0,11	2	1548	8
Gomo2	0,38	4	2160	8

Tabela 5.11: Comparação dos programas: Gomo2, Branch2 e Dual.

5.12 - 3

$$f(A, B, C) = \sum m(4, 8, 9, 14, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	3	1584	11
Dual	0,17	3	1188	11
Gomo2	0,17	3	1584	11

Tabela 5.12: Comparação dos programas: Gomo2, Branch2 e Dual.

5.13 - 3

$$f(A, B, C) = \sum m(3, 5, 8, 10, 15) + d(0, 1, 7)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,17	3	1584	11
Dual	0,16	3	1188	11
Gomo2	0,17	3	1584	11

Tabela 5.13: Comparação dos programas: Gomo2, Branch2 e Dual.

5.14 - 3

$$f(A, B, C) = \sum m(5, 7, 10, 11, 15) + d(1, 2, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,33	5	2504	9
Dual	0,11	3	1954	9
Gomo2	0,38	5	2696	9

Tabela 5.14: Comparação dos programas: Gomo2, Branch2 e Dual.

5.15 - 3

$$f(A, B, C) = \sum m(0, 2, 4, 6) + d(1, 3, 5)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,28	4	1968	2
Dual	0,11	1	1596	2
Gomo2	0,28	4	2128	2

Tabela 5.15: Comparação dos programas: Gomo2, Branch2 e Dual.

5.16 - 3

$$f(A, B, C) = \sum m(0, 1, 2) + d(3, 5, 6, 7)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	1640	2
Dual	0,11	1	1204	2
Gomo2	0,22	3	1768	2

Tabela 5.16: Comparação dos programas: Gomo2, Branch2 e Dual.

5.17 - 3

$$f(A, B, C) = \sum m(1, 2) + d(0, 3, 5, 6, 7)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,06	1	1136	2
Dual	0,06	1	876	2
Gomo2	0,05	1	1136	2

Tabela 5.17: Comparação dos programas: Gomo2, Branch2 e Dual.

5.18 - 3

$$f(A, B, C) = \sum m(1, 2, 4) + d(0, 3, 5, 6)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,11	5	2444	4
Dual	0,06	2	1172	4
Gomo2	0,11	4	1864	4

Tabela 5.18: Comparação dos programas: Gomo2, Branch2 e Dual.

5.19 - 3

$$f(A, B, C, D, E) = \sum m(4, 8, 14) + d(0, 6, 10, 12)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,49	5	2444	6
Dual	0,11	2	1172	6
Gomo2	0,49	4	1864	6

Tabela 5.19: Comparação dos programas: Gomo2, Branch2 e Dual.

5.20 - 4

$$f(A, B, C) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,27	6	3508	9
Dual	0,11	3	1780	9
Gomo2	0,22	5	2704	9

Tabela 5.20: Comparação dos programas: Gomo2, Branch2 e Dual.

5.21 - 4

$$f(A, B, C) = \sum m(4, 6, 8, 13, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	2461	12
Dual	0,17	3	1780	12
Gomo2	0,22	3	2464	12

Tabela 5.21: Comparação dos programas: Gomo2, Branch2 e Dual.

Tabela 5.22: Comparação dos programas: Gomo2, Branch2 e Dual.

5.22 - 4

$f(A, B, C) = \sum m(4, 5, 8, 13, 15) + d(0, 1, 3)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	2464	11
Dual	0,17	3	1780	11
Gomo2	0,22	3	2464	11

Tabela 5.22: Comparação dos programas: Gomo2, Branch2 e Dual.

5.23 - 4

$f(A, B, C) = \sum m(4, 8, 9, 13, 15) + d(0, 1, 3)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,50	5	2800	11
Dual	0,11	3	2228	11
Gomo2	0,49	5	3040	11

Tabela 5.23: Comparação dos programas: Gomo2, Branch2 e Dual.

5.24 - 4

$f(A, B, C) = \sum m(4, 8, 11, 13, 15) + d(0, 1, 3)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	3	2464	12
Dual	0,11	3	1780	12
Gomo2	0,16	3	2464	12

Tabela 5.24: Comparação dos programas: Gomo2, Branch2 e Dual.

5.25 - 4

$f(A, B, C) = \sum m(4, 8, 13, 14, 15) + d(0, 1, 3)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	2464	12
Dual	0,11	3	1780	12
Gomo2	0,22	3	2464	12

Tabela 5.25: Comparação dos programas: Gomo2, Branch2 e Dual.

5.26 - 4

$$f(A, B, C) = \sum m(4, 8, 9, 13, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,49	5	2784	11
Dual	0,17	3	2228	11
Gomo2	0,49	5	3024	11

Tabela 5.26: Comparação dos programas: Gomo2, Branch2 e Dual.

5.27 - 4

$$f(A, B, C) = \sum m(4, 8, 9, 10, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,16	4	2376	17
Dual	0,11	4	1784	17
Gomo2	0,17	4	2376	17

Tabela 5.27: Comparação dos programas: Gomo2, Branch2 e Dual.

5.28 - 4

$$f(A, B, C) = \sum m(2, 3, 5, 8, 10, 15) + d(0, 1, 7)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,50	5	2789	10
Dual	0,17	3	2228	10
Gomo2	0,49	5	3024	10

Tabela 5.28: Comparação dos programas: Gomo2, Branch2 e Dual.

5.29 - 4

$$f(A, B, C) = \sum m(0, 6, 8, 10, 15) + d(1, 2, 7, 9, 11, 14)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,33	6	3508	9
Dual	0,11	3	1780	9
Gomo2	0,22	5	2704	9

Tabela 5.29: Comparação dos programas: Gomo2, Branch2 e Dual.

5.30 - 5

$$f(A, B, C) = \sum m(4, 7, 8, 13, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,27	4	3448	16
Dual	0,22	4	2500	16
Gomo2	0,33	4	3448	16

Tabela 5.30: Comparação dos programas: Gomo2, Branch2 e Dual.

5.31 - 5

$$f(A, B, C) = \sum m(4, 8, 10, 13, 15) + d(0, 1, 3)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,38	5	3448	16
Dual	0,11	4	2500	16
Gomo2	0,44	5	3736	16

Tabela 5.31: Comparação dos programas: Gomo2, Branch2 e Dual.

5.32 - 5

$$f(A, B, C) = \sum m(2, 6, 9, 14) + d(0, 3, 5)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,22	3	2456	13
Dual	0,16	3	2012	13
Gomo2	0,21	3	2456	13

Tabela 5.32: Comparação dos programas: Gomo2, Branch2 e Dual.

5.33 - 5

$$f(A, B, C, D, E) = \sum m(18, 35, 49, 55, 59) + d(19, 34, 48, 50, 51, 54, 58)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,28	5	3344	25
Dual	0,11	5	2500	25
Gomo2	0,27	5	3344	25

Tabela 5.33: Comparação dos programas: Gomo2, Branch2 e Dual.

5.34 - 6

$$f(A, B, C, D, E, F) = \sum m(5, 33, 36, 38, 53) + d(21, 37, 45)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,49	5	3328	17
Dual	0,11	3	2774	17
Gomo2	0,49	5	3664	17

Tabela 5.34: Comparação dos programas: Gomo2, Branch2 e Dual.

5.35 - 6

$$f(A, B, C, D, E) = \sum m(10, 11, 18, 27, 28) + d(2, 14, 22, 24, 25, 26, 30)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,72	5	5128	12
Dual	0,38	6	3840	12
Gomo2	0,72	5	5576	12

Tabela 5.35: Comparação dos programas: Gomo2, Branch2 e Dual.

5.36 - 6

$$f(A, B, C, D, E) = \sum m(10, 22, 24, 27) + d(2, 11, 14, 18, 25, 26, 28, 30)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,66	6	3344	12
Dual	0,33	3	3182	12
Gomo2	0,66	6	3632	12

Tabela 5.36: Comparação dos programas: Gomo2, Branch2 e Dual.

5.37 - 6

$$f(A, B, C, D, E) = \sum m(2, 11, 14, 22, 25, 30) + d(10, 18, 24, 26, 27, 28)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,76	7	4608	20
Dual	0,39	5	3812	20
Gomo2	0,77	7	4944	20

Tabela 5.37 Comparação dos programas: Gomo2, Branch2 e Dual.



5.38 - 6

$$f(A, B, C, D, E, F) = \sum m(2, 10, 18, 24, 25) + d(11, 14, 22, 26, 27, 28, 30)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,60	8	3672	8
Dual	0,27	2	3182	8
Gomo2	0,60	8	4008	8

Tabela 5.38: Comparação dos programas: Gomo2, Branch2 e Dual.

5.39 - 7

$$f(A, B, C, D, E, F) = \sum m(12, 13, 33, 37, 41, 44, 61) + d(5, 29, 43, 45, 46, 47, 53)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,88	7	6440	15
Dual	0,38	3	4918	15
Gomo2	0,83	7	6952	15

Tabela 5.39: Comparação dos programas: Gomo2, Branch2 e Dual.

5.40 - 7

$$f(A, B, C, D, E, F) = \sum m(5, 12, 29, 33, 41, 44, 53) + d(13, 37, 43, 45, 46, 47, 61)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,99	7	6456	25
Dual	0,44	5	4916	25
Gomo2	0,99	7	6968	25

Tabela 5.40: Comparação dos programas: Gomo2, Branch2 e Dual.

5.41 - 7

$$f(A, B, C, D, E, F) = \sum m(13, 33, 37, 44, 47, 53) + d(5, 12, 29, 41, 43, 45, 46, 61)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,82	6	4808	20
Dual	0,39	4	4174	20
Gomo2	0,77	6	5256	20

Tabela 5.41: Comparação dos programas: Gomo2, Branch2 e Dual.

5.42 - 7

$$f(A, B, C, D, E, F) = \sum m(12, 29, 33, 37, 53, 61) + d(5, 13, 41, 43, 45, 46)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,88	7	6448	20
Dual	0,88	4	4918	20
Gomo2	0,44	7	6960	20

Tabela 5.42: Comparação dos programas: Gomo2, Branch2 e Dual.

5.43 - 7

$$f(A, B, C, D, E, F) = \sum m(5, 12, 29, 33, 37, 44, 47, 61) + d(13, 41, 43, 45, 46, 53)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,88	7	6448	20
Dual	0,44	4	4918	20
Gomo2	0,88	7	6960	20

Tabela 5.43: Comparação dos programas: Gomo2, Branch2 e Dual.

5.44 - 7

$$f(A, B, C, D, E, F) = \sum m(12, 29, 37, 43, 44, 61) + d(5, 13, 33, 41, 45, 46, 47, 53)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,77	6	4808	20
Dual	0,44	4	4188	20
Gomo2	0,77	6	5246	20

Tabela 5.44: Comparação dos programas: Gomo2, Branch2 e Dual.

5.45 - 7

$$f(A, B, C, D, E, F) = \sum m(5, 13, 33, 46, 47, 53, 61) + d(12, 29, 37, 41, 43, 44, 45)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	0,88	6	4816	25
Dual	0,44	4	4188	25
Gomo2	0,88	6	5264	25

Tabela 5.45: Comparação dos programas: Gomo2, Branch2 e Dual.

5.46 - 8

$f(A, B, C, D, E, F) = \sum m(2, 18, 35, 43, 50, 55, 59) + d(19, 27, 39, 49, 51, 53, 57, 63)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,15	8	4528	15
Dual	0,44	3	4564	15
Gomo2	1,10	8	4464	15

Tabela 5.46: Comparação dos programas: Gomo2, Branch2 e Dual.

5.47 - 8

$f(A, B, C, D, E, F) = \sum m(18, 27, 39, 43, 49, 53, 59, 63) + d(2, 19, 35, 50, 51, 55, 57)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,26	8	5496	25
Dual	0,55	5	5340	25
Gomo2	1,26	8	6008	25

Tabela 5.47: Comparação dos programas: Gomo2, Branch2 e Dual.

5.48 - 8

$f(A, B, C, D, E, F) = \sum m(2, 18, 35, 43, 49, 53, 59) + d(19, 27, 39, 50, 51, 55, 57, 63)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,20	8	5488	20
Dual	0,50	4	5340	20
Gomo2	1,26	8	6000	20

Tabela 5.48: Comparação dos programas: Gomo2, Branch2 e Dual.

5.49 - 8

$f(A, B, C, D, E, F) = \sum m(2, 18, 27, 35, 43, 49, 53, 63) + d(19, 39, 50, 51, 55, 57, 59)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,26	8	5488	20
Dual	0,50	4	5340	20
Gomo2	1,21	8	6000	20

Tabela 5.49: Comparação dos programas: Gomo2, Branch2 e Dual.

5.50 - 8

$$f(A, B, C, D, E, F) = \sum m(2, 18, 27, 39, 53, 59, 63) + d(19, 35, 43, 49, 50, 51, 55, 57)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,32	8	5496	25
Dual	0,55	5	5340	25
Gomo2	1,21	8	6008	25

Tabela 5.33: Comparação dos programas: Gomo2, Branch2 e Dual.

5.51 - 8

$$f(A, B, C, D, E, F) = \sum m(18, 39, 43, 49, 53, 55, 57, 59) + d(2, 19, 27, 35, 50, 51, 63)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,32	7	8248	20
Dual	0,55	4	6164	20
Gomo2	1,15	7	8960	20

Tabela 5.51: Comparação dos programas: Gomo2, Branch2 e Dual.

5.52 - 8

$$f(A, B, C, D, E, F) = \sum m(2, 19, 27, 35, 43, 49, 53, 59) + d(18, 39, 50, 51, 55, 57, 63)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,42	10	7320	20
Dual	0,49	4	6164	20
Gomo2	1,48	10	7834	20

Tabela 5.52: Comparação dos programas: Gomo2, Branch2 e Dual.

5.53 - 8

$$f(A, B, C, D, E, F) = \sum m(2, 18, 19, 39, 53, 59, 63) + d(39, 50, 51, 55, 57, 59)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,26	8	5496	25
Dual	0,55	5	5340	25
Gomo2	1,32	8	6008	25

Tabela 5.53: Comparação dos programas: Gomo2, Branch2 e Dual.

5.54 - 10

$f(A, B, C, D, E) = \sum m(4, 8, 12, 16, 21, 22, 25, 27, 29, 31) + d(0, 1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15, 17, 18, 19, 20, 23, 24, 26, 28, 30)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,09	11	8096	15
Dual	0,66	5	8076	15
Gomo2	2,14	11	8736	15

Tabela 5.54: Comparação dos programas: Gomo2, Branch2 e Dual.

5.55 - 10

$f(A, B, C, D, E) = \sum m(3, 4, 10, 11, 16, 21, 18, 30, 31) + d(0, 1, 2, 5, 6, 7, 8, 9, 12, 13, 14, 15, 17, 118, 19, 20, 22, 23, 24, 25, 27, 29)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,54	9	7184	12
Dual	0,61	4	7110	12
Gomo2	1,54	9	7904	12

Tabela 5.55: Comparação dos programas: Gomo2, Branch2 e Dual.

5.56- 10

$f(A, B, C, D, E) = \sum m(1, 2, 4, 7, 11, 15, 20, 24, 25, 29, 30) + d(0, 3, 5, 6, 8, 9, 10, 12, 13, 14, 16, 17, 18, 19, 21, 22, 23, 25, 27, 28, 31)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,81	9	9128	12
Dual	0,66	4	8062	12
Gomo2	1,92	9	10008	12

Tabela 5.56: Comparação dos programas: Gomo2, Branch2 e Dual.

5.57 - 10

$f(A, B, C, D, E) = \sum m(0, 2, 4, 7, 10, 13, 16, 19, 23, 25, 26, 31) + d(1, 5, 6, 8, 9, 11, 12, 14, 15, 17, 18, 19, 20, 21, 22, 24, 27, 28, 29, 30)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,87	9	11041	12
Dual	0,71	4	8078	12
Gomo2	1,87	9	10008	12

Tabela 5.57: Comparação dos programas: Gomo2, Branch2 e Dual.

5.58 - 10

$$f(A, B, C, D, E) = \sum m(1, 5, 7, 17, 20, 23, 29) + d(4, 13, 21)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,21	9	7400	16
Dual	0,55	4	6164	16
Gomo2	1,32	9	7976	16

Tabela 5.58: Comparação dos programas: Gomo2, Branch2 e Dual.

5.59 - 10

$$f(A, B, C, D, E) = \sum m(1, 5, 7, 13, 17, 13, 17, 20, 23, 29) + d(21, 4)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,20	9	7400	12
Dual	0,50	4	6164	12
Gomo2	1,21	9	7976	12

Tabela 5.59: Comparação dos programas: Gomo2, Branch2 e Dual.

5.60 - 10

$$f(A, B, C, D, E) = \sum m(0, 1, 3, 7, 11, 15, 19, 20, 27, 29, 30, 31) + d(2, 4, 5, 6, 8, 9, 10, 12, 13, 14, 16, 17, 18, 25, 21, 22, 23, 24)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,03	9	12112	12
Dual	0,71	4	9094	12
Gomo2	2,04	9	13168	12

Tabela 5.60: Comparação dos programas: Gomo2, Branch2 e Dual.

5.61 - 12

$$f(A, B, C, D, E, F, G, H, I) = \sum m(5, 9, 12, 15, 37, 44, 61, 141, 109, 173) + d(13, 29, 41, 45, 77, 165, 169, 175, 189, 261, 265, 268, 269, 271, 285, 293, 297, 300, 301, 303, 317, 333, 365, 397, 421, 425, 429, 431, 445, 493)$$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,42	10	11472	56
Dual	1,10	8	10214	56
Gomo2	2,36	10	12616	56

Tabela 5.61: Comparação dos programas: Gomo2, Branch2 e Dual.

5.62 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(9, 29, 44, 77, 141, 173, 237, 269, 301, 333, 421, 445) + d(5, 12, 13, 15, 37, 41, 45, 47, 61, 109, 165, 169, 175, 189, 26, 265, 268, 271, 285, 293, 297, 300, 303, 317, 365, 397, 425, 429, 431, 493)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,07	11	11824	49
Dual	1,21	7	11374	49
Gomo2	2,91	11	13072	49

Tabela 5.62: Comparação dos programas: Gomo2, Branch2 e Dual.

5.63 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(13, 37, 47, 109, 173, 268, 317, 333, 421, 425, 445, 493) + d(5, 9, 12, 15, 29, 41, 44, 45, 61, 77, 141, 165, 169, 175, 189, 237, 261, 265, 269, 271, 285, 293, 297, 300, 301, 303, 365, 397, 429, 431)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,57	12	15448	42
Dual	0,93	6	12598	42
Gomo2	3,57	12	16800	42

Tabela 5.63: Comparação dos programas: Gomo2, Branch2 e Dual.

5.64 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(9, 44, 165, 189, 269, 300, 317, 397, 425, 431, 445, 493) + d(5, 12, 13, 15, 29, 37, 41, 45, 47, 61, 77, 109, 141, 169, 173, 175, 237, 261, 265, 268, 271, 285, 293, 297, 301, 303, 333, 365, 421, 429)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,70	11	10280	56
Dual	0,99	8	10230	56
Gomo2	2,64	11	11336	56

Tabela 5.64: Comparação dos programas: Gomo2, Branch2 e Dual.

5.65 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(9, 12, 29, 44, 141, 271, 293, 365, 425, 421, 445, 493) + d(5, 13, 15, 37, 41, 45, 47, 61, 77, 109, 165, 169, 173, 175, 189, 237, 261, 265, 268, 269, 285, 297, 300, 301, 303, 317, 333, 397, 425, 429)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,14	12	11554	63
Dual	1,04	9	11390	63
Gomo2	3,13	12	12696	63

Tabela 5.65: Comparação dos programas: Gomo2, Branch2 e Dual.

5.66 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(12, 15, 29, 37, 61, 173, 261, 293, 303, 365, 429, 497) + d(5, 9, 13, 41, 44, 45, 47, 77, 109, 141, 165, 169, 175, 189, 237, 265, 268, 269, 271, 285, 293, 300, 301, 317, 333, 397, 421, 425, 431, 445)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,13	12	11528	42
Dual	0,93	6	11290	42
Gomo2	3,19	12	12680	42

Tabela 5.66: Comparação dos programas: Gomo2, Branch2 e Dual.

5.67 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(5, 12, 13, 41, 45, 109, 173, 301, 333, 421, 429) + d(9, 15, 29, 37, 44, 47, 61, 77, 141, 165, 169, 175, 189, 237, 261, 265, 268, 269, 271, 285, 293, 297, 300, 303, 365, 397, 425, 431, 445, 493)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,69	12	11475	56
Dual	0,88	7	10230	56
Gomo2	2,63	12	12301	56

Tabela 5.67: Comparação dos programas: Gomo2, Branch2 e Dual.

5.68 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(5, 13, 44, 61, 109, 175, 271, 297, 303, 425) + d(9, 12, 15, 29, 37, 41, 45, 47, 77, 141, 165, 169, 173, 189, 237, 261, 265, 268, 269, 285, 293, 300, 301, 317, 333, 365, 397, 421, 429, 431, 493)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,59	11	10288	49
Dual	0,93	7	10230	49
Gomo2	2,58	11	11334	49

Tabela 5.68: Comparação dos programas: Gomo2, Branch2 e Dual.

5.69 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(5, 15, 29, 41, 61, 141, 173, 269, 293, 317, 431) + d(9, 12, 13, 37, 44, 45, 47, 77, 109, 165, 169, 175, 189, 237, 261, 265, 268, 271, 285, 297, 300, 301, 303, 333, 365, 397, 421, 425, 429, 445, 493)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,76	10	7864	35
Dual	0,71	5	8054	35
Gomo2	1,70	10	8656	35

Tabela 5.69: Comparação dos programas: Gomo2, Branch2 e Dual.

5.70 - 12

$f(A, B, C, D, E, F, G, H, I) = \sum m(19, 12, 15, 29, 47, 61, 77, 141, 173, 445) + d(13, 37, 41, 44, 45, 109, 165, 169, 175, 189, 237, 261, 265, 268, 269, 271, 285, 293, 292, 300, 301, 303, 317, 333, 365, 397, 421, 425, 429, 431, 493)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	1,76	10	7864	35
Dual	0,71	5	8054	35
Gomo2	1,70	10	8656	35

Tabela 5.70: Comparação dos programas: Gomo2, Branch2 e Dual.

5.71 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(21, 52, 61, 87, 162, 176, 182, 187, 191, 246, 247) + d(23, 53, 54, 55, 63, 118, 119, 135, 143, 149, 151, 157, 159, 163, 166, 167, 171, 175, 177, 178, 179, 180, 181, 182, 185, 189, 199, 215, 231, 242, 243).$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,25	12	13288	48
Dual	1,21	8	12006	48
Gomo2	2,52	12	13288	48

Tabela 5.71: Comparação dos programas: Gomo2, Branch2 e Dual.

5.72 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(23, 53, 118, 143, 163, 181, 189, 231, 242, 247) + d(21, 52, 54, 55, 61, 63, 87, 119, 135, 149, 151, 157, 159, 162, 166, 167, 171, 175, 176, 177, 178, 179, 180, 182, 183, 185, 187, 191, 199, 215, 243, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,02	13	10544	48
Dual	1,10	7	10798	48
Gomo2	3,08	13	11512	48

Tabela 5.72: Comparação dos programas: Gomo2, Branch2 e Dual.

5.73 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(52, 55, 63, 119, 149, 159, 163, 178, 180, 189, 199, 231, 247) + d(21, 23, 53, 54, 61, 87, 118, 135, 143, 151, 157, 162, 166, 167, 171, 175, 176, 177, 179, 181, 182, 183, 185, 187, 191, 215, 243, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,28	17	12936	54
Dual	1,21	6	13262	54
Gomo2	4,29	17	13872	54

Tabela 5.73: Comparação dos programas: Gomo2, Branch2 e Dual.

5.74 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(23, 53, 119, 151, 167, 178, 199, 246, 247) + d(21, 52, 54, 55, 61, 63, 67, 118, 135, 143, 149, 157, 159, 162, 163, 166, 171, 175, 176, 177, 179, 180, 181, 183, 185, 189, 191, 215, 231, 243)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,24	15	11704	49
Dual	1,10	5	12006	49
Gomo2	3,30	15	12664	49

Tabela 5.74: Comparação dos programas: Gomo2, Branch2 e Dual.

5.75 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(52, 87, 118, 135, 151, 162, 167, 178, 180, 187, 215) + d(21, 23, 53, 54, 55, 61, 63, 119, 143, 149, 157, 159, 163, 166, 171, 175, 76, 177, 179, 183, 181, 182, 185, 189, 191, 199, 231, 242, 243, 246, 247)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,08	11	13432	49
Dual	0,88	7	12006	49
Gomo2	3,08	11	14776	49

Tabela 5.75: Comparação dos programas: Gomo2, Branch2 e Dual.

5.76 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(23, 54, 63, 135, 149, 162, 175, 178, 182, 189, 191, 231, 243) + d(21, 52, 53, 55, 61, 87, 118, 119, 143, 151, 157, 159, 163, 166, 167, 171, 176, 177, 179, 180, 181, 183, 185, 187, 199, 215, 242, 246, 247)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,29	15	17504	56
Dual	1,32	8	14566	56
Gomo2	4,29	15	18848	56

Tabela 5.76: Comparação dos programas: Gomo2, Branch2 e Dual.

5.77 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(21, 54, 63, 143, 159, 163, 175, 180, 181, 199, 231, 42, 247) + d(23, 52, 53, 55, 61, 87, 118, 119, 135, 149, 151, 157, 162, 1663, 167, 171, 176, 177, 178, 179, 182, 183, 185, 187, 189, 191, 215, 243, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,95	13	13392	49
Dual	1,10	7	13262	49
Gomo2	3,73	13	14744	49

Tabela 5.77: Comparação dos programas: Gomo2, Branch2 e Dual.

5.78 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(23, 55, 118, 151, 162, 163, 175, 182, 185, 242, 247) + d(21, 52, 53, 54, 61, 63, 87, 119, 135, 143, 149, 157, 159, 166, 167, 171, 176, 177, 178, 179, 180, 181, 183, 187, 189, 191, 199, 215, 231, 243, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,13	13	11920	30
Dual	0,99	5	12006	30
Gomo2	3,19	13	13072	30

Tabela 5.78: Comparação dos programas: Gomo2, Branch2 e Dual.

5.79 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(52, 54, 63, 119, 149, 166, 175, 178, 182, 191, 215, 243) + d(21, 23, 53, 55, 61, 87, 118, 135, 143, 151, 157, 159, 162, 163, 167, 171, 176, 177, 179, 180, 181, 182, 183, 185, 187, 189, 199, 231, 242, 243, 247)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,13	13	11936	48
Dual	1,21	8	12006	48
Gomo2	3,13	13	13008	48

Tabela 5.79: Comparação dos programas: Gomo2, Branch2 e Dual.

5.80 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(21, 52, 83, 118, 135, 162, 167, 180, 189, 199, 231, 247) + d(23, 53, 54, 55, 61, 87, 119, 143, 149, 151, 157, 159, 163, 166, 171, 175, 176, 177, 178, 179, 181, 182, 183, 185, 187, 191, 215, 242, 243, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,24	12	12024	36
Dual	0,98	6	12006	36
Gomo2	3,40	12	13272	36

Tabela 5.80: Comparação dos programas: Gomo2, Branch2 e Dual.

5.81 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(54, 149, 157, 163, 175, 180, 187, 231, 242, 247) + d(21, 23, 52, 53, 55, 61, 87, 118, 119, 135, 143, 151, 159, 162, 166, 167, 171, 176, 177, 178, 179, 181, 182, 183, 185, 189, 191, 199, 215, 243, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	2,64	11	10728	36
Dual	1,10	6	10798	36
Gomo2	2,64	11	11872	36

Tabela 5.81: Comparação dos programas: Gomo2, Branch2 e Dual.

5.82 - 13

$f(A, B, C, D, E, F, G, H) = \sum m(21, 53, 87, 135, 159, 166, 177, 182, 191, 243, 247) + d(23, 52, 54, 55, 61, 63, 118, 119, 143, 149, 151, 157, 161, 163, 167, 171, 175, 176, 178, 179, 180, 181, 183, 185, 187, 189, 199, 215, 231, 242, 246)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,19	12	12032	42
Dual	1,05	7	12006	42
Gomo2	3,19	12	13280	42

Tabela 5.82: Comparação dos programas: Gomo2, Branch2 e Dual.

5.83 - 14

$f(A, B, C, D, E, F, G, H) = \sum m(5, 12, 15, 29, 46, 68, 73, 85, 92, 108, 110, 124, 125) + d(8, 9, 13, 14, 21, 44, 45, 61, 65, 67, 69, 70, 71, 72, 74, 75, 76, 77, 78, 79, 81, 89, 91, 93, 95, 97, 101, 105, 109, 117)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,41	13	12424	35
Dual	0,99	6	12622	35
Gomo2	3,46	13	13672	35

Tabela 5.83: Comparação dos programas: Gomo2, Branch2 e Dual.

5.84 - 14

$f(A, B, C, D, E, F, G) = \sum m(5, 9, 12, 21, 45, 68, 72, 73, 75, 77, 79, 85, 97, 109) + d(8, 13, 14, 29, 44, 46, 61, 65, 67, 69, 70, 71, 74, 75, 76, 78, 89, 91, 92, 93, 95, 101, 105, 106, 110, 117, 124, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,90	15	13704	35
Dual	1,10	6	13926	35
Gomo2	3,96	15	14952	35

Tabela 5.84: Comparação dos programas: Gomo2, Branch2 e Dual.

5.85 - 14

$f(A, B, C, D, E, F, G, H) = \sum m(12, 15, 29, 44, 65, 68, 70, 73, 81, 93, 101, 110) + d(5, 8, 9, 13, 14, 21, 45, 46, 61, 67, 69, 71, 72, 74, 75, 76, 77, 78, 79, 85, 89, 91, 92, 95, 97, 105, 108, 109, 117, 124, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,41	14	12328	40
Dual	1,21	7	12622	40
Gomo2	3,40	14	13480	40

Tabela 5.85: Comparação dos programas: Gomo2, Branch2 e Dual.

5.86 - 14

$f(A, B, C, D, E, F, G) = \sum m(5, 9, 13, 29, 46, 65, 68, 72, 78, 85, 93, 105, 117, 125) + d(8, 12, 14, 15, 21, 44, 45, 61, 67, 69, 70, 71, 73, 74, 75, 76, 77, 79, 81, 89, 91, 92, 95, 97, 101, 103, 109, 110, 124)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,94	16	19920	35
Dual	1,26	7	16678	35
Gomo2	4,99	16	21480	35

Tabela 5.86: Comparação dos programas: Gomo2, Branch2 e Dual.

5.87 - 14

$f(A, B, C, D, E, F, G) = \sum m(8, 14, 15, 29, 46, 69, 72, 75, 78, 81, 92, 93, 108, 124) + d(5, 9, 12, 13, 21, 44, 45, 61, 65, 67, 68, 70, 71, 73, 74, 76, 77, 79, 85, 89, 91, 95, 101, 105, 109, 110, 117, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,72	15	15400	45
Dual	1,27	8	15278	45
Gomo2	4,45	15	16856	45

Tabela 5.87: Comparação dos programas: Gomo2, Branch2 e Dual.

5.88 - 14

$f(A, B, C, D, E, F, G, H) = \sum m(5, 9, 44, 65, 70, 74, 79, 81, 93, 97, 108, 125) + d(8, 12, 13, 14, 15, 21, 29, 45, 46, 61, 67, 68, 69, 71, 72, 73, 75, 76, 77, 78, 85, 89, 91, 92, 93, 95, 97, 101, 105, 108, 109, 110, 117, 124)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,79	13	13952	45
Dual	1,38	9	15408	45
Gomo2	3,79	13	13926	45

Tabela 5.88: Comparação dos programas: Gomo2, Branch2 e Dual.

5.89 - 14

$f(A, B, C, D, E, F, G, H) = \sum m(8, 13, 14, 29, 46, 65, 68, 71, 75, 77, 85, 92, 93, 101) + d(5, 9, 12, 15, 21, 44, 45, 61, 67, 69, 70, 72, 73, 74, 76, 78, 79, 81, 89, 91, 95, 97, 105, 108, 109, 110, 117, 124, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,62	16	15160	35
Dual	1,21	7	15278	35
Gomo2	4,61	16	16504	35

Tabela 5.89: Comparação dos programas: Gomo2, Branch2 e Dual.

5.90 - 14

$f(A, B, C, D, E, F, G, H) = \sum m(5, 12, 29, 46, 68, 73, 78, 89, 92, 105, 109, 125) + d(8, 9, 14, 15, 21, 44, 45, 61, 65, 67, 69, 70, 71, 72, 74, 75, 76, 77, 79, 81, 85, 91, 93, 95, 97, 101, 108, 110, 117, 124)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	3,90	14	13928	30
Dual	1,27	6	13926	30
Gomo2	3,90	14	15280	30

Tabela 5.90: Comparação dos programas: Gomo2, Branch2 e Dual.

5.91 - 14

$f(A, B, C, D, E, F, G, H) = \sum m(10, 11, 42, 48, 50, 59, 67, 74, 83, 98, 104, 111, 115, 125) + d(9, 27, 35, 39, 41, 43, 46, 47, 49, 51, 71, 73, 75, 73, 75, 79, 91, 96, 97, 99, 101, 103, 105, 106, 107, 109, 110, 119, 121, 122, 123, 127)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,67	15	15864	35
Dual	1,43	7	15990	35
Gomo2	4,67	15	17432	35

Tabela 5.91: Comparação dos programas: Gomo2, Branch2 e Dual.

5.92 - 15

$f(A, B, C, D, E, F, G) = \sum m(9, 11, 50, 71, 75, 79, 96, 103, 106, 108, 109, 111, 119, 123, 127) + d(10, 27, 35, 39, 41, 42, 43, 46, 47, 48, 49, 51, 59, 67, 73, 74, 75, 83, 91, 97, 98, 99, 101, 104, 105, 107, 110, 115, 121, 122, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	5,33	17	17312	45
Dual	1,43	9	17438	45
Gomo2	5,32	17	18872	45

Tabela 5.92: Comparação dos programas: Gomo2, Branch2 e Dual.

5.93 - 15

$f(A, B, C, D, E, F, G) = \sum m(11, 39, 42, 43, 48, 59, 74, 79, 96, 99, 104, 106, 121, 123, 127) + d(9, 10, 27, 35, 41, 46, 47, 49, 50, 51, 67, 71, 73, 75, 83, 91, 97, 98, 101, 103, 107, 108, 109, 110, 111, 115, 119, 122, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	5,77	20	22224	40
Dual	1,43	7	18934	40
Gomo2	5,72	20	23632	40

Tabela 5.93: Comparação dos programas: Gomo2, Branch2 e Dual.

5.94 - 15

$f(A, B, C, D, E, F, G) = \sum m(9, 27, 35, 42, 46, 49, 59, 67, 71, 83, 101, 105, 119, 122, 125) + d(10, 11, 39, 41, 43, 47, 48, 50, 51, 73, 74, 75, 79, 91, 96, 97, 98, 99, 103, 104, 106, 107, 108, 109, 110, 111, 115, 121, 123, 127)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	4,89	14	17816	45
Dual	1,43	9	17438	45
Gomo2	4,89	14	19736	45

Tabela 5.94: Comparação dos programas: Gomo2, Branch2 e Dual.

5.95 - 18

$f(A, B, C, D, E, F, G) = \sum m(12, 29, 43, 46, 69, 71, 76, 89, 93, 104, 107, 111, 117, 119, 121, 124, 125, 127) + d(5, 13, 37, 41, 44, 45, 47, 55, 57, 60, 61, 73, 77, 78, 79, 85, 96, 97, 99, 100, 101, 103, 105, 106, 108, 109, 110, 113, 120)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	8,35	20	22576	45
Dual	1,75	9	23046	45
Gomo2	7,80	20	24616	45

Tabela 5.95: Comparação dos programas: Gomo2, Branch2 e Dual.

5.96 - 18

$f(A, B, C, D, E, F, G) = \sum m(5, 13, 44, 55, 60, 73, 78, 85, 86, 96, 100, 105, 108, 110, 175, 124, 127) + d(12, 29, 37, 41, 43, 45, 46, 47, 57, 61, 69, 71, 76, 77, 79, 89, 93, 97, 99, 101, 104, 106, 107, 109, 111, 117, 119, 120, 121, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	7,19	18	2300	45
Dual	1,75	9	23046	45
Gomo2	7,25	18	25312	45

Tabela 5.96: Comparação dos programas: Gomo2, Branch2 e Dual.

5.97 - 18

$f(A, B, C, D, E, F, G) = \sum m(12, 37, 41, 47, 71, 78, 85, 93, 97, 103, 106, 107, 110, 117, 120, 121, 125, 127) + d(5, 13, 29, 43, 44, 45, 46, 55, 57, 60, 61, 69, 73, 76, 77, 79, 89, 96, 99, 100, 101, 104, 105, 108, 111, 113, 119, 124)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	7,75	19	22728	55
Dual	1,87	10	23046	55
Gomo2	7,80	19	24904	55

Tabela 5.97: Comparação dos programas: Gomo2, Branch2 e Dual.

5.98 - 18

$f(A, B, C, D, E, F, G) = \sum m(43, 45, 55, 57, 61, 69, 76, 85, 96, 97, 100, 101, 111, 113, 119, 121, 125) + d(5, 12, 13, 29, 37, 41, 44, 46, 47, 60, 71, 73, 77, 78, 79, 89, 93, 99, 103, 104, 105, 106, 107, 108, 109, 110, 117, 120, 124, 127)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	6,92	19	20712	50
Dual	1,60	8	21358	50
Gomo2	6,97	19	22504	50

Tabela 5.98: Comparação dos programas: Gomo2, Branch2 e Dual.

5.99 - 18

$f(A, B, C, D, E, F, G) = \sum m(5, 13, 29, 43, 45, 55, 69, 76, 78, 89, 93, 101, 105, 108, 113, 121, 124, 125) + d(12, 37, 41, 44, 46, 47, 57, 60, 61, 71, 73, 77, 79, 85, 96, 97, 99, 100, 103, 104, 106, 107, 109, 110, 111, 117, 119, 120, 127)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	9,72	22	31680	50
Dual	2,03	10	26566	50
Gomo2	9,72	22	33960	50

Tabela 5.49: Comparação dos programas: Gomo2, Branch2 e Dual.

5.100 - 20

$f(A, B, C, D, E, F, G) = \sum m(16, 25, 32, 55, 58, 65, 80, 82, 89, 97, 100, 105, 114, 115, 118, 119, 121, 122, 124, 127) + d(1, 17, 33, 34, 37, 48, 49, 50, 51, 53, 57, 69, 73, 81, 83, 84, 85, 90, 91, 96, 101, 107, 109, 112, 113, 116, 117, 120, 123, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	6,65	18	20288	50
Dual	1,76	9	21238	50
Gomo2	6,70	18	22208	50

Tabela 5.100: Comparação dos programas: Gomo2, Branch2 e Dual.

5.101 - 20

$f(A, B, C, D, E, F, G) = \sum m(1, 25, 33, 48, 53, 55, 58, 65, 83, 85, 91, 96, 114, 116, 118, 119, 123, 125) + d(16, 17, 32, 34, 37, 49, 50, 51, 57, 69, 73, 80, 81, 82, 84, 89, 97, 99, 100, 101, 105, 107, 109, 112, 113, 115, 117, 120, 121, 122, 124, 127)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	10,27	21	28960	50
Dual	2,20	10	28470	50
Gomo2	10,33	21	30832	50

Tabela 5.101: Comparação dos programas: Gomo2, Branch2 e Dual.

5.102 - 20

$f(A, B, C, D, E, F, G) = \sum m(1, 25, 33, 48, 50, 53, 55, 69, 73, 82, 84, 91, 99, 101, 113, 115, 118, 123, 127) + d(16, 17, 32, 34, 37, 49, 51, 57, 58, 65, 80, 81, 83, 85, 89, 96, 97, 100, 105, 107, 112, 114, 116, 117, 119, 120, 121, 122, 124, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	10	24	28248	55
Dual	2,41	13	28470	55
Gomo2	9,89	24	30984	55

Tabela 5.102: Comparação dos programas: Gomo2, Branch2 e Dual.

5.103 - 20

$f(A, B, C, D, E, F, G) = \sum m(1, 17, 32, 34, 37, 51, 57, 69, 81, 84, 91, 99, 101, 112, 115, 117, 118, 121, 124, 127) + d(16, 25, 33, 48, 50, 53, 55, 58, 65, 73, 80, 82, 83, 85, 89, 96, 97, 100, 105, 107, 109, 113, 114, 116, 119, 120, 122, 123, 125)$

Programa	Tempo	Iterações	Memória	Custos
Branch2	12,03	24	29848	65
Dual	2,69	13	30398	65
Gomo2	12,09	24	32408	65

Tabela 5.103: Comparação dos programas: Gomo2, Branch2 e Dual.



UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

Câmpus de Ilha Solteira
Programa de Pós-Graduação em Engenharia Elétrica
Av. Brasil Centro, 56
15385-000 Ilha Solteira - SP
www.dee.feis.unesp.br

