UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ANDERSON SANTOS DA SILVA

Um método de equivalência de funções Booleanas através de grafos bipartidos

Trabalho de Graduação.

Prof. Dr. Renato Ribas
Orientador

Porto Alegre, dezembro de 2013.

AGRADECIMENTOS

Gostaria de agradecer aqui a todos que de alguma forma me ajudaram e contribuíram para minha formação em Ciência da Computação.

Primeiramente, gostaria de agradecer a Deus, pela força e tudo que sempre me proporcionou.

A meus pais e irmãos, que sempre acreditaram em mim e me deram força. Eles me ensinam todos os dias o que muitas vezes eu não encontro nos livros que tanto gosto de ler e escrever.

A meus amigos de graduação, Alexandre, Guilherme, os dois Lucas e Luan que compartilharam diversos trabalhos, RU's, discussões sobre lógica, filosofia, etc.... Foram, sem dúvida, a parte mais importante da minha graduação.

Aos diversos professores que tive na graduação, em especial a Renato Perez Ribas e André Reis, meus orientadores por muito tempo. Acreditaram no trabalho exposto aqui e sempre me deram força.

Ao laboratório LOGICS inteiro. Grandes amigos a que tive o prazer de ficar muito tempo junto.

A Vinicius Callegaro, que realmente me ensinou como ser um cientista da computação. Me ensinou que grafos são estruturas de dados formidáveis. Me ensinou a ter paciência agüentando meus erros e discussões sobre algoritmos malucos. Com certeza um grande amigo.

Enfim, a todos que de alguma forma, em algum momento, participaram dos bons momentos que passei durante esta etapa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

P	Permutação
NP	Negação - Permutação
NPN	Negação – Permutação - Negação
GBB	Grafo Booleano Bipartido
BDD	Binary Decision Diagram

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Atualmente uma das etapas mais críticas no fluxo de projeto de circuitos integrados é a etapa denominada mapeamento tecnológico. Essa dificuldade se deve ao fato que esta etapa precisa resolver um problema NP-completo denominado equivalência Booleana. A proposta deste trabalho é a de acelerar esse processo representando uma função Booleana através de um grafo bipartido, e utilizar esta estrutura para calcular P-equivalência por permutação, uma etapa da equivalência Booleana. O grafo bipartido gerado é denominado Grafo Bipartido Booleano, ou GBB. Diversos métodos de resolução são apresentados e uma discussão quanto à qualidade deles é exposta. Os resultados mostram que essa estrutura é correta e que pode ser utilizada para calcular eficientemente equivalência por permutação. Trabalhos futuros vão desde estender esta abordagem para outros tipos de equivalência até a utilização do GBB para resolução de outros problemas na área de síntese de circuitos digitais.

Palavras-Chave: Equivalência Booleana, equivalência-P, mapeamento tecnológico, grafos bipartidos, isomorfismo de grafos.

A method for Boolean functions equivalence through bipartite graphs

ABSTRACT

Currently one of the most critical steps in the design flow of integrated circuits is the step called technology mapping. This difficulty is due to the fact that this step needs to solve an NP-complete problem called Boolean matching. The purpose of this work is to accelerate this process representing a Boolean function by a bipartite graph, and use this approach to calculate Boolen equivalence by permutation, a step of Boolean matching. The bipartite graph generated is called Bipartite Boolean Graph, or GBB. Various methods of resolution are presented and a discussion about the quality of them is exposed. The results show that this structure is correct and can be used to efficiently compute equivalence by permutation. Future work will extend from this approach to other types of equivalence to the use of GBB to solving other problems in the field of synthesis of digital circuits.

Keywords: Boolean Mathcing, P-matching, technology mapping, bipartirte graph, graph isomorphism.

1 INTRODUÇÃO

A metodologia de projeto de circuitos integrados digitais mais utilizada atualmente é denominada *standard cell*. A ideia central desta metodologia é de fazer a síntese automática de circuitos integrados digitais utilizando ferramentas CAD (software de apoio) e deste modo utilizar portas lógicas pré-implementadas para a construção de circuitos mais complexos. Os problemas enfrentados por esta metodologia impactam no desenvolvimento tecnológico em geral, pois questões como qualidade e eficiência são centrais nesse contexto.

Dentre as diversas etapas presentes nessa metodologia, a síntese lógica representa uma etapa importante. Sua atuação vai desde a definição lógica do circuito a ser implementado (VHDL) até a definição das portas lógicas utilizadas no circuito final. Diversas otimizações são feitas aqui, sejam elas dependentes ou não de tecnologia.

O processo de definição das portas lógicas utilizadas no circuito final é denominado mapeamento tecnológico. O mapeamento tecnológico é um processo crítico, pois precisa buscar por equivalência entre funções Booleanas. O problema de busca por equivalência entre funções Booleanas é denominado de *equivalência Booleana* e é NP-completo no caso geral (De Micheli, 1994). Logo, como o mapeamento tecnológico é uma etapa da síntese lógica, esta também se torna um processo crítico.

Existem diversos métodos para equivalência Booleana, em parte por que este é um processo puramente computacional (e não elétrico). Assim, diversas formas de representação do problema são possíveis e frequentemente abordagens utilizando grafos, fatoração e outros aspectos teóricos surgem para tratar o problema.

Os principais desafios enfrentados se concentram nas complexidades dos problemas envolvidos. Funções Booleanas em geral crescem exponencialmente se considerarmos sua quantidade de variáveis e a equivalência Booleana utiliza permutações como operação mais básica. Logo, estamos diante de problemas exponenciais cujo tratamento passa a ser proibitivo para instâncias grandes.

A proposta desse trabalho é a de oferecer uma nova estrutura de representação para funções Booleanas que permita resolver o problema de equivalência Booleana, especificamente a etapa de equivalência por permutação de entradas (equivalência-P), de forma eficiente. A ideia básica á de utilizar um grafo bipartido como forma de representação e deste modo utilizar informações de topologia dos grafos envolvidos para resolver a equivalência Booleana.

A estrutura deste é como segue. No capítulo 2 são apresentados conceitos gerais sobre álgebra e funções Booleanas. Este capítulo é importante, pois apresenta tabelas-verdade e outras terminologias importantes para o entendimento do trabalho. O capítulo 3 apresenta conceitos sobre a teoria dos grafos. Este capítulo é necessário para definir os

termos utilizados ao longo do trabalho quanto à teoria dos grafos. O capítulo 4 discute a metodologia s*tandard cell* ressaltando os principais desafios enfrentados. A importância deste capítulo é grande, pois é deste contexto que surge a motivação para a estrutura de dados proposta nesse trabalho. O capítulo 5 propõe o grafo bipartido Booleano (GBB), mostra exemplos e teoremas úteis as ideias desenvolvidas. O capítulo 6 ilustra diversas formas de resolução de equivalência-P com esta estrutura de dados e discute sua viabilidade. Em particular um algoritmo eficiente é proposto. O capítulo 7 apresenta conclusões e trabalhos futuros.

2 ÁLGEBRA E FUNÇÕES BOOLEANAS

Ao longo dos próximos três capítulos será apresentada uma revisão teórica básica de diversos conceitos importantes para o entendimento deste trabalho. Este capítulo faz uma revisão sobre conceitos em torno da álgebra e funções Booleanas.

2.1 Álgebra Booleana

A noção de valor lógico esta intimamente ligada à noção de sentença na lógica clássica. Uma *sentença lógica* ou *proposição* é uma frase afirmativa que é *verdadeira* ou *falsa*, mas nunca, *verdadeira* e *falsa,* ao mesmo tempo. Toda sentença possui um valor lógico associado e este pode ser representado de diversas formas. As formas mais usuais são V e F ou 1 e 0, representando *verdadeiro* e *falso*, respectivamente.

O valor lógico de uma sentença é *verdadeiro* caso a sentença seja verdadeira e *falso*, caso contrário. O princípio do terceiro excluído (Morais Filho, 2007) garante que não possa haver uma terceira opção quanto a isso.

Exemplo 1:

"Hoje esta chovendo" - é uma proposição.

"Este é um número primo" - não é uma proposição.

Considerando a existência de somente dois valores lógicos distintos, o conjunto desses valores lógicos é denotado por B = {0,1}. A definição de uma álgebra sobre o conjunto B necessita de operadores sobre os elementos de B (De Micheli, 1994) de modo que diversas propriedades desejáveis no sistema obtido sejam garantidas. As operações mais básicas que podem ser definidas sobre os elementos de B representam a noção de *conjunção* e *disjunção*, representadas por "*" e "+", respectivamente. Cada uma dessas operações possui um elemento neutro (a conjunção possui 1 como elemento neutro e a disjunção possui 0 como elemento neutro) e são comutativas e distributivas.

A adição de uma terceira operação representando a noção de complemento estende consideravelmente a álgebra definida anteriormente. Todo elemento $a \in B$ possui um complemento denotado por *a'* tal que $a+a' = 1$ e $a * a' = 0$. A operação de complemento é denotada por "!" e é uma função com somente um argumento que toma um elemento *a* e retorna o seu complementar *a'*. Essas três operações, "!", "+" e "*" são consideradas operações primitivas e são freqüentemente utilizadas na formação de outras operações mais complexas como o conceito de *implicação* ou *ou-exclusivo*.

Essa álgebra é denominada álgebra Booleana. A álgebra Booleana foi estudada principalmente por George Boole (Menezes, 2005) em torno de 1850 e sua contribuição foi tanta dentro deste campo que ela recebeu seu nome. Com o objetivo de desenvolver regras algébricas para o raciocínio lógico, semelhantes às regras algébricas existentes para o raciocínio numérico, diversas propriedades úteis foram descobertas. Algumas destas propriedades são apresentadas na Tabela 1.1. Hoje em dia, a álgebra Booleana tornou-se um dos pilares fundamentais para a eletrônica de computadores.

Tabela 1.1: Algumas propriedades da Álgebra Booleana

Propriedade	+	*
Comutatividade	A + B = B + A	A * B = B * A
Associatividade	(A + B) + C = A + (B + C)	(A * B) * C = A * (B * C)
De Morgan	!(A + B) = !A * !B	!(A * B) = !A + !B

2.2 Funções Booleanas

Uma função Booleana é uma função $f: B^n \rightarrow B^m$, onde $m,n \in N^+$.

O domínio de uma função Booleana é representado por B^n. O conjunto B^n é o resultado de um produto cartesiano *n-ário* em *B*. Portanto, o domínio de uma função Booleana é uma tupla de *n* componentes.

Exemplo 2:

Para n=2, o domínio de f: $B^n \rightarrow B^m$ *fica sendo:*

$B^2 = B \times B = \{0,1\}x\{0,1\} = \{(0,0), (0,1), (1,0), (1,1)\}$

A imagem de uma função Booleana é representada por B^m. Funções Booleanas com $m > 1$ são ditas funções Booleanas com múltiplas saídas. O caso $m=1$ representa funções Booleanas com uma única saída.

Quando *f* é total, ou seja, definida para todo o seu domínio, diz-se que *f* é completamente especificada. Caso contrário, se *f* está definida somente para um subconjunto de seu domínio, dizemos que *f* é incompletamente especificada. As *funções incompletamente especificadas* têm uma grande importância dentro do estudo de funções Booleanas (Sasao, 1999). Os pontos onde *f* não esta definida possui a denominação de *don't care* (que significa "não importa"). Funções Booleanas que admitem *don't cares* estendem o conjunto *B* de modo a representar essa situação, o tornando *B= {0, 1, X}*, onde *X* representa o *don't care*.

A composição de funções Booleanas também é uma função Booleana e, portanto, podemos compor os operadores "+" e "*" livremente, construindo funções Booleanas mais complexas com base nas funções elementares. Pela definição acima, percebemos que as operações Booleanas "+" e "*" também são funções Booleanas.

O domínio de *f* é um espaço Booleano de dimensão *n*. Um ponto dentro deste espaço gerado é um vetor ou tupla de dimensão *n* constituído de valores binários. O tamanho dessas tuplas define o número de *variáveis* de uma função Booleana, ou seja, o número de elementos existentes dentro de uma tupla do domínio de *f*. O conceito de variável está intimamente ligado ao conceito de dimensão do espaço Booleano.

Exemplo 3:

$f : B^5 \rightarrow B$ tem um domínio constituído de tuplas $a = (a_1, a_2, a_3, a_4, a_5)$ de 5 componentes. Neste contexto, a_1 é uma variável de *f*.

A introdução de variáveis associadas a funções Booleanas permite uma relação forte com a álgebra Booleana, permitindo a utilização das propriedades referentes a esta álgebra.

Outros conceitos também são fundamentais quando passamos a considerar as variáveis de uma função Booleana. Chamamos de *mintermo* a interpretação dos elementos de uma tupla como uma conjunção. Chamamos de *maxtermo* a interpretação dos elementos de uma tupla como uma disjunção.

Exemplo 4:

Considere a tupla a = (a_1, a_2, a_3, a_4, a_5) presente do domínio de $f: B^5 \rightarrow B$

$a_1 * a_2 * a_3 * a_4 * a_5$ é o mintermo construído de elementos de a

$a_1 + a_2 + a_3 + a_4 + a_5$ é o maxtermo construído de elementos de a

Ao longo do texto, utilizaremos o termo função Booleana para representar uma função Booleana completamente especificada com uma única saída. A menos quando for explicitamente informado do contrário, estaremos lidando somente com este tipo de função Booleana.

2.2.1 Formas de representação

Existem diversas formas para representar uma função Booleana. Cada uma possui limitações e propriedades interessantes dependendo do contexto utilizado. Um grande problema comum a todas, que em parte é inerente à própria definição de funções Booleanas, é o tamanho de seu domínio. Como a cardinalidade do conjunto *B*, denotada por $|B|$, é tal que $|B| = 2$, o produto cartesiano *n-ário* utilizado para gerar o domínio B^n, produzirá um conjunto de tamanho igual a 2^n.

Como cada elemento no domínio de *f* pode assumir um valor em $B=\{0,1\}$, temos que cada uma das 2^n possibilidades do domínio pode ser 0 ou 1, ou seja, isso resulta em 2^{2^n} possibilidades de funções Booleanas utilizando este domínio B^n. Essa natureza exponencial dificulta a obtenção de abordagens eficientes para representação de todo o universo de funções Booleanas, visto que, para *n* tão pequeno quanto $n=5$, obtemos 2^{2^5} = 4.294.967.296 possibilidades.

2.2.1.1 Tabelas-verdade

Talvez a abordagem mais natural para representar uma função Booleana seja uma tabela-verdade. Essa construção parte do princípio de enumerar em uma tabela todos os elementos do domínio da função Booleana e assinalar um valor de saída associado a esta entrada.

A construção é muito simples. Considere $n=2$:

$$B^2 \rightarrow B$$

$$\{(0,0), (0,1), (1,0), (1,1)\} \rightarrow B$$

$$\{(0,0), (0,1), (1,0), (1,1)\} \rightarrow \{0,1\}$$

$$\{(0,0), (0,1), (1,0), (1,1)\} \rightarrow \{a,b,c,d\}, \text{ onde } a,b,c \text{ e } d \in B$$

Vendo isto em forma de tabela, temos que o resultado mostrado na figura 2.1.

A	B	Saída
0	0	a
0	1	b
1	0	c
1	1	d

Figura 2.1: Tabela-verdade genérica para duas variáveis.

Cada coluna nomeia sua correspondente variável. Cada linha pode representar um *mintermo* ou *maxtermo* contanto que se use a mesma convenção para todas as linhas na tabela-verdade. A instanciação de valores para as incógnitas *a,b,c* e *d* define uma função Booleana sobre B^2.

Note que não existem restrições rígidas quanto ao ordenamento que as tuplas de uma função Booleanas deve seguir. Uma convenção aceitável é de ordenar o domínio lexicograficamente iniciando pelos primeiros componentes (note que isto foi feito na figura 2.1). Procedendo desta forma, uma ordenação muito semelhante à ordenação dos números naturais surge possibilitando a representação de uma linha da tabela-verdade como um número inteiro na base binária. A figura 2.2 ilustra um exemplo disto.

Exemplo 5:

A	B	F
1	1	1
0	0	1
0	1	1
1	0	1

A	B	F
0	0	1
0	1	1
1	0	1
1	1	1

Figura 2.2: Exemplo de ordenamento lexicográfico para as linhas de uma tabela-verdade.

Uma tabela-verdade possui um número exponencial de linhas de acordo com o domínio da função Booleana representada. Em geral, utiliza *n+1* colunas, onde *n* representa a quantidade de variáveis utilizadas pela função adicionado de uma coluna para representar sua saída. Logo após, resta enumerar todas as possibilidades de configurações para as variáveis de entrada, fazendo com que o tamanho de uma tabela-verdade simples seja $(2^n + 1)(n+1)$.

2.2.1.2 Representação por Hexa

Considerando uma tabela-verdade com suas linhas ordenadas é possível representar a saída desta tabela através de um número inteiro na base binária. A construção também é muito simples, basta justapor cada elemento presente na saída partindo das linhas superiores até as inferiores (ou das inferiores para superiores).

Procedendo deste modo, o resultado final será uma sequência de 0 e 1, que pode ser interpretada como um número binário. Tendo um número binário é possível convertê-lo para qualquer outra base numérica representando uma função Booleana de uma forma muito simples.

Exemplo 6:

Considere a função Booleana representada pela tabela-verdade na figura 2.3 conhecida como a tabela-verdade do operador "+".

A	B	Saída
0	0	0
0	1	1
1	0	1
1	1	1

Figura 2.3: Tabela-verdade para o operador +.

Considerando a construção da sequência de bits das posições superiores para as inferiores, temos o seguinte resultado:

0111 = 0x13

Considerando a ordem, iniciando nas posições inferiores e seguindo para os superiores, temos:

1110 = 0x31

Historicamente, a base mais utilizada para este tipo de representação é a hexadecimal devido à conversão direta a partir do sistema binário e ao alto grau de compactação devido à quantidade elevada de símbolos utilizadas por esta base.

Essa representação sofre do mesmo problema de crescimento exponencial do domínio de uma função Booleana com o aumento do número de variáveis. Dada uma função Booleana com n variáveis são necessários 2^n bits para representar esta função como um número hexadecimal em meio computacional, o que pode ser proibitivo para um valor elevado de n.

2.2.1.3 Representação por expressões

Outra forma de representação interessante para uma função Booleana é a de expressão lógica. De fato, existem diversas maneiras de gerar uma expressão lógica para uma função Booleana (De Micheli, 1994). Talvez a forma mais intuitiva seja diretamente de sua tabela-verdade. Para tanto, basta interpretar a tabela-verdade como uma soma-de-produtos (SOP, do inglês, "sum of products") ou produto-de-somas (POS, do inglês, "product of sums").

No caso de soma-de-produtos, cada linha se torna um produto de variáveis. Cada variável que está assumindo o valor zero é representada juntamente como seu complemento. Cada variável que esta assumindo o valor 1 é representada normalmente. Logo após, basta colocar um operador de disjunção entre cada linha para gerar a soma de produtos de uma função Booleana. A figura 2.4 mostra um exemplo para uma soma-de-produtos.

A	B	F	
0	**0**	**1**	!A*!B + A*!B + A*B
0	**1**	**0**	
1	**0**	**1**	
1	**1**	**1**	

Figura 2.4: Tabela-verdade e sua representação por soma-de-produtos.

Linhas onde *F=0* não participam da expressão lógica final, e é por isso que o termo *!A*B* não aparece na expressão da figura 2.4.

O caso de produto-de-somas segue uma idéia parecida e pode ser obtido através da negação de uma soma-de-produtos. Cada linha ao invés de ser um produto, será uma soma e todas as linhas são ligadas através de produtos.

Essa representação é canônica e possui um mapeamento um para um para a tabela-verdade. Métodos de minimização (De Micheli, 1994) recebem uma expressão desse tipo e aplicam regras da álgebra Booleana ou então conceitos mais sofisticados para gerar expressões mínimas, ou seja, reduzindo algum critério de otimização como, por exemplo, número de variáveis aparecendo na forma final. Quando isso acontece, diz-se que a expressão está em sua forma fatorada.

3 TEORIA DOS GRAFOS

O estudo de propriedades geométricas que não são alteradas por mudanças de forma oferece certamente um conjunto de questões difíceis. O primeiro matemático a lidar com este tipo de pensamento foi Leonard Euler resolvendo o problema que tempos depois passou a ser conhecido como caminho de Euler (Boaventura Netto, 2006). Após esse primeiro impulso, mais estudos nesse campo permitiram o surgimento do que hoje chamamos de Teoria dos Grafos.

A gama de aplicações a que a teoria dos grafos se adequa é enorme. E em parte, isso se deve ao fato do conceito por trás de toda a teoria ser muito intuitivo. Entre um conjunto de objetos pertencentes a determinado domínio representam-se as relações entre eles usando ligações, retas traçadas no espaço ligando os elementos entre si. Esse tipo de construção é tão antigo que mesmo os antigos povos já utilizavam construções semelhantes quando imaginavam constelações no céu. A figura 3.1 ilustra um exemplo dessa idéia.

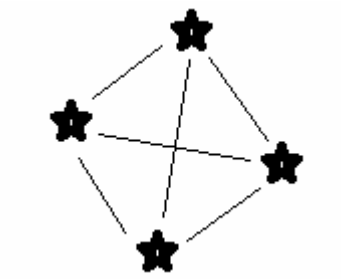

Figura 3.1: Constelação representada como um grafo simples.

Contudo, com o objetivo de modelagem da realidade e construções de ideias mais sofisticadas em torno dessa abordagem, foi necessário um suporte matemático preciso. Ao longo do tempo, diversos teoremas e resultados interessantes foram descobertos em torno dessa teoria.

3.1 Conceitos Básicos

Os resultados da teoria dos grafos mais essenciais para o entendimento do método proposto nesse texto são apresentados nas sub-seções seguintes.

3.1.1 Definição e Terminologia

Um grafo é um par ordenado G = (V, E), onde:

V – é um conjunto de elementos chamados nodos, e

E – é um conjunto de elementos chamados arestas.

No grafo apresentado na figura 3.1, os nodos são as estrelas e as arestas são as ligações entre as estrelas.

Dados dois nodos u e $v \in$ V, se existe uma aresta em E que tem como extremidade u e v então diz–se que u e v são *adjacentes*. Na figura 3.2.a os nodos pretos são adjacentes enquanto o nodo cinza não é adjacente a algum nodo. Um *laço* é uma aresta tal que suas extremidades u e v são iguais, ou seja, $u=v$. Muitas vezes esse tipo de construção é denominada *endoarco*. Na figura 3.2.b, o nodo preto possui um *endoarco*.

Duas arestas com a mesma extremidade são chamadas de *arestas paralelas*. Um *grafo simples* é um grafo que não tem laços nem arestas paralelas. Um *grafo completo* é um grafo no qual para cada par de nodos u e $v \in$ V existe uma aresta os ligando. Um *subgrafo* é um conjunto de nodos e arestas que são um subconjunto de um outro grafo original e que respeitam as mesmas relações de adjacência impostas no grafo original.

Um *caminho* entre os nodos n_0 e $n_k \in V$ é uma seqüência de nodos $n_0, a_0, n_1, a_1, n_2, a_2, \ldots n_k, a_k$, tal que a aresta a_i tem como extremidades os nodos n_i e n_{i+1}. Um *ciclo* é um caminho de um nodo $u \in V$ para ele mesmo tal que nenhum aresta apareça mais de uma vez, sendo que o único nodo que se repete é u e essa repetição somente ocorre nas extremidades do ciclo. A figura 3.2.c ilustra um exemplo de ciclo.

Um *grafo completo* é um grafo que possui uma aresta para cada par (u,v) de nodos $\in V$. Geralmente, grafos completos são denominados por K_n, onde n é um valor natural que indica o número de nodos que o grafo contém. A figura 3.2.c ilustra um grafo completo.

Dado um grafo $G= (V,E)$ e um nodo $v \in V$, denomina-se grau de um nodo o número de arestas que têm v como extremidade. O nodo cinza na figura 3.2.d possui grau 3.

Dado um grafo $G =(V, E)$, G é dito conexo se existe um nodo $v \in$ V tal que para todo $u \in$ V, existe um caminho no grafo iniciando em v e terminando em u. Os grafos na figura 3.2.c e 3.2.d são conexos. O grafo na figura 3.2.e não é conexo, pois o nodo em destaque não respeita o critério de conectividade.

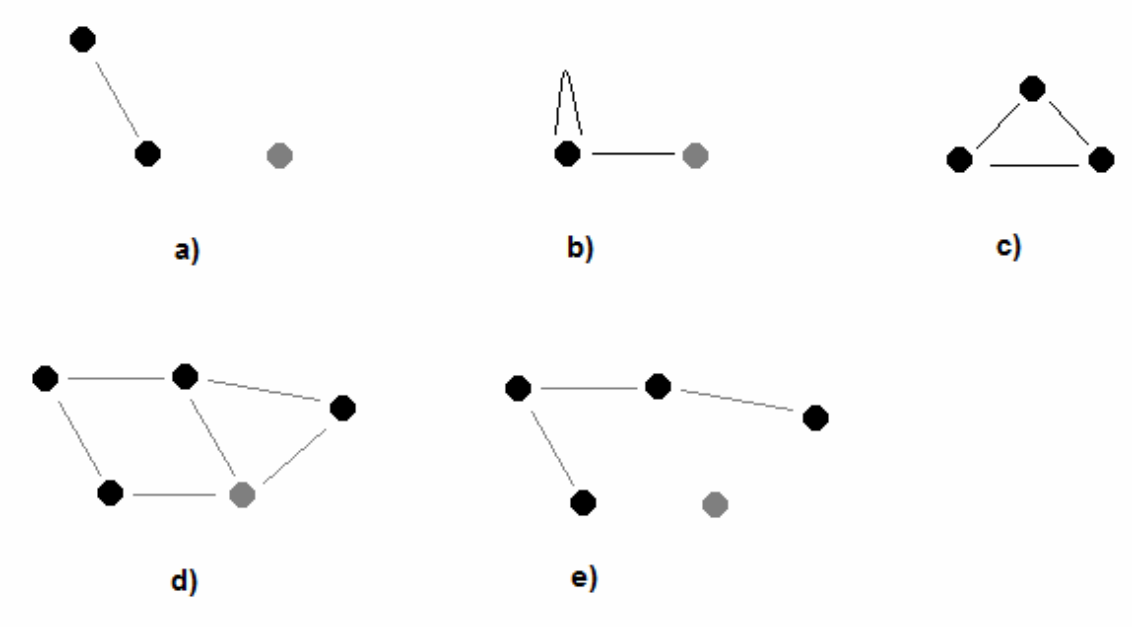

Figura 3.2: Diversos exemplos de grafos.

3.1.2 Grafos Rotulados e Formas de Representação

Um grafo *G=(V,E)* é dito rotulado caso tenha nomes associados a seus nodos ou arestas.

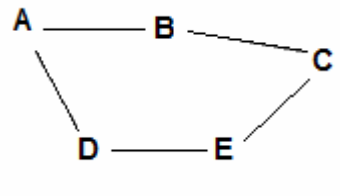

Figura 3.3: Grafo rotulado.

A figura 3.3 representa um grafo rotulado com rótulos *{A, B, C, D, E}* no local de seus nodos. Grafos rotulados são importantes, pois associam algum tipo de nome a estrutura de seus nodos permitindo a criação de estruturas de dados eficientes para armazenar os nodos.

Em meio computacional, grafos podem ser representados de diversos modos. As duas formas mais comuns são: *matriz de adjacência* e *lista de adjacência*.

3.1.2.1 Matrizes de adjacência

Uma matriz de adjacência é uma estrutura de dados simples para representação de um grafo qualquer com *n* nodos. A ideia geral é a de rotular os nodos do grafo de algum modo arbitrário e colocá-los como índices de uma matriz *n x n*. O elemento $a_{i,j} = 1$ se a aresta *(i,j)* pertence ao grafo. Caso contrário, $a_{i,j} = 0$.

A figura 3.4 mostra um exemplo de matriz de adjacência. Note que se um grafo tem *n* nodos, então sua matriz de adjacência terá n^2 posições preenchidas com 0 ou 1.

	A	B	C	D	E
A	0	1	0	1	0
B	1	0	1	0	0
C	0	1	0	0	1
D	1	0	0	0	1
E	0	0	1	1	0

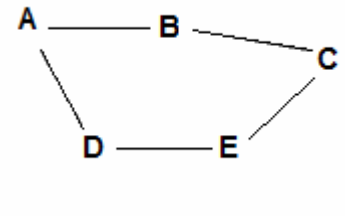

Figura 3.4: Exemplo de matriz de adjacência.

Matrizes de adjacência são muitos semelhantes a tabelas-verdade. Essa semelhança é um dos fundamentos da forma de representação de funções Booleanas proposta nesse trabalho.

Matrizes de adjacência sofrem por desperdiçar muita memória. Para grafos esparsos, ou seja, com poucas arestas, muitas células da matriz ficarão vazias. Para grafos com milhares de nodos, seu tamanho quadrático pode se tornar proibitivo.

3.1.2.2 Listas de adjacência

Listas de adjacência são estruturas de dados que resolvem o principal problema de matrizes de adjacência, ou seja, a representação de grafos esparsos. A ideia aqui é rotular os nodos do grafo de forma arbitrária, colocá-los numa lista encadeada, sendo que cada nodo possui uma lista individual de seus nodos vizinhos. Desse modo, somente é preciso salvar a informação de arestas existentes.

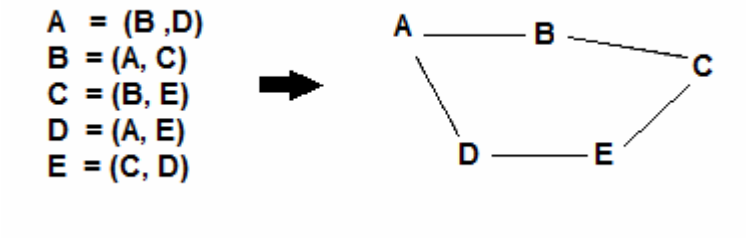

Figura 3.5: Exemplo de lista de adjacência

3.1.3 Grafos Bipartidos

Um grafo $G=(V,E)$ é dito bipartido caso o seu conjunto V possa ser dividido em dois subconjuntos disjuntos V_1 e V_2 tal que dois nodos u e v são adjacentes se e somente se $u \in V_1$ e $v \in V_2$ (ou vice-versa).

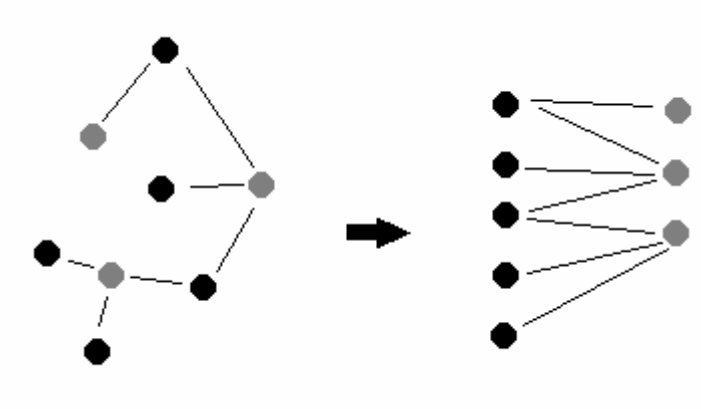

Figura 3.6: Exemplo de grafo bipartido.

Na figura 3.6 podemos visualizar um grafo bipartido. Grafos bipartidos são uma classe especial de grafos com diversas aplicações. Em especial são muito utilizados na modelagem de problemas de acoplamentos e na teoria dos códigos com os grafos de Tanner (Rochol, 2012).

Grafos bipartidos completos podem ser representados por $K_{n,m}$, sendo que n e m são valores quaisquer naturais. Por exemplo, o grafo bipartido completo $K_{3,3}$ está representado na figura 3.7.

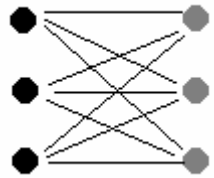

Figura 3.7: Exemplo de grafo bipartido completo.

3.1.4 Isomorfismo

Dois grafos $A = (V_1, E_1)$ e $B = (V_2, E_2)$ são ditos isomorfos caso existam bijeções $f: V_1 \rightarrow V_2$ tal que para toda aresta $(u,v) \in E_1$, $(f(u), f(v)) \in E_2$.

Informalmente, dois grafos são isomorfos quando um deles puder ser rearranjado espacialmente de modo a se tornar o outro, isto é, com o mesmo desenho.

As figuras 3.8 e 3.9 ilustram caso de grafos isomorfos.

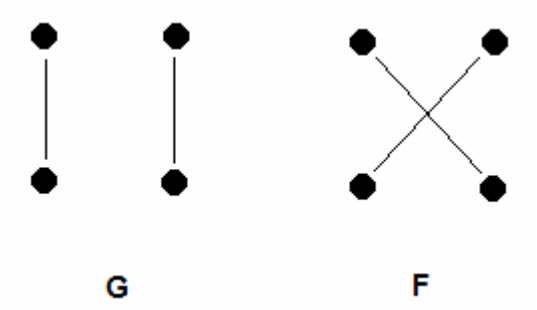

Figura 3.8: Exemplo simples de grafos isomorfos.

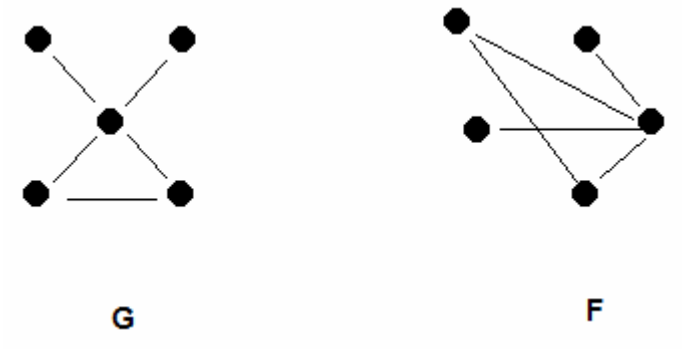

Figura 3.9: Exemplo mais complexo de grafos isomorfos.

Procurar por isomorfismo no caso geral é um problema difícil. Não existe um algoritmo de tempo polinomial para resolver se dois grafos são isomorfos tampouco é provado que não exista tal ou que esse problema seja NP-Completo (Skiena, 2008).

O melhor algoritmo conhecido até então é de um trabalho de 1981 (Luks, 1981). Apesar de satisfatório para algumas instâncias, esse algoritmo possui tempo exponencial no pior caso. Embora estudos recentes tenham demonstrado que grafos planares, árvores e mais algumas variações de grafos possuírem algoritmo em tempo polinomial para resolver isomorfismo, diversas classes importantes ainda não possuem soluções eficientes.

O problema de Isomorfismo de grafos é um dos 12 problemas listados por Garey & Johnson, em 1979, que ainda tem sua complexidade em aberto. A importância desse problema é tanta dentro do ramo da Ciência da Computação que uma classe especial de problemas foi criada de modo a auxiliar o seu estudo. A classe de problemas GI é a classe de problemas que possuem uma redução em tempo polinomial ao problema de isomorfismo de grafos (Skiena, 2008). Diversos problemas se encaixam dentro desta classe.

Todavia, mesmo isomorfismo de grafos sendo um problema muito difícil, de fato, alguns critérios servem para detectar rapidamente quando dois grafos **não** são isomorfos e, portanto, podem ser utilizados em alguma abordagem algorítmica para "podar" o espaço de busca.

Para dois grafos A e B, eles **não** são isomorfos se:

- o número de nodos de A tem é diferente do número de nodos de B;
- o número de arestas de A é diferente do número de arestas de B;
- se um deles possui um ciclo e outro não;
- se um deles tem um nodo de grau *i* e o outro não;
- A é conexo e B não;
- A tem aresta paralelas e B não;
- A é conexo e B não;

Quando esses critérios são satisfeitos, nada pode ser afirmado, ou seja, os grafos poderão ou não ser isomorfos.

3.1.5 Grafos Planares

A ideia de grafo planar nos remete a noção de um mapa. Um grafo é planar se seu esquema puder ser traçado sobre o plano de modo que suas arestas não se toquem, salvo em seus nodos.

Existem diversos critérios e resultados interessantes sobre planaridade de grafos e uma grande discussão pode ser vista em (Skiena, 2008) e (Boaventura Netto, 2006). Para este trabalho um critério muito simples serve para os resultados utilizados durante este texto.

O grafo bipartido $K_{3,3}$ não é planar. Uma prova disto pode ser encontrada em (Boaventua Netto, 2006). Utilizando esse fato, o teorema de Kuratowski afirma o seguinte:

"Um grafo é planar se e somente se não tiver, como subgrafo um grafo 'homeomorfo' a K_5 ou $K_{3,3}$".

4 EQUIVALÊNCIA BOOLEANA E MAPEAMENTO TECNOLÓGICO

A metodologia de projeto de circuitos integrados digitais mais utilizada atualmente é denominada *standard cell* (Sasao, 1999). A idéia central desta metodologia é utilizar portas lógicas pré-implementadas para construção de circuitos mais complexos.

Um projeto de circuito integrado digital é descrito inicialmente de forma mais abstrata através de linguagens como VHDL ou Verilog (Sasao, 1999). Note que este é um nível de abstração elevado, sendo que o objetivo mais geral é o de especificar como os componentes se interconectam e se relacionam entre si. Após o projeto inicial, essa descrição passa por ferramentas de CAD (computer-aided-design) que interpretam os componentes descritos no projeto e constróem um modelo do circuito definindo seus blocos principais.

A construção real de um circuito acontece quando cada um de seus blocos principais é mapeado para uma implementação real. Como não é possível manter um conjunto de todos os circuitos reais possíveis, um conjunto reduzido geralmente é utilizado neste processo. Esse conjunto reduzido é formado por componentes otimizados ou simplesmente componentes que apresentam propriedades interessantes como baixo consumo de energia ou pouco atraso. Componentes com características semelhantes são agrupados em uma bibloteca.

É desejável que uma biblioteca seja abrangente, ou seja, o conjunto reduzido de circuitos presentes nela precisa cobrir qualquer bloco funcional requerido no processo de construção de um dado circuito. Nesse enfoque, é necessário que um componente de biblioteca possa alterar sua funcionalidade de acordo com o contexto em que é utilizado. Existem algumas operações Booleanas que podem ser aplicadas sobre um dado componente de modo a trocar a função Booleana oferecida por ele. Esse conjunto de operações tem o intuito de buscar por equivalência entre funções Booleanas.

O processo de escolha de quais componentes de biblioteca serão utilizados na construção de um circuito real se denomina Mapeamento Tecnológico. O mapeamento tecnológico é uma etapa importante da metodologia *standard cells* e oferece grandes desafios quanto a abordagens eficientes. Esses desafios se devem principalmente a necessidade do processo em resolver diversos problemas não triviais como utilizar o menor número de componentes, definir o tamanho das portas lógicas, executar a equivalência Booleana (um problema NP-Completo), entre outros.

4.1 Mapeamento Tecnológico

O fluxo *standard cell* é freqüentemente dividido como mostra a figura 4.1**.**

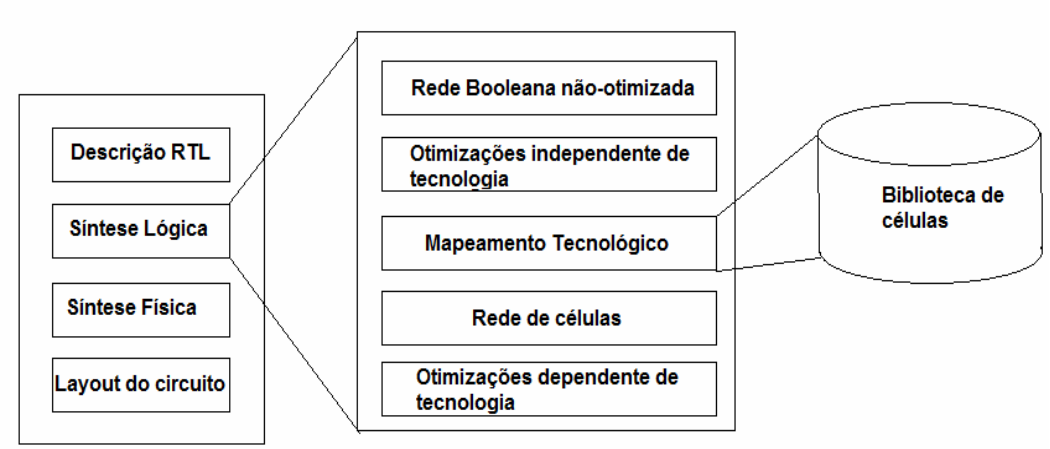

Figura 4.1: Fluxo standard cell.

Existem diversas etapas dentro deste fluxo. Para este trabalho, o interesse se concentra na etapa denominada s*íntese lógica*. O bloco síntese lógica engloba diversas subcategorias muito importantes dentro da construção de um circuito. Entre todas, o maior destaque fica em torno da etapa de *mapeamento tecnológico* devido a sua complexidade.

O mapeamento tecnológico utiliza uma biblioteca de componentes. Cada componente, ou célula como é a terminologia neste campo, é parte de um conjunto finito de células que implementam funções lógicas disponíveis para construção de um circuito. Nesse ponto fica evidente que quanto maior a variabilidade de células presentes dentro de uma biblioteca, melhor ela será. A variabilidade oferecida por ela culminará em construções mais otimizadas para um dado circuito, mas também definirá o que é possível ou não de construir com um determinado conjunto.

4.1.1 Decomposição lógica

Dado um circuito de entrada descrito por alguma linguagem de alto nível, a descrição de seus blocos principais pode ser representada por alguma estrutura de dados canônica de modo a permitir a seleção de pedaços desse componente promissores para mapeamento junto a biblioteca. Diversas estruturas de dados são possíveis e as mais freqüentes se resumem a DAG, árvores, AIG (De Micheli, 1994). A estrutura preferida é a que aumenta a granularidade dos blocos constituintes e favorece a procura por padrões.

A decomposição lógica em si manipula a estrutura de dados escolhida para representar o circuito de entrada de forma conveniente ao mapeamento tecnológico. Lembre-se que existem diversas formas de representação de um circuito e, portanto, o maior desafio desta etapa é o de prover a mesma solução para diversas representações de um mesmo circuito. A Fig 4.2 mostra um exemplo de decomposição típica, nesse exemplo, é utilizada a representação por árvore lógica onde é extraída a expressão lógica do circuito e esta é representada em forma de árvore lógica.

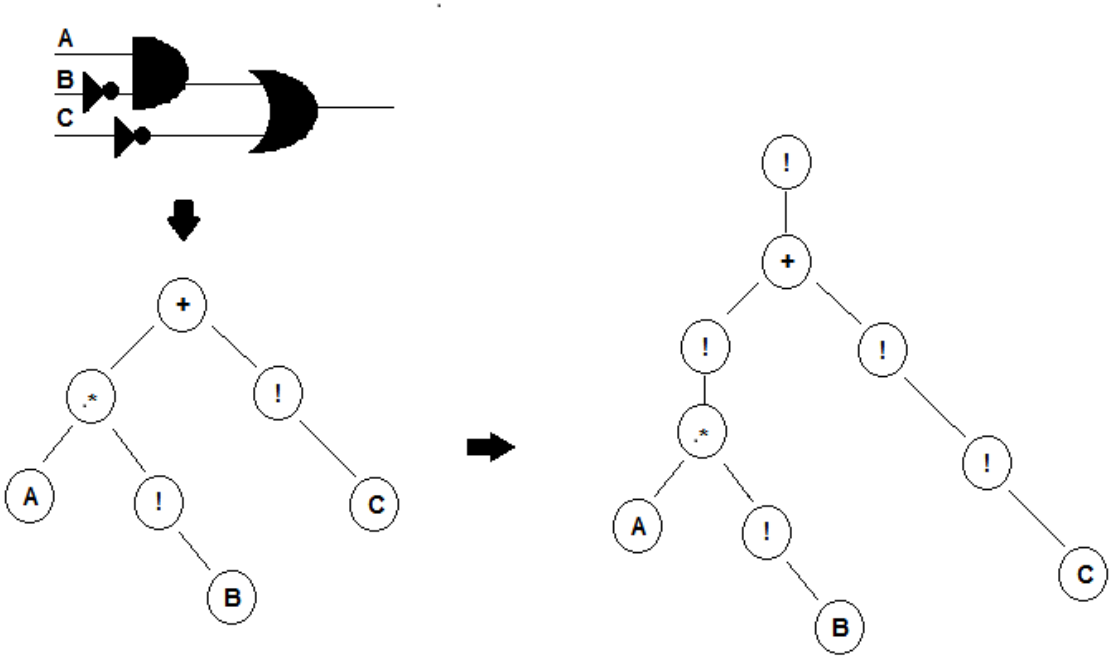

Figura 4.2: Decomposição lógica.

4.1.2 Cobertura e busca por equivalência

Após a etapa de decomposição lógica, o circuito está pronto para ser construído juntamente com os componentes presentes em uma biblioteca. Existem duas abordagens interessantes nesse ponto. É possível fazer busca por equivalência estrutural, ou seja, para um dado componente existente no circuito de entrada, busca-se uma célula na biblioteca que tenha a mesma estrutura. Ou então é possível procurar por equivalência Booleana, ou seja, procurar alguma célula na biblioteca que implemente a mesma função lógica do bloco funcional destino.

Ambas as abordagens tem vantagens e desvantagens. A equivalência estrutural sofre a desvantagem de depender da existência de uma estrutura equivalente dentro da biblioteca. A equivalência Booleana é um procedimento mais flexível, contudo sofre a restrição de não ser escalável por ser um problema NP-completo.

Por fim, a ultima etapa faz a escolha entre as células equivalentes ao componente na biblioteca de modo a cobrir todas as restrições impostas pelo projeto do circuito integrado. É um problema de otimização onde se deseja minimizar o custo sujeito a diversas restrições como minimizar a área ocupada pelo circuito final ou otimizar a velocidade de operação. A Figura 4.3 mostra um exemplo de escolha de componente a mapear na biblioteca.

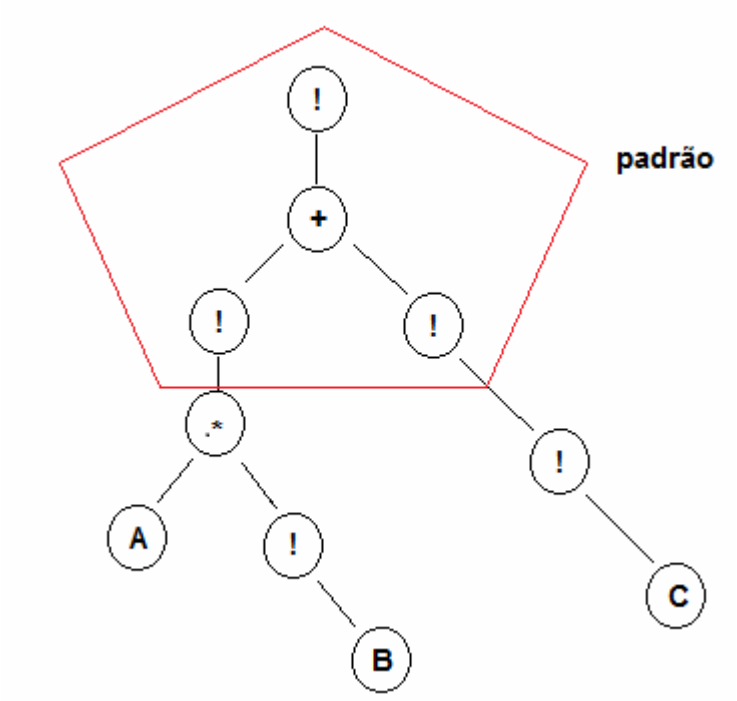

Figura 4.3: Escolha de componente para mapear junto a biblioteca.

4.2 Equivalência Booleana

A identificação de padrões através de métodos Booleanos é uma alternativa interessante dentro do contexto de mapeamento tecnológico. Contudo, mesmo sendo uma abordagem preferida, seu uso é proibitivo visto que o problema de equivalência Booleana é NP-completo para o caso geral (De Micheli, 1994). Diversas abordagens foram desenvolvidas para lidar com este tipo de restrição e diversos avanços foram alcançados (Benini, 1997). Contudo, os métodos que lidam com estes problemas estão limitados a circuitos com um número pequeno de variáveis.

Um algoritmo para tratar o problema denominado equivalência-P é apresentado neste trabalho. O problema de equivalência-P é um subcaso do problema maior denominado equivalência Booleana. Primeiramente vamos entender o porquê desse problema ser tão difícil e de fato estar na classe NP-completo, e logo após investigar estratégias de resolução.

4.2.1 Equivalência-P

O objetivo geral da operação de equivalência Booleana é o de buscar por equivalência funcional de um circuito A com algum circuito B. Assumindo que f_A seja a função lógica implementada pelo circuito A e f_B a função lógica implementada pelo circuito B, podemos dizer que o problema de equivalência Booleana resolve se f_A é equivalente a f_B.

Essa noção de equivalência entre funções Booleanas induz a ideia de que se f_A é equivalente a f_B então existe um modo de transformar f_A em f_B, ou seja, é possível que exista uma sequência finita de operações sobre f_A que acaba por transformar f_A em f_B. Essa sequência finita de operações pode ser vista como perturbações que fazemos no domínio de uma função Booleana de modo a alterar sua imagem.

Dadas duas funções Booleanas f_A e $f_{B,}$, a equivalência-P verifica se existe uma permutação nas variáveis de f_A que a torne em f_B. Portanto, a equivalência-P é um tipo de equivalência Booleana que utiliza permutações na busca por equivalência. De fato, esta operação de permutação nas variáveis de uma função muda sua imagem. Observe a figura 4.4.

Considere o circuito e a permutação mostrada na figura 4.4.

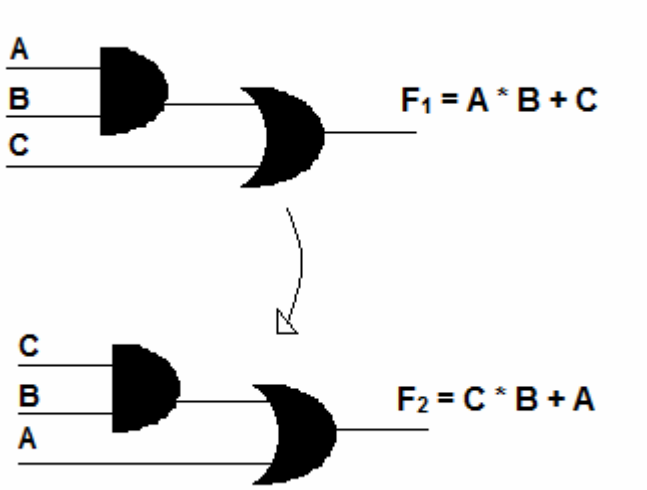

Figura 4.4: Circuito e uma possível permutação de suas entradas.

Note que somente as entradas foram modificadas, mas a estrutura do circuito não. Claramente, a imagem da função Booleana F_1 foi alterada por esta operação fazendo ela se tornar a função Booleana F_2. Neste caso, F_1= A*B+C e F_2 = C*B+A são ditas equivalentes.

Especificamente, se uma função Booleana f_A puder se tornar uma função Booleana f_B através da permutação de suas variáveis então f_A e f_B são ditas *funções equivalentes-P*. Dado uma função Booleana f_A é possível produzirmos todas as permutações em suas variáveis então gerar um conjunto de todas as funções equivalentes-P a f_A.

Supondo um universo de k funções Booleanas distintas de n variáveis, se retirarmos uma f_i deste universo e calcular todas as permutações de f_i sobre suas n variáveis então podemos retirar as $n!$ funções equivalentes a f_i do conjunto original e deixar as restantes. A redução do número de funções que efetivamente precisam ser guardados é enorme, e isso vem de encontro ao problema mencionado anteriormente no contexto de uma biblioteca de células reduzidas que represente um conjunto muito grande funções Booleanas. Essa abordagem, ao menos nesse enfoque, é muito promissora.

Sem dúvida, podemos pegar todo o espaço Booleano de n variáveis e representar ele por somente m funções Booleanas que garantidamente não são equivalentes entre si. Denomina-se classe P (ou *P-class*, do termo em inglês) o nome dado ao conjunto de funções com n variáveis tais que nenhuma função dentro deste conjunto é P-equivalente a outra neste mesmo conjunto. A classe P é interessante por sua propriedade de compactação de um universo total de tamanho $2^{2^{\wedge}n}$, para o espaço Booleano de dimensão n.

O maior problema quanto a esta abordagem é o fato de ser muito difícil verificar se duas funções Booleanas são equivalentes-P. Suponha que se deseja saber se f_A é equivalente-P a f_B. É claro que se o número de variáveis de f_A for diferente de f_B então ambas as funções não poderão ser equivalentes-P, pois permutar as variáveis de f_A de modo a chegar em f_B nunca poderia retirar ou inserir novas variáveis em f_B. Logo, para serem equivalentes-P, o primeiro critério que duas funções Booleanas candidatas precisam satisfazer é o de ter *o mesmo número de variáveis*.

Uma solução simples para detectar equivalência-P utiliza f_A como base e tenta todas as permutações nas variáveis de f_A de modo a atingir f_B. Considerando que ambas as funções Booleanas possuem n variáveis, essa abordagem utilizará $O(n!)$ passos, o que passa a ser proibitivo se $n > 10$. Diversas abordagens para cálculo de equivalência-P foram desenvolvidas (Benini, 1997), mas todas possuem a restrição pesada quanto a número de variáveis de entrada. Em especial, a seção 4.3 discute algumas delas.

A complexidade maior do problema de equivalência-P entre duas funções Booleanas se dá pelo fato de que este é um problema NP-completo (De Micheli, 1994), logo ainda não existe uma abordagem muito eficiente para resolver tal problema no caso geral.

4.2.2 Equivalência-NP e equivalência-NPN

Existem outras operações possíveis de fazer no domínio de uma função Booleana. A equivalência-NP, além de permutar as variáveis de uma função Booleana, ainda considera todas as negações possíveis em suas variáveis. Tal abordagem utiliza $O(2^n n!)$ passos, pois para cada negação possível nas variáveis de entrada deve-se testar todas as permutações de suas variáveis de modo a verificar equivalência-NP.

O caso de equivalência-NPN faz uma equivalência-NP, mas ainda considera a negação da saída da função Booleana base, ou seja, para cada uma das $2^n . n!$ possibilidades, testa a função direta e seu complemento. Claramente, essa abordagem utiliza $O(2^{n+1} . n!)$ passos.

Como todas as abordagens descritas acima incluem o cálculo de equivalência-P, elas também são pertencentes a classe NP-completo.

4.2.3 Assinaturas

Um conceito interessante que pode acelerar a equivalência-P é o de assinatura. Suponha que queremos saber se f_A é equivalente-P a f_B. Ao invés de fazer todas as permutações das variáveis em f_A e verificar se alguma é igual a f_B, podemos primeiramente calcular todas as permutações das variáveis de f_A e escolher uma das funções geradas como o representante da classe. Esse representante da classe poderia ser, por exemplo, a menor função Booleana gerada quando consideramos sua representação em formato hexadecimal. Esse representante é denominado de assinatura, especificamente a menor assinatura.

Desse modo, se fizermos o mesmo com f_B e compararmos as menores assinaturas de ambas, elas somente irão ser iguais caso essas funções Booleanas sejam equivalentes-P. É claro que todas as permutações precisam ser calculadas para f_A e f_B. Contudo depois que isto for feito uma vez, qualquer equivalência-P envolvendo f_A ou f_B pode ser decidida rapidamente utilizando a assinatura da função Booleana.

4.3 Trabalhos Relacionados

Historicamente, o primeiro algoritmo que calculou equivalência-P foi de Mailhot e De Micheli, em 1993 (Benini, 1997). A idéia básica era a de comparar a representação por BDD de ambas funções Booleanas. Logo após isso, diversas novas abordagens surgiram, indo desde formas canônicas até métodos espectrais (Benini, 1997). Uma dessas abordagens foi implementada e testada juntamente com o algoritmo proposto neste trabalho.

4.3.1 Método de Sasao

O artigo de (Debatosh e Sasao, 2004) introduz uma abordagem interessante para resolver equivalência-P utilizando assinaturas. A estratégia é a de pegar uma tabela-verdade e fazer todas as permutações em suas variáveis. A figura 4.5 introduz a idéia com uma tabela-verdade de 2 variáveis.

A	B
0	0
0	1
1	0
1	1

Permutação 0

B	A
0	0
1	0
0	1
1	1

Permutação 1

Figura 4.5: Permutação em uma tabela-verdade

Com todas as permutações, é possível otimizar o armazenamento em memória utilizando a seguinte estratégia. Note que a primeira linha de ambas as tabelas-verdade não se modificaram à medida que as permutações são feitas. Se uma árvore *n-ária* for utilizada para armazenar o conteúdo dessas diversas tabelas, o seguinte resultado será obtido, conforme a figura 4.6.

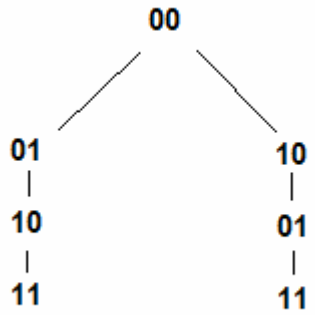

Figura 4.6: Resultado de árvore n-ária para permutações de tabela-verdade.

Esta árvore gerada otimiza o espaço de memória utilizado para representar todas as tabelas-verdade representando as permutações utilizadas. A busca pela menor assinatura

pode ser feita utilizando-se uma busca em largura na árvore trocando o valor dos mintermos pelos seus valores de saída.

A complexidade do método ainda é elevada. Para uma função Booleana com 8 variáveis, o algoritmo levará 8! passos para gerar essa árvore. O tamanho da árvore em memória será 256. 8! no pior caso, o que é proibitivo para elevada quantidade de variáveis.

4.3.2 Método de Heinsberger

A abordagem de Heinsberger (Heinsberger, 1998) utiliza uma ideia semelhante a abordagem de Sasao. Contudo ao invés de gerar uma árvore inteira, critérios de 'prunning' são utilizados para não gerar ramos que não levarão a equivalência. Deste modo, o algoritmo consegue eliminar vários ramos da árvore final gerada.

Mesmo tendo um grande avanço quanto ao espaço a número de variáveis que consegue lidar, o algoritmo ainda está preso a complexidade $O(2^n n!)$ no pior caso.

5 GRAFO BIPARTIDO BOOLEANO

Existem diversas formas de representar uma função Booleana (Benini, 1997). Cada forma de representação evidencia certas propriedades interessantes ao contexto a que são utilizadas.

Tabelas-verdade são interessantes quando o número de variáveis da função Booleana é pequeno, visto que a estrutura de dados cresce exponencialmente juntamente com o crescimento de *n*. Expressões Booleanas são úteis por suas propriedades algébricas e diversas manipulações que sua construção oferece. BDDs (Sasao, 1999) são importantes no contexto de cofatores, pois evidenciam diversas propriedades referentes a este domínio. Porém, uma forma eficiente de representação para funções Booleanas no contexto de equivalência-P seria de grande auxílio a novas investidas nesse contexto. Historicamente, diversas abordagens para equivalência-P utilizam formas de representação não muito flexíveis ao problema (Debatosh e Sasao, 2004).

Este capítulo introduz o conceito de grafo bipartido Booleano, ou GBB, que é uma nova forma de representação de funções Booleanas favorável para o cálculo de equivalência-P. O grande ganho desta estrutura de dados é o de mostrar uma forma simples de visualização de uma função Booleana que permite a detecção imediata de equivalência-P quando duas funções Booleanas têm seus GBB desenhados de forma idêntica.

Com a utilização do GBB foi possível dividir o espaço de funções Booleanas entre funções que permitem equivalência-P em tempo linear e outras que demorariam um tempo exponencial.

5.1 Grafo Bipartido Booleano: Motivação

Suponha que temos uma função Booleana *f* com *n* variáveis e queremos gerar todas as suas funções equivalentes-P. É necessário uma estrutura de dados que facilite a obtenção das *n!* possibilidades de modo que esse processo seja o menos custoso possível.

Duas estruturas de dados candidatas seriam a expressão lógica de *f* ou a sua tabela-verdade. Porém, ambas estruturas necessitam de *n!* passos para gerar todas as funções equivalentes-P de *f*, ou seja, elas não permitem algo eficiente por definição. Além disso, expressões Booleanas não são únicas para representar uma dada função Booleana e uma tabela-verdade não favorece a visualização das permutações envolvidas. É claro que essas estruturas não se adequam ao cálculo de equivalência-P, pois não evidenciam a informação crucial para resolução deste problema: a relação existente entre mintermo e variável.

Uma análise criteriosa do problema permite verificar que um mintermo está intimamente ligado à informação presente em suas variáveis. De fato, um mintermo de uma função Booleana com *n* variáveis é uma tupla composta de 0 e 1 onde cada componente da tupla corresponde a uma variável ou ao seu complemento. É impossível modificar esta relação existente entre mintermo e suas variáveis mediante uma permutação dos componentes do mintermo. Essa informação é importante e mantém-se imutável durante o processo de equivalência-P.

Além do mais, não é possível modificar o número de variáveis de um mintermo mediante uma permutação. Essa é outra propriedade que agrega informação referente a mintermo e variáveis. A união dessas duas propriedades interessantes induzem uma estrutura de dados para computar equivalência-P, estrutura esta que será mostrada nas próximas seções.

5.2 Grafo Bipartido Booleano (GBB)

Considerando uma função Booleana *f* e sua tabela-verdade *T*, um grafo *G = (V, E)* pode ser construído a partir de *T* do seguinte modo:

- O conjunto *V* é formado pela união das variáveis de *f* e de seus mintermos que possuem saída igual a 1. Em *T*, isso corresponde a todas as colunas e todas as linhas em que *F=1*, respectivamente.
- Cada ocorrência da variável *v* no mintermo *m* induz uma aresta *(v, m)* em *E*.

A Figura 5.1 mostra um exemplo de construção do GBB para a função Booleana com todos os mintermos com saída em 1. De forma simplificada, pode-se dizer que cada valor 1 presente em um mintermo com saída 1 induz uma aresta com uma extremidade na variável da coluna e outra no mintermo da linha. Essa representação "enxerga" uma tabela-verdade como uma matriz de adjacência de um grafo não direcionado e se aproveita disso para representar uma função Booleana.

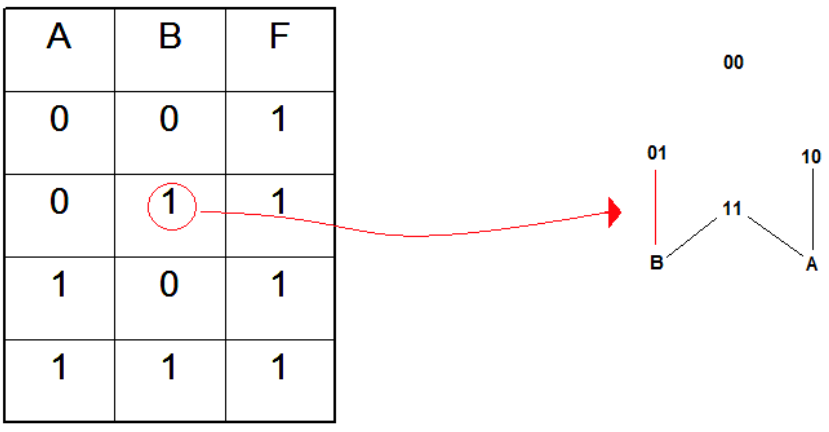

Figura 5.1: Grafo bipartido Booleano.

.

Apesar de representações similares como o grafo de Levi e o grafo fator (factor graph), o grafo apresentado aqui é uma nova forma de representar uma função Booleana denominado *grafo bipartido Booleano,* ou GBB. A propriedade mais interessante do GBB é sua relação com o problema de equivalência-P. Como uma permutação nas

colunas (ou nas linhas) de uma tabela-verdade nunca poderá mudar a relação de adjacência (arestas) criada na definição desta estrutura, qualquer permutação de variável não mudará a forma de representação da função Booleana como GBB. Nesse enfoque, todas as permutações nas variáveis de uma função Booleana não poderão alterar o seu GBB, pois isso não é possível mediante permutações.

Essa propriedade induz a ideia de que duas funções Booleanas quaisquer gerarão GBB iguais a menos de isomorfismo. E aqui não há mais necessidade de se preocupar com *n!* possibilidades de permutação das variáveis envolvidas, mas sim somente na detecção de isomorfismo para este tipo de grafo.

Essa ideia unifica o equivalência-P juntamente com isomorfismo de grafos. Claro que estamos limitados ao contexto de GBB. Contudo essa abordagem permite a visualização de que na verdade esses dois problemas são os mesmos. A conjectura a seguir, base da todo este trabalho, utiliza esta noção:

Conjectura sobre Grafo bipartido Booleano e equivalência-P

Duas funções Booleanas serão P-equivalente se e somente se seus grafos bipartidos Booleanos são isomorfos.

O capítulo 6 discute estratégias para calcular equivalência-P com o GBB. O restante deste capítulo se destina a provar os conceitos utilizados futuramente, bem como enumerar diversas propriedades e perfils do GBB interessantes para resoluções de problemas envolvendo funções Booleanas.

5.3 Grafo Bipartido Booleano: Terminologia

O GBB é um grafo rotulado. Convenciona-se aqui que nodos representando variáveis serão representados por letras maiúsculas e nodos representando mintermos serão representados como números não negativos. A figura 5.2 ilustra alguns exemplos.

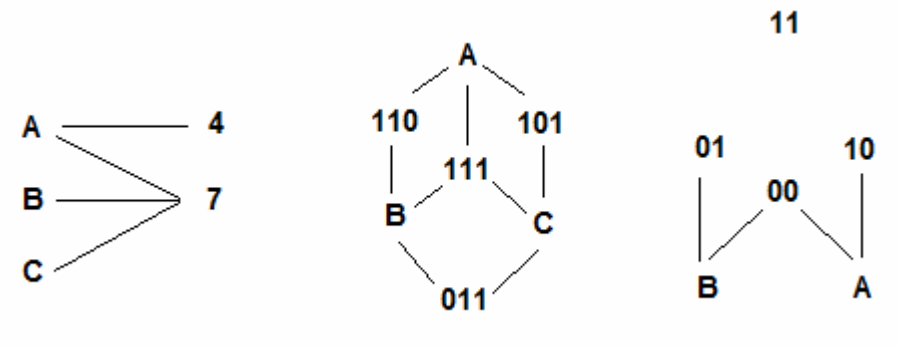

Figura 5.2: Exemplos de GBB.

Um nodo de grau 0 ou 1 será denominado de *nodo de borda*. Este conceito é muito importante para as idéias desenvolvidas no capítulo 6. Nodos de borda possuem a propriedade de nunca estarem dentro de um ciclo.

Um *nodo interno* é um nodo de grau maior ou igual a 1. Nodos internos eventualmente podem estar dentro de um ciclo. A figura 5.3 ilustra esse conceitos em um GBB.

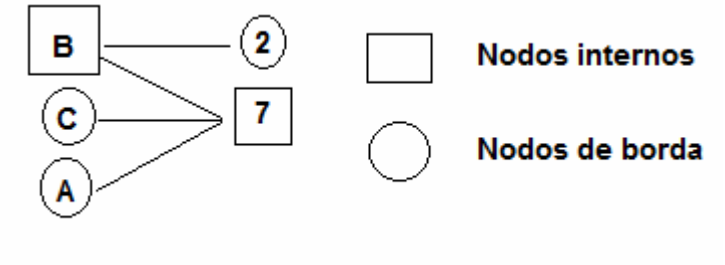

Figura 5.3: Nodos internos e nodos de borda.

5.4 Grafo Bipartido Booleano : Algumas propriedades

A estrutura de dados denominado GBB transforma uma função Booleana em um grafo $G = (V,E)$. A ideia maior dessa conversão é de poder utilizar conhecimentos já provados da teoria dos grafos que possibilitem acelerar de algum modo os problemas enfrentados pela equivalência-P atualmente. Com base nas seguintes propriedades listadas aqui diversos algoritmos podem ser criados.

Teorema GBB-1: O GBB associado a uma função Booleana é único, a menos de isomorfismo.

Prova: Por redução ao absurdo

Suponha que uma função Booleana f com uma tabela-verdade T possui dois GBB associados, G_1 e G_2, de tal modo que G_1 é diferente de G_2 e G_1 e G_2 foram gerados a partir de T.

Como os grafos são diferentes, algum deles tem uma aresta ou nodo que difere do outro. Suponha que seja G_1. Caso G_1 tenha um nodo a mais do que G_2 então pela definição da construção do grafo, a tabela-verdade que gerou G_1 necessita ter um mintermo ou variável a mais do que a tabela-verdade utilizada para gerar G_2. Isso é um absurdo, visto que ambos os grafos vieram da mesma tabela-verdade T, por hipótese.

Caso G_1 tenha uma aresta diferente de G_2 então, pela definição da construção do GBB, a tabela-verdade da qual veio G_1 necessita ter um valor 1 que não aparece na tabela-verdade utilizada para gerar G_2. Isto também é um absurdo visto que ambos os grafos vieram de T.

Q. E. D

O teorema GBB-1 nos deixa seguros quanto a utilização da estrutura de dados para calcular equivalência-P. A unicidade é garantida e, portanto a topologia do grafo de fato é a assinatura da classe de funções Booleanas equivalentes-P.

Propriedade GBB-2: O GBB é bipartido.

Propriedade GBB-3: O mintermo 0 é sempre um nodo desconexo.

Propriedade GBB-4: Em um GBB, nunca existe uma aresta ligando duas variáveis (ou ligando dois mintermos).

Propriedade GBB-5: Para n variáveis, o maior GBB utiliza 2^n+n nodos

5.5 Grafo bipartido Booleano : Exemplos

As propriedades e exemplos listados aqui servem para auxiliar os algoritmos propostos no próximo capítulo e eliminar abordagens impossíveis frente ao perfil do GBB.

Existe um GBB não planar: A figura 5.4 mostra um GBB cujo subgrafo é $K_{3,3}$. Pelo teorema de Kuratowski (Boventura Netto, 2006), esse grafo é não-planar.

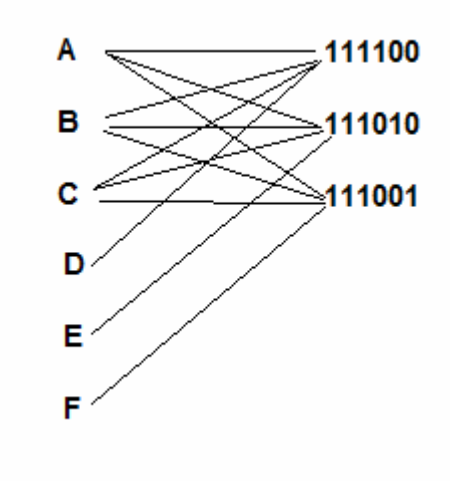

Figura 5.4: $K_{3,3}$.

A não-planaridade do GBB impede o uso de uma abordagem eficiente para cálculo de equivalência-P, pois é conhecido um algoritmo para isomorfismo de grafos em tempo linear quando os grafos são planares (Skiena, 2008).

Existe um GBB com ciclos: A figura 5.5 mostra um GBB com um ciclo.

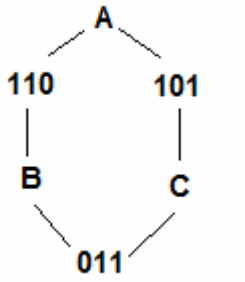

Figura 5.5: GBB com ciclos.

De fato, a porcentagem de GBB com ciclos mediante todos os GBB possíveis não é muito grande, mas eles existem. Como existe ao menos um GBB com um ciclo, não podemos dizer que ele seja uma árvore no caso geral. Assim, isso não permite o uso de algoritmo para isomorfismo de árvores (Skiena, 2008) que também seria linear.

Existe um grafo GBB desconexo: A figura 5.6 mostra um GBB com um ciclo.

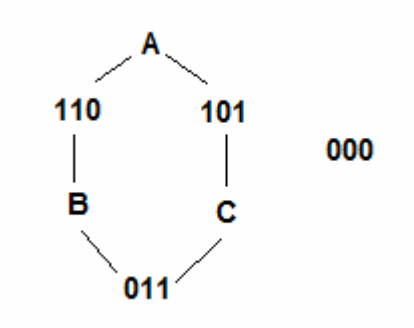

Figura 5.6: GBB desconexo.

Esse resultado evidencia que qualquer algoritmo para equivalência-P utilizando esta estrutura deverá considerar o caso do grafo ser desconexo. Na verdade, o GBB sempre tem o mintermo 0 como um nodo desconexo pois ele não tem nenhum valor 1 para gerar uma aresta.

O GBB não possui ciclos de comprimento impar: Essa é uma propriedade herdada do caso de todo GBB ser um grafo bipartido.

O GBB pode possuir diversos ciclos encadeados: A figura 5.7 mostra um GBB com diversos ciclos encadeados.

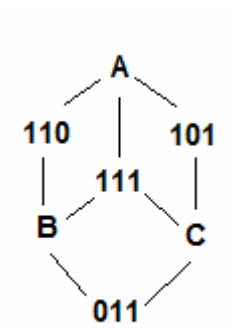

Figura 5.7: GBB com diversos ciclos.

O GBB considera o tipo do nodo quanto a detecção de isomorfimos: Essa propriedade afirma que não basta dois GBB terem a mesma topologia para ser equivalentes-P. Como o grafo é rotulado, além da topologia, o tipo dos nodos (mintermo ou variável) precisam coincidir também. A Figura 5.8 mostra uma exemplo de dois grafos com a mesma topologia, mas não isomorfos pois não possuem os mesmo tipos de nodos.

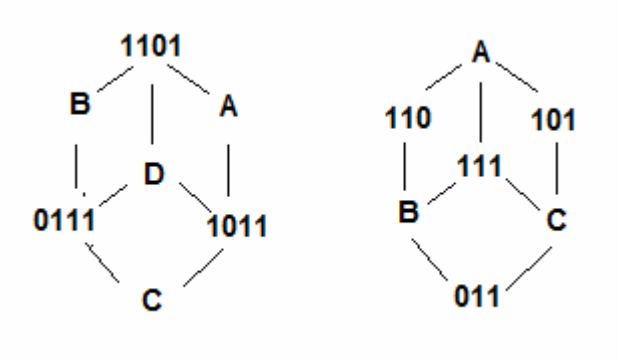

Figura 5.8: Exemplo de GBB com mesma topologia mas não isomorfos no tipo dos nodos

Apesar de mesma topologia, os dois grafos diferem quanto ao número de variáveis e configuração delas. Por exemplo, no GBB da direita, somente existem nodos variáveis de grau 3, o que não ocorre no GBB da esquerda.

5.6 Grafo bipartido Booleano: Dual

O dual de um GBB é o grafo formado pela escolha dos valores em 0 e não dos valores em 1 para formar suas arestas quando temos um mintermo com saída 1.

Considerando uma função Booleana *f* e sua tabela-verdade T, o GBB dual *G = (V, E)* pode ser construído a partir de *T* do seguinte modo:

- O conjunto *V* é formado pela união das variáveis de *f* e de seus mintermos que possuem saída igual a 1. Em *T*, isso corresponde a todas as colunas e todas as linhas em que *F=1*, respectivamente.
- Cada ocorrência da variável complementada !*v* no mintermo *m* induz uma aresta *(v, m)* em *E*.

A figura 5.9 mostra o mesmo exemplo utilizado na elaboração do GBB para elaborar o GBB dual.

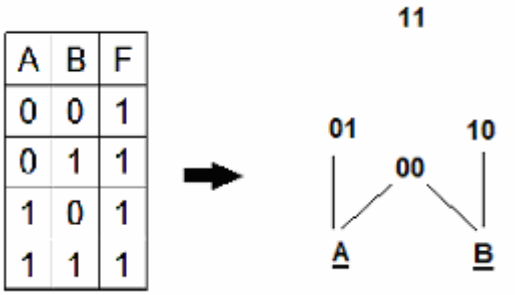

Figura 5.9: GBB dual

Note que por mais que o grafo seja o mesmo nesse caso, em outros os grafos gerados podem ser completamente diferentes. O GBB dual é uma alternativa interessante no caso em que o GBB direto não for suficiente e não chega a ser utilizado em algum algoritmo deste trabalho. Sua apresentação deixa em aberto uma possibilidade interessante para trabalhos futuros.

6 MÉTODOS DE RESOLUÇÃO PARA EQUIVALÊNCIA-P COM GRAFOS BIPARTIDOS BOOLEANOS

Este capítulo explora a estrutura GBB no contexto de cálculo de equivalência-P. Como diversas abordagens são possíveis, aqui é discutido qual delas funcionam e quais não funcionam. Ao final de cada técnica de resolução, é apresentado o aprendizado alcançado com cada implementação possibilitando um entendimento maior das restrições envolvidas nesse problema.

No contexto de isomorfismo de GBB, diversas curiosidades interessantes ocorrem. Isso se deve ao fato de termos um problema que resulta na combinação de dois problemas muito difíceis: equivalência-P e isomorfismo de grafos. Contudo, existem diversas propriedades do GBB que podem ser utilizadas para calcular isomorfismo. Neste enfoque, as estratégias de resolução são apresentadas em ordem de qualidade. Começa-se por abordagens ingênuas culminando em um eficiente algoritmo para calcular equivalência-P utilizando GBB. Note que o algoritmo apresentado em (Debatosh e Sasao, 2004) foi implementado e este é usado como referência para a verificação dos resultados obtidos pelos algoritmos testados.

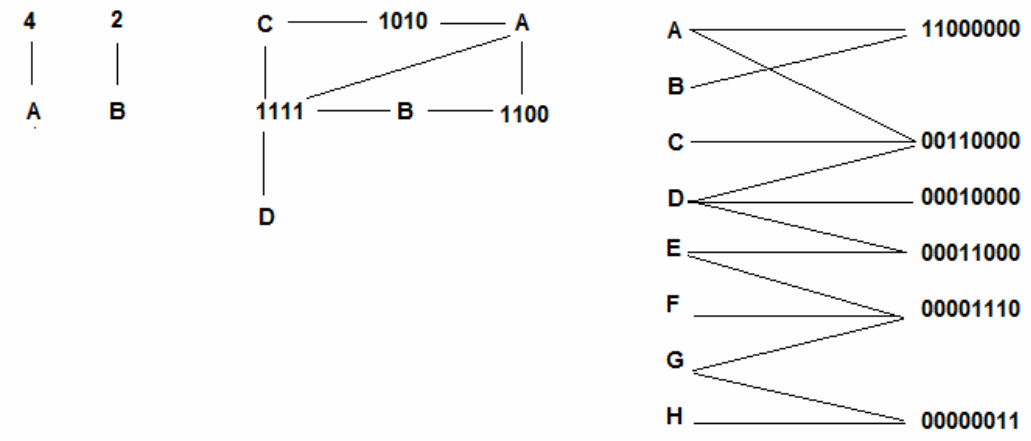

Figura 6.1: Crescimento do GBB quando o número de variáveis cresce.

A figura 6.1 ilustra exemplos de GBB com algumas variáveis (2, 4 e 8). È possível notar que o tamanho do GBB quanto à estrutura de dados cresce exponencialmente. Isso pode parecer ruim à primeira vista, contudo não poderia ser diferente, pois ele representa uma função Booleana. Assim, somente uma abordagem algorítmica, baseada em propriedades dos nodos e arestas pode ser promissora quanto à detecção de

isomorfismo, pois logo o número de nodos tornasse excessivo para a visualização desta propriedade. Os algoritmos nas próximas seções se destinam a resolver tal problema.

6.1 Casos Triviais

Como estamos investigando por isomorfismo de grafos, os mesmos caso triviais apresentados no capitulo 2 valem neste ponto com a devida interpretação nesse novo contexto.

Para dois grafos gerais A e B eles **não** são isomorfos, se:

- o número de nodos que A tem é diferente do número de nodos de B;
- o número de arestas de A é diferente do número de arestas de B;
- se um deles possui um ciclo e outro não;
- se um deles tem um nodo de grau g e o outro não;
- A é conexo e B não;
- A tem arestas paralelas e B não;

Para duas funções Booleanas representadas por seus grafos bipartidos Booleanos A e B, elas **não** são equivalentes-P (ou seja, seus GBB não são isomorfos) se:

- o número de variáveis/mintermos de A é diferente do número de variáveis/mintermos de B;
- o número de 1's na tabela-verdade de A é diferente do número de 1's na tabela-verdade de B;
- se um GBB possui um ciclo e outro não;
- se um GBB tem um nodo de grau g e o outro não;
- A é conexo e B não;
- A tem arestas paralelas e B não;
- A é conexo e B não

Esses casos são ditos triviais, pois nenhum outro algoritmo precisa ser feito caso algum desses critérios não seja satisfeito. Somente funções Booleanas que respeitam esses critérios são utilizadas em abordagens mais sofisticadas para busca de equivalência-P.

6.2 Algoritmo Exaustivo

Para os casos em que todos os critérios triviais são satisfeitos então as duas funções envolvidas podem ou não ser equivalentes-P. Nesses casos, é necessário algum algoritmo para verificar a ocorrência de isomorfismo/equivalência-P quando estes testes não concluem que não há equivalência. O algoritmo mais básico que pode ser imaginado é uma estratégia exaustiva, ou seja, dado f_A uma função Booleana com n

variáveis, basta varrer todas as *n!* possibilidades de permutação em suas variáveis de modo a verificar se alguma delas é P-equivalente a uma f_B destino.

Essa ideia não acrescenta informação alguma ao conhecimento existente sobre equivalência-P, pois um algoritmo exaustivo não contribui para melhorar sua eficiência. Contudo, essa estratégia é interessante do ponto de vista que valida a estrutura de dados GBB quanto a sua corretude se utilizarmos ela nesse algoritmo.

A estratégia utilizada nessa implementação foi a seguinte. Um GBB é um grafo bipartido onde um lado é representado por nodos mintermo e o outro por nodos variáveis. Como a topologia do grafo é a assinatura da classe, ou seja, não se altera mediante qualquer permutação, podemos manter a topologia do GBB intacta e permutar os nodos variáveis de alguma forma. A figura 6.2 mostra essa ideia em funcionamento.

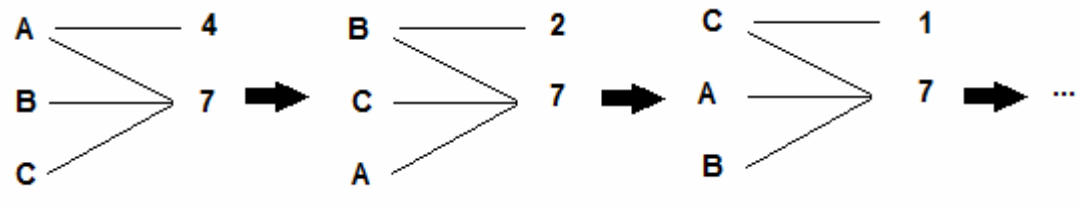

Figura 6.2: Algoritmo exaustivo.

A permutação de variáveis num GBB gera novos mintermos que podem ou não ser os mesmos que os procurados para implementar uma função destino (no caso, f_B). Caso isso ocorra, então as duas funções serão equivalentes-P após uma sequência de permutações.

Esta implementação foi testada e comparada com a implementação da abordagem apresentada (Debatosh e Sasao, 2004) através do teste de todas os pares de funções Booleanas possíveis até 4 variáveis. Nenhum erro foi encontrado.

Em suma:

Algoritmo Exaustivo:

Complexidade: O(n!),

Contribuição: Provar que para até 4 variáveis, a conjectura sobre GBB e equivalência-P está correta.

6.3 Algoritmo Backtracking

A abordagem exaustiva é muito ingênua para calcular equivalência-P, pois o GBB possui diversas propriedades que podem ser exploradas para "podar o espaço de busca". Deste modo, uma estratégia *backtracking* foi utilizada (Skiena, 2008).

A ideia principal é a seguinte. É possível representar o GBB como uma grande *string*. A figura 6.3 mostra como isso é possível.

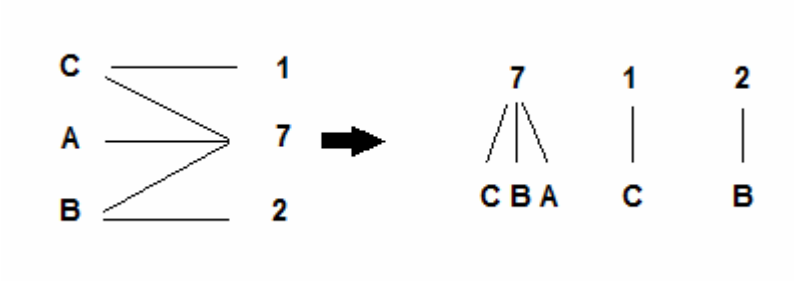

Figura 6.3: Transformação GBB em string.

Supondo duas funções f_A e f_B, basta gerar a *string* para f_A e começar a substituir as variáveis de modo que não seja gerado algum mintermo que não esteja presente em f_B. Por exemplo, temos CBA, C e B na figura 6.3. Se trocarmos C por A e vice-versa, obtemos o seguinte resultado ilustrado na figura 6.4:

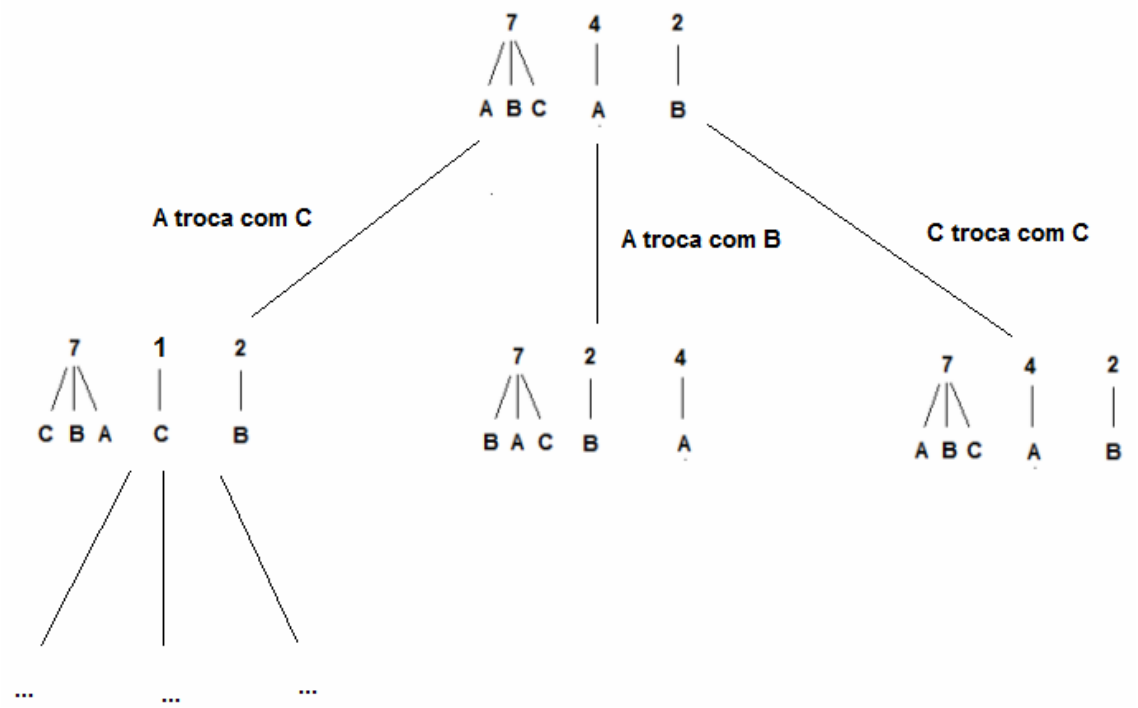

Figura 6.4: Simples troca de variável.

Claro que C poderia ser trocado por B ou ficar intacto e que não faz sentido fazer a troca B por A se já foi feita A por B. Desse modo, dado uma letra na string de f_A, é possível tentar todas as possibilidades de trocas para C, logo depois tentamos as possibilidades restantes as letras que sobraram. Isso induz uma busca em profundidade que freqüentemente é denominada de backtracking.

Na Figura 6.4, note que o mintermo 4 foi gerado e os outros permaneceram intactos. Se o mintermo 4 faz parte de f_B então essa permutação é aceita e o algorimo continua mapeando as variáveis que faltam. Caso o mintermo 4 não seja um mintermo presente em f_B então esse mapeamento é descartado e outro é tentado nesta mesma posição. É fácil perceber que esta abordagem, no pior caso, tentará $n!$ possibilidades. Porém, com sorte, as diversas restrições quanto à formação de mintermos acabam por "podar" tentativas que não levarão à resposta.

Esse algoritmo melhorou os tempo observado no caso exaustivo consideravelmente. Nenhum erro foi observado, mas ainda não foi possível fugir das diversas possibilidades inerentes ao problema, existindo casos em que houve *n!* tentativas.

Em suma:

Complexidade: *O(n!)*, no pior caso.

Contribuição: Acelerar o algoritmo exaustivo anterior deixando diversos testes mais factíveis.

6.4 Algoritmo Guloso

Nesse caso estratégias algorítmicas mais avançadas são necessárias. O maior problema enfrentado pela abordagem em *backtracking* é o fato de não existir um critério que permita quantificar o quão longe da solução ótima está a atual solução. A escolha por qual variável começar a substituição é feita de forma arbitrária e, portanto, seria necessária uma estratégia mais inteligente.

A abordagem apresentada em (Debatosh e Sasao, 2004) introduziu a ideia de assinatura, ou seja, escolher uma função Booleana dentro de uma classe de permutações que consiga representar toda a classe. Neste trabalho, a abordagem escolhida foi procurar pela menor função Booleana possível de se gerar através de permutação quando consideramos sua representação por hexa.

Aqui a mesma ideia é utilizada: investigar como utilizar o GBB para buscar a menor assinatura e esperar que algum modo eficiente seja possível. A figura 6.5 ilustra a ideia por meio de um exemplo.

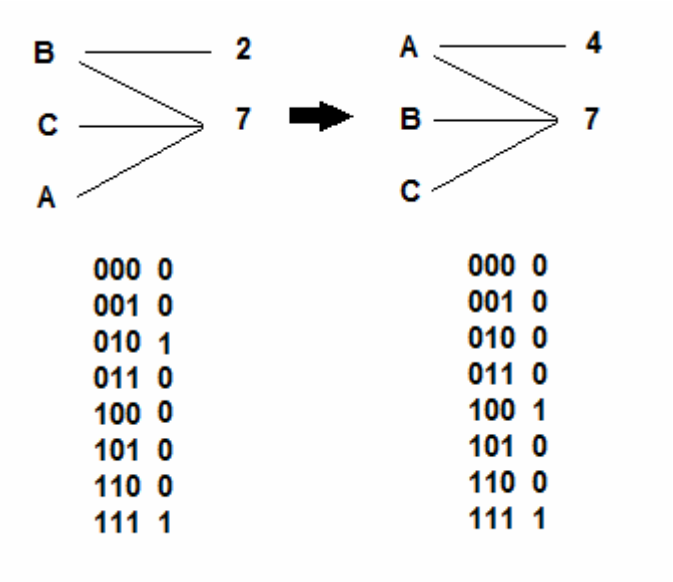

Figura 6.5: Exemplo de permutação utilizada no algoritmo guloso.

Pela figura 6.5 é possível notar que a permutação utilizada diminuiu a representação por hexa da tabela-verdade de saída (4ª coluna nas tabelas-verdade abaixo dos grafos) e, portanto, essa permutação aproxima um pouco mais a solução atual do menor representativo global. Essa simples ideia induz o seguinte algoritmo.

Dado uma função f_A de entrada, basta fazer todas as combinações de pares de variáveis para permutar. Isso gerará $n(n-1)/2$ possibilidades onde ao menos uma delas deve diminuir a representação em hexa de f_A fazendo com que a solução atual fique menor. Ao chegar numa solução com representação em hexa menor, aceita esta e repete o processo até não ter mais como diminuir este hexa. A etapa gulosa deste algoritmo se concentra em escolher a menor representação em hexa gerada dentre essas combinações locais.

Essa abordagem não utiliza $n!$ passos para executar, mas, de fato, não funciona. Não é possível construir uma estratégia gulosa para resolver equivalência-P visando diminuir ou aumentar a representação em hexa, pois muitas vezes é necessário aumentar a representação em hexa para num próximo passo, diminuir novamente. Esse comportamento oscilante obriga a estratégia gulosa a considerar uma solução pior do que a melhor solução encontrada até então, o que vai contra a ideia geral de otimização inicial desta proposta.

Em suma:

Estratégias gulosas não funcionam para calcular equivalência-P utilizando GBB. Essa percepção é importante pois será utilizada em algoritmo futuros.

6.5 Algoritmo de Divisão e Conquista

A ideia central de divisão e conquista é conseguir dividir um problema de modo que a solução de seus subproblemas possam ser combinadas para gerar a solução do problema inicial. Infelizmente, isso não funciona com o GBB e, portanto, para a equivalência-P. Vejamos isso através de um exemplo.

A figura 6.6 abaixo ilustra dois GBB que **não** são isomorfos.

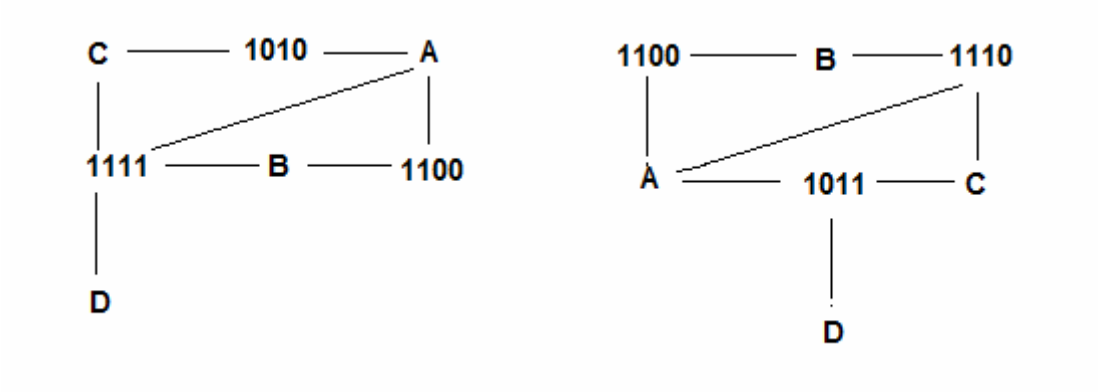

Figura 6.6: Remoção de um nodo deixa os grafos isomorfos.

A eliminação do nodo "D" em ambos os grafos os deixa aparentemente isomorfos e portanto equivalentes-P, o que não condiz com a situação inicial com a presença de "D". Em geral, não é possível dividir a busca por equivalência-P em subproblemas sem precisar levar a informação de estado do problema anterior em consideração. Abordagens que apagam arestas indiscriminadamente ou que tentam dividir o grafo ou assemelhado, não funcionarão pelo mesmo argumento.

Em suma:

Também é um algoritmo que não funciona, mas com ele aprende-se que não é possível subdividir o problema em subproblemas independentes por remoção de nodos sem manter algum tipo de informação. Em uma remoção é necessária a propagação de informação representando topologia que existia antes da remoção.

6.6 Algoritmo de Construção

Após a discussão utilizando algumas técnicas algorítmicas clássicas vamos verificar e discutir, agora algoritmos baseados em outros conceitos. A propriedade fundamental do GBB é a sua topologia. A topologia do GBB identifica a forma que qualquer função Booleana equivalente-P deve satisfazer, ou seja, qualquer função equivalente-P precisa ter seus mintermos e variáveis inseridos dentro da topologia do GBB ao qual se deseja verificar equivalência.

Essa idéia simples induz um algoritmo construtivo, onde o objetivo é o de ter uma topologia base, digamos a topologia de uma das funções envolvidas na equivalência-P e, logo após, tentar encaixar os nodos da função Booleana restante dentro dessa forma.

Como mostra a figura 6.7, a Booleana f_A pode oferecer sua topologia se esquecermos os nomes dos nodos envolvidos. Deste modo, podemos considerar somente suas arestas e nodos fictícios que serão preenchidos pelos nodos de f_B.

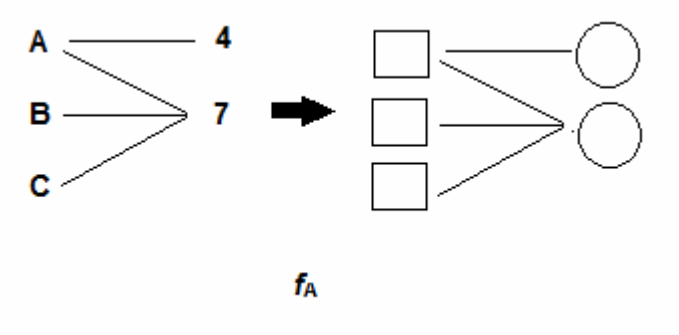

Figura 6.7: Representação da topologia de uma função Booleana

A segunda etapa é a de inserir os nodos de f_B na topologia de f_A. A figura 6.8 ilustra esse processo.

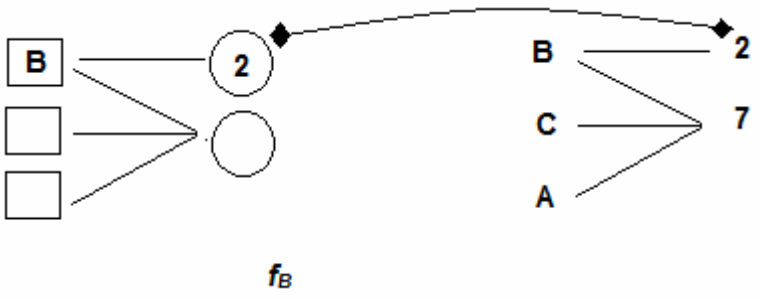

Figura 6.8: Inserção de nodos em uma topologia.

A inserção de nodos de f_B dentro da topologia dos nodos presentes na topologia de f_A é uma tarefa difícil. De fato, esse algoritmo é ainda mais difícil que a própria equivalência-P em si. Como não há critério algum para escolha do nodo inicial,

poderíamos começar pelo nodo 7. Nesse caso, não sabemos se esta posição para o nodo 7 é a posição correta, e então teremos que lidar com o fato desta tentativa estar errada. Se estiver errada, teremos então que tentar outro nodo e assim por diante. Logo, inicialmente testamos k possibilidades onde k é o número de nodos do grafo, e pode ser exponencial. Ainda considerando o caso do nodo 7 vemos que ele possui três vizinhos do tipo **variável**. Também não é possível saber como esses vizinhos devem se arranjar, logo devemos tentar todas as possibilidades, ou seja, 3! tentativas. Como possuímos, no pior caso, 2^n mintermos, e estes possuem n variáveis, o pior caso desse algoritmo testa 2^n. $n!$, o que é muito pior do que $n!$ utilizado pela equivalência-P pura. Logo, essa abordagem é uma ideia impraticável.

6.7 Algoritmo de Redução

Este é o melhor algoritmo para resolver equivalência-P utilizando o GBB. A ideia de construir o grafo do zero não é interessante pelos motivos explicitados acima. Em contrapartida, a ideia de destruir/reduzir o GBB não foi explorada. O algoritmo que utiliza este técnica dentro do contexto de GBB é denominado **algoritmo de redução**.

O algoritmo de redução se baseia no princípio de que todo nodo de um grafo pode ser classificado como "pertencente a um ciclo" ou "não pertencente a um ciclo". Utilizando a terminologia definida anteriormente, estes nodos são nodos de borda e nodos internos, respectivamente. Assim, dado um grafo qualquer, é possível comprimir seus nodos de modo a gerar uma grafo menor. O primeiro critério de redução que aparece é o de reduzir nodos de grau 0 e 1, ou seja, seus nodos de borda. A figura 6.9 ilustra um exemplo didático.

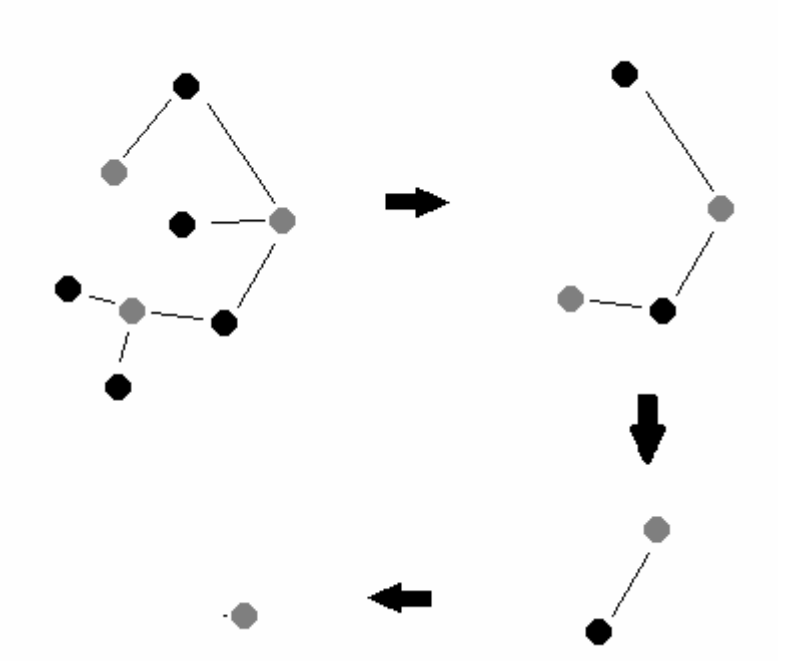

Figura 6.9: Nodos internos e nodos de borda.

Note que a cada passo os nodos de borda são eliminados. Foi percebido com o algoritmo de construção que é necessário ter informação do problema inicial para conseguir gerar um subproblema válido. Logo, utilizando o que foi aprendido com aquela abordagem propagamos aqui uma informação (que ainda será definida) a cada

eliminação de nodo, ou seja, um nodo ao ser eliminado manda para seu nodo adjacente uma informação especifica. Isso é ilustrado na figura 6.10.

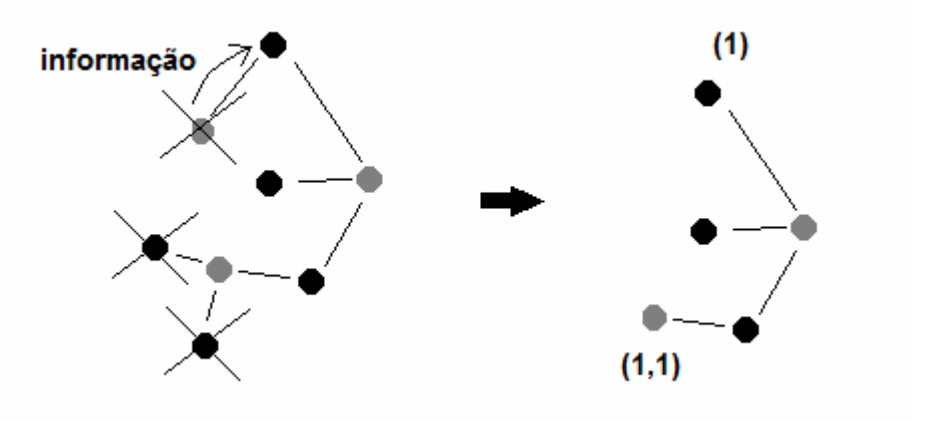

Figura 6.10: Propagação de informação mediante remoção de um nodo.

Deste modo, um nodo ao ser eliminado, ainda influencia no resultado do próximo passo com sua informação propagada. Essa mesma rotina pode ser repetida até não haver mais nodos de borda, e nesse momento temos duas possibilidades a tratar: o grafo está vazio, e nesse caso conseguimos um algoritmo linear para equivalência-P, ou existem nodos internos.

Quando só restam nodos internos, a abordagem precisa ser estendida. A figura 6.11 mostra um caso ilustrativo.

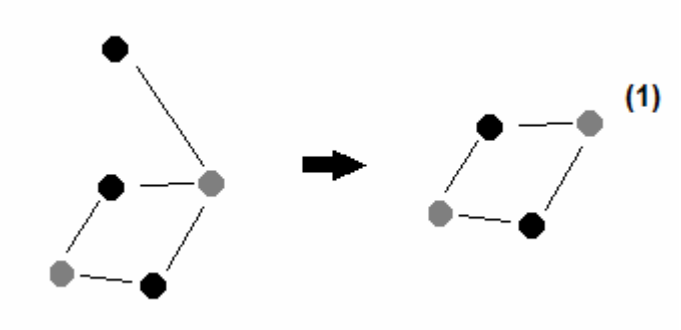

Figura 6.11: Remoção resulta em nodos internos.

Quando não temos mais nodos de borda, somente sobram nodos internos e, portanto, temos a certeza de que ao menos um ciclo existe no grafo. Nesse caso, é necessário quebrar o ciclo de algum modo. Diversas estratégias são possíveis. Contudo, duas se destacam:

1) é possível buscar um nodo com propriedades únicas e removê-lo, ou seja, que tenha uma informação propagada única no GBB ou então tenha grau único.;

2) se isso não existir, será necessário considerar todas as possibilidades de ponto de quebra, mas nesse caso o problema volta explodir o número de combinações possíveis;

A estratégia para melhorar essa perspectiva vem da própria definição de equivalência-P. Na proposta do problema temos duas funções Booleanas, f_A e f_B, e, pela conjectura base deste trabalho, se elas foram equivalentes-P seus grafos serão isomorfos. Logo a sequência de remoção de nodos e propagação de informação deve ser

idêntica em ambos os GBB caso essa eliminações de nodos sejam feitas simultaneamente em ambos. A cada passo, um conjunto de nodos é removido e é testada a equivalência desses nodos (mesmo tipo, grau, informação salva...). Caso esteja tudo certo, mais uma rodada de eliminação é feita.

De fato essa mudança conserta os problemas enfrentados quanto a nodos internos.

1) se existir algum nodo único no GBB de f_A então deve existir um nodo com as mesmas características em f_B, basta remover ambos;

2) se esse nodos não existirem, sabemos que um dado nodo variável v_1 no GBB de f_A terá que ser mapeado a um nodo variável v_2 presente no GBB de f_B, assim, para v_1 temos n possibilidades, onde n é o número de variáveis de ambas as funções Booleanas.

Inicialmente, todo nodo de grau 1 recebe como código a seguinte palavra "( )". Cada remoção de nodo é seguida de uma propagação de seu código. Nodos de grau 0 não propagam código pois não possuem nodos vizinhos. Nodo de grau 1 encapsula seu código dentro de um parênteses e o propaga para seus vizinhos desse modo: "(código do nodo)". Nodo de grau maior do que 1 salvam seu código em uma estrutura de dados e propaga o índice de seu código salvo nesse estrutura de dados. A fig 6.12 ilustra um exemplo dessa dinâmica.

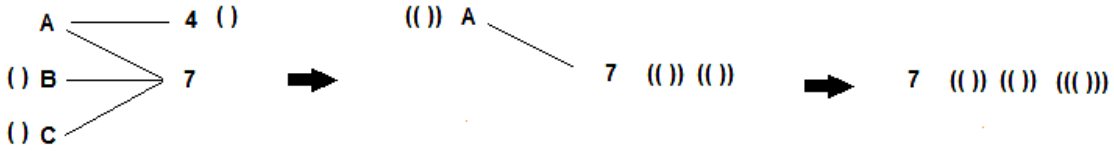

Figura 6.12: Dinâmica remoção e propagação de código.

Quando há um nodo interno, então o processo é o ilustrado na figura 6.13.

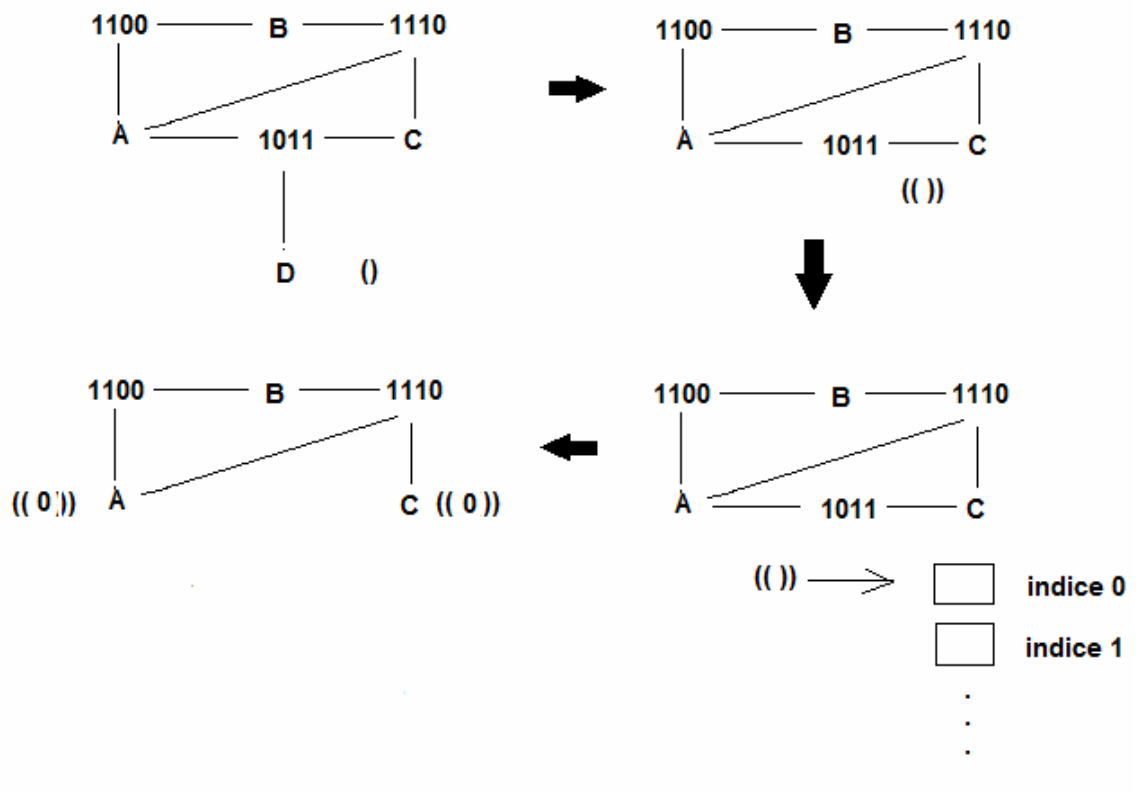

Figura 6.13: Remoção quando há nodo de grau maior do que 2.

Após isso o processo continua reduzindo os nodos até os grafos se tornarem vazios.

Note que cada remoção de nodo obrigatoriamente propaga um código para o nodo vizinho. Aqui uma discussão mais precisa é necessária. Por que os códigos são representados por parênteses? Pois desse modo existe a noção de encapsulamento. Um dado nodo com código *x,* este propagará o código (*x*). Desse modo é possível desempacotar esses códigos sabendo que cada parênteses que abre é um novo nodo.

No caso de nodos de grau 2 ou mais é necessário salvar seu código em uma estrutura de dados. Por que? Isto diferencia o caso em que um nodo N_1 de grau *n* propaga seu código *c* para os seus *n* vizinhos do caso em que *k* nodos de borda propagam o mesmo código *c* para os mesmos *n* nodos vizinhos. Com a adição de uma estrutura de dados, nodos de grau 2 salvam seu código na estrutura de dados e propagam este índice. Assim há uma diferenciação desses dois casos.

O algoritmo completo é o que segue:

Algoritmo de Redução (G_1, G_2)

Se (vazio(G_1) && vazio(G_2))
    retorna equivalente;
//REMOVE NODOS DE GRAU ZERO
remove_nodos_grau_zero(G_1)
remove_nodos_grau_zero(G_2)
Se (numero_de_nodos_removidos(G_1) != numero_de_nodos_removidos(G_2))
    retorna não equivalente;
Se (numero_de_nodos_removidos(G_1) != 0)
    retorna Algoritmo de Redução (G_1, G_2)
//REMOVE NODOS DE GRAU UM
remove_nodos_grau_um(G_1)
remove_nodos_grau_um(G_2)
Se (numero_de_nodos_removidos(G_1) != numero_de_nodos_removidos(G_2))
    retorna não equivalente;
Se (numero_de_nodos_removidos(G_1) != 0)
    retorna Algoritmo de Redução (G_1, G_2)
//REMOVE NODO UNICO
extrai_nodo_único(G_1)
extrai_nodo_único(G_2)
Se (numero_de_nodos_removidos(G_1) != numero_de_nodos_removidos(G_2))
    retorna não equivalente;
Se (numero_de_nodos_removidos(G_1) != 0)
    retorna Algoritmo de Redução (G_1, G_2)
//REMOVE NODOS QUEBRANDO O CICLO
n = escolhe_variável(G_1)
para toda variável v em G_2
    remove_nodo(n, G_1)
    remove_nodo(v, G_2)
    Se (Algoritmo de Redução (G_1, G_2))
        retorna equivalente
    Senão insere_nodo(n, G_1)
        insere_nodo(v, G_2)

A primeira impressão que temos desse algoritmo é que ele não explode em um número elevado de combinações. Sua execução não é como o *backtracking*, pois caso não haja equivalência possível para v_1 então as funções Booleanas não serão equivalentes-P. Logo, o algoritmo pára e não tenta combinar outro nodo variável.

6.7.1 Análise do Algoritmo de Redução

O algoritmo de redução foi implementado e testado nas mesmas condições que os algoritmos anteriores. O teste feito foi a busca por equivalência-P entre todo o par de funções com até 4 variáveis. Em todos os testes ele conseguiu êxito e seus tempos de execução são os apresentados na Tabela 2.1.

Tabela 2.1- Geração da P-class com *n* variáveis

Num. Variáveis (n)	Alg. de Redução
0	5 ms
1	6 ms
2	26 ms
3	181 ms
4	401691 ms

O número de funções Booleanas que tem ciclos e, portanto, precisaram da etapa adicional não linear foi conforme apresentado na Tabela 3.1.

Tabela 3.1 – Ocorrência de ciclos sobre todas as funções possíveis com *n* variáveis

Numero de variáveis (n)	Numero de ciclos / nº total de funções
0	0/0
1	0/2
2	0/4
3	44/256
4	9.376/65536
5	42.936/616125

A complexidade do algoritmo de redução é linear no número de nodos do GBB (que pode ser exponencial) caso não haja ciclos, e exponencial caso haja ciclos. Logo, é possível concluir que existe um algoritmo eficiente para equivalência-P que depende somente da ocorrência de ciclos presentes em sua representação por GBB.

6.7.2 Discussão sobre os resultados

A maior contribuição do algoritmo de redução é a de dividir o espaço de funções Booleanas em dois pedaços: um com equivalência-P em tempo linear e outro com

equivalência-P não-linear. Poucas funções se enquadram nesse quesito e, portanto, sua ocorrência deve ser estudada em trabalhos futuros.

O algoritmo de redução também não muda a classificação de equivalência-P quanto a NP-completo. Por mais que seja dado um algoritmo de tempo linear no número de nodos, a quantidade de nodos é exponencial, logo o problema ainda continua NP-completo. O que acontecia anteriormente é que diversos algoritmos levavam tempo exponencial para executar sobre uma entrada exponencial (mintermos de uma tabela-verdade). O que é proposto neste trabalho é um algoritmo eficiente que opera sobre uma entrada exponencial, ou seja, a complexidade ainda continua exponencial, mas não por causa do algoritmo proposto e sim pela natureza das funções Booleanas e seu domínio de tamanho exponencial. Um GBB de uma função Booleana com 10^6 variáveis terá $2^{10\text{^}6}$ mintermos, e se todos eles tiverem saída 1, o GBB gerado terá toda essa infinidade de nodos. Contudo, se a função Booleana de 10^6 tiver somente 10 mintermos com saída em 1 então sua equivalência-P poderá ser feita de modo muito rápido.

A título de compreensão sobre o tamanho do espaço utilizado nas geração dos GBB presentes nesses estudo, o número de funções presente em cada P-class é ilustrado nos histogramas abaixo.

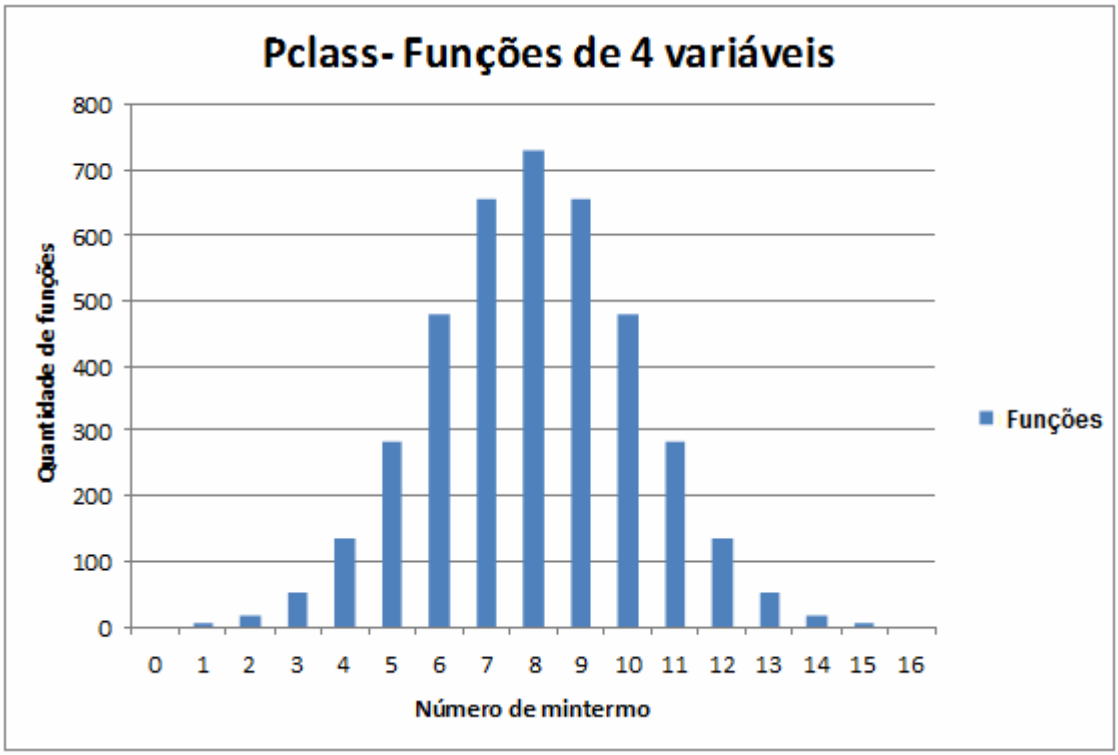

Figura 6.14: Números de funções diferentes gerada por permutação para cada função de 4 variáveis.

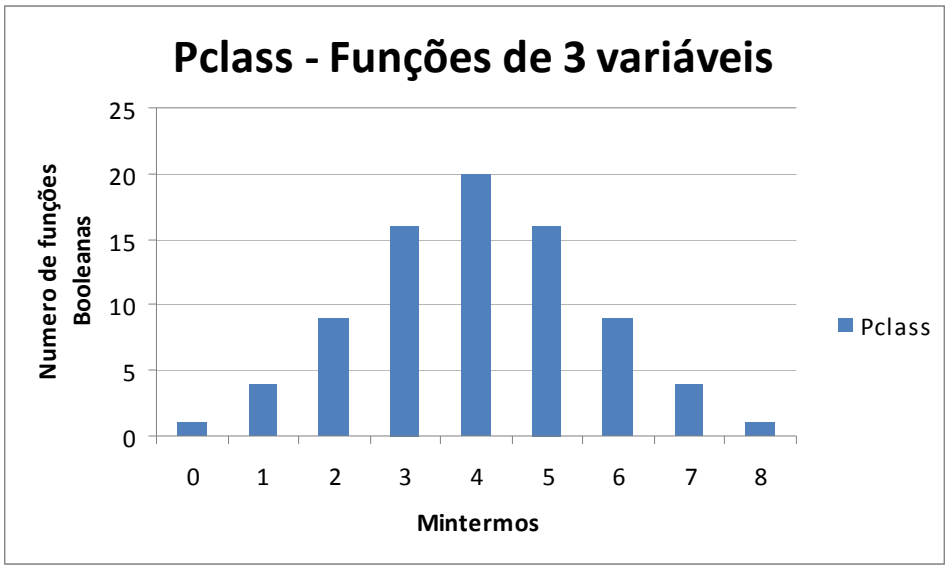

Figura 6.15: Números de funções diferentes gerada por permutação para cada função de 3 variáveis.

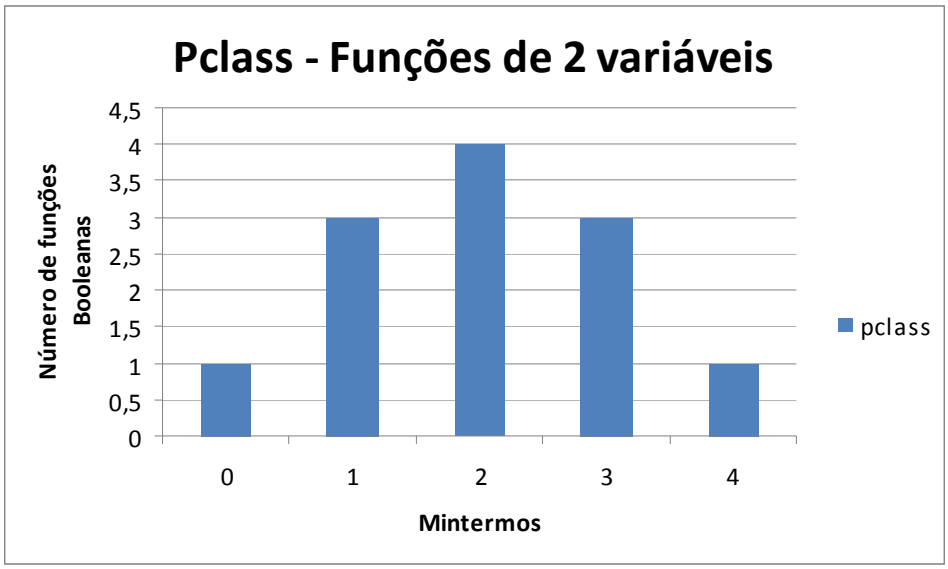

Figura 6.16: Números de funções diferentes gerada por permutação para cada função de 2 variáveis.

Observando as figuras 6.14, 6.15 e 6.16 é possível notar que a distribuição de funções Booleanas lembra uma curva de distribuição Gaussiana e, portanto, as funções no centro do gráfico (próximas a 2^{n-1}) são as que mais informação carregam.

6.7.3 Resultados Adicionais

A representação de funções Booleanas como grafos bipartidos ainda introduz mais uma contribuição interessante para a teoria dos grafos. De fato, o problema de isomorfismo de grafos bipartidos é NP-completo no caso geral.

É comum na área de Complexidade de Algoritmos utilizar o conceito de Redução (Skiena, 2008). Redução é uma operação que transforma um problema em outro, ou seja, é um modo de mostrar que na verdade um problema é tão difícil quanto o outro. Um algoritmo rápido para resolver um dos problemas implica que o algoritmo também seja rápido para o outro problema.

Este artifício é muito importante visto que também serve para demonstrar que um algoritmo é tão difícil quanto outro. Imagine que saibamos que o algoritmo A é muito difícil. Se conseguirmos demonstrar que o algoritmo B representa o mesmo problema que A então o algoritmo B é tão difícil quanto A.

Os conceitos introduzidos nesse trabalho permitem a seguinte hipótese:

"Isomorfismo de grafos bipartidos provavelmente seja NP-completo"

O problema de equivalência-P é NP-Completo. Se conseguirmos demonstrar de algum modo que equivalência-P é na verdade o mesmo problema que isomorfismo de grafos bipartidos, então a redução estará pronta.

De fato, o GBB representa a redução de um problema de equivalência-P para o um subconjunto do problema de isomorfismo de grafos bipartidos. Neste enfoque, os dois problemas são idênticos e, portanto, isomorfismo de grafos bipartidos provavelmente seja NP-completo, pois no caso geral existe ao menos um grafo bipartido que irá levar um número exponencial de passos para ter isomorfismo detectado.

7 CONCLUSÃO

Este trabalho utiliza conceitos provenientes da Ciência da Computação, Eletrônica e Matemática para propor um novo algoritmo para equivalência entre funções Booleanas. Funções Booleanas são representadas como grafos bipartidos e esta representação permite a definição de um novo algoritmo para computar equivalência-P que pode ser utilizado no fluxo de projeto de circuitos integrados digitais.

A primeira contribuição é a proposta de uma novo forma de representação para funções Booleanas que apresenta algumas propriedades interessantes e promissoras. O nome desta estrutura é grafo bipartido Booleano, ou GBB.

A segunda contribuição é em um novo algoritmo para cálculo de equivalência-P. O presente trabalho analisou diversas abordagens para resolver equivalência-P utilizando o GBB, e oferece um algoritmo que define qual o critério para uma função Booleana ter equivalência-P em tempo linear. Logo, este trabalho divide o espaço de funções Booleanas em dois grupos: funções Booleanas que possuem equivalência-P em tempo linear e que não possuem.

A terceira contribuição é uma possível demonstração que isomorfismo de grafos bipartidos é um problema NP-completo, pois o GBB representa uma redução de um problema NP-completo a um problema que ainda não estava classificado. Logo, é possível que esta abordagem tenha potencial para provar que isomorfismo de grafos bipartidos é NP-Completo.

O GBB possui diversas outras aplicações que fogem do escopo deste trabalho como, por exemplo, cálculo de cofatores de modo muito eficiente, detecção de simetria entre variáveis de uma função Booleana, entre outros. De fato, um estudo maior sobre o GBB ainda poderá ser feito. Em particular, o GBB dual pode ser explorado e a definição de uma assinatura para a topologia do grafo pode ser desenvolvida em trabalhos futuros estendendo os resultados propostos aqui.

REFERÊNCIAS

Sasao, Tsutomo. **Switching Theory for Logic Synthesis**. Massachusetts: [s.n.], 1999. .

Gersting, Judith L. **Fundamentos matemáticos para ciência da computação**. 4rd ed. Rio de Janeiro: LTC-Editora, 2001.

Morais Filho, Daniel Cordeiro de. **Um convite à matemática**. Campina Grande: EDUFCG, 2007.

Skiena, Steven S. **The algorithm design manual**. 2nd.Ed. Londres: Springer, 2008.

Boaventura Netto, Paulo Oswaldo. **Grafos: teoria, modelos, algoritmos.** 4rd ed. São Paulo: Edgard Blucher, 2006.

De Micheli, Giovanni. **Synthesis and Optimization of Digital Circuits**. 1994

Menezes, Paulo Blauth. Matemática **Discreta para Computação e Informática**. 2 ed. Porto Alegre: Sagra Luzatto,2005

Rochol, Juergen. **Comunicação de Dados**. Porto Alegre: Ed. Bookman, 2012.

Eugene M. Luks. **Isomorphism of Graphs of Bounded Valence Can be teste in polynomial Time**. Journal of Computer and System Science,1981.

Heinsberger, Uwe. Kolla, Reiner. **Boolean Matching for large libraries**, 1998.

Debatosh, Debnath. Sasao, Tsutomo. **Efficient Computation of Canonical Form for Boolean Matching in Large Libraries**. Asia and South Pacific Design Automation Conference,2004

Benini, Luca. De Micheli, Giovanni. **A Survey of Boolean Matching Techniques for Library Binding**.ACM Transactions on Design Automation of Electronic Systems, 1997.