Geração de Redes de Transistores Otimizadas Utilizando uma Abordagem Baseada em Grafos

Julio S. Domingues Júnior, Vinicius N. Possani, Renato S. de Souza, Felipe de S. Marques, Leomar S. da Rosa Jr.

Grupo de Arquiteturas e Circuitos Integrados – GACI
Universidade Federal de Pelotas (UFPel) – Pelotas, RS – Brasil

{jsdomingues,vnpossani,rsdsouza,felipem,leomarjr}@inf.ufpel.edu.br

Abstract. *This paper presents an alternative technique to generate optimized logic cells at transistor level using a graph-based approach. Two algorithms are presented in order to deliver optimized transistor networks and reduced Boolean expressions that represent the networks. The first uses an edges sharing strategy in a graph structure, while the other exploits edges compression. Results demonstrate that both algorithms can deliver good results when comparing to traditional approaches that are capable to deliver only series-parallel transistor arrangements.*

Resumo. *Este trabalho tem o objetivo de apresentar uma solução alternativa para geração de células lógicas otimizadas no nível de transistores através de uma abordagem baseada em grafos. Neste sentido, dois algoritmos distintos baseados em grafos foram desenvolvidos, um para otimizar redes de transistores utilizando compartilhamento de arestas e outro para extrair expressões Booleanas que representam os grafos otimizados através da técnica de compactação de arestas. Resultados demonstram a viabilidade de utilização dos algoritmos, os quais são capazes de apresentar ganhos quando comparados às abordagens clássicas para geração de redes de transistores.*

1. Introdução

O avanço da tecnologia atual para o desenvolvimento de circuitos integrados tem permitido que cada vez mais transistores sejam integrados em um único chip. Entretanto, o projeto dos circuitos integrados deve ser realizado com a utilização da menor área de silício possível, de modo que o custo em área do circuito não se torne proibitivo. Essa restrição aumenta a importância da fase de projeto na concepção de circuitos VLSI (*Very Large Scale Integration*), o que leva os projetistas à necessidade de utilizar ferramentas que facilitem o trabalho de projetar os circuitos, assim como diminuir o tempo de projeto. Em particular, ferramentas voltadas para síntese lógica e algoritmos especialmente desenvolvidos para este fim têm contribuído consideravelmente para facilitar a execução de projetos.

Em projetos customizados no nível de rede de transistores, cada célula lógica de um bloco funcional é concebida manualmente, o que demanda muito tempo. Neste sentido, torna-se necessário ter algoritmos eficientes para geração automática de células otimizadas. Exemplo disto são as ferramentas de CAD (*Computer Aided Design*) para circuitos integrados que trabalham com otimização em nível de síntese lógica, aplicando algoritmos de fatoração no intuito de derivar redes lógicas otimizadas [Brayton 1987],

[Mintz 2005], [Yoshida 2006], [Golumbic 2008]. A fatoração manipula uma expressão Booleana recebida como entrada, a fim de reduzir o número de literais que compõem essa expressão. Posteriormente, essa expressão otimizada é mapeada para uma rede de transistores com número reduzido de chaves.

Existem na literatura métodos alternativos, como por exemplo, a otimização baseada na utilização de grafos. Nessa abordagem a rede de transistores é mapeada para uma estrutura de grafo, onde cada aresta possui associação direta com um transistor na rede. A idéia principal é tentar minimizar a quantidade de arestas [Kohavi 1970], [Dietmeyer 1971], [Wu 1985], [Zhu 1993] ou até mesmo criar um novo grafo com menor número de arestas [Kagaris 2007]. Porém, essas manipulações no grafo devem garantir que o comportamento lógico do circuito inicial não seja alterado. Com a aplicação dessas técnicas é possível obter um circuito com mesma função lógica, entretanto, com menor número de transistores, resultando em menor área, menor consumo de energia e maior eficiência.

Neste contexto, este trabalho apresenta um método para geração de redes de transistores otimizadas através de uma abordagem baseada em grafos e a extração da expressão Booleana que representa essa rede otimizada. A abordagem proposta recebe como entrada uma Soma de Produtos (SOP) e a transforma em um grafo, que, posteriormente, é minimizado através do compartilhamento de arestas, resultando em um grafo que é mapeado diretamente para uma rede de transistores otimizada. Então, através de um algoritmo que aplica compactação de arestas do grafo, a expressão Booleana que representa a rede otimizada é obtida. Além disso, o algoritmo é capaz de reconhecer o tipo de arranjo de transistores que compõem o circuito após a otimização, identificando arranjos conhecidos como redes *Wheatstone Bridge*. Esse tipo de arranjo, quando pode ser atingido, consegue representar redes com um número reduzido de transistores, quando comparado ao equivalente na forma de arranjo série-paralelo. Através da integração dos algoritmos desenvolvidos, surgiu a ferramenta chamada de *Soptimizer* [Possani 2011].

O restante deste artigo está organizado da seguinte forma: A seção 2 apresenta os conceitos básicos necessários para o entendimento deste trabalho. A seção 3 apresenta o método proposto para otimizar as redes de transistores. Na seção 4 é apresentado o algoritmo de geração de expressões Booleanas a partir do grafo otimizado. A seção 5 discute os resultados experimentais. Finalmente, as conclusões são apresentadas na Seção 6.

2. Conceitos Preliminares

Uma rede de transistores pode ser representada por um grafo. Um grafo é um par ordenado $G = (V, E)$ que compreende um conjunto V de vértices ou nós, juntamente com um conjunto E de arestas ou linhas, que são dois elementos do subconjunto de V. Um grafo dirigido é um par ordenado $D = (V, A)$ com V vértices, e A um conjunto de pares ordenados de vértices, chamados arcos, arestas dirigidas, ou flechas. Um arco $a = (x, y)$ é considerado ser dirigido a partir de x para y, então y é chamado de cabeça e x é chamado de cauda do arco; y é dito ser um sucessor direto de x, e x é dito ser um antecessor direto de y. Se um caminho conduz a partir de x para y, então y é dito ser um sucessor de x e acessível a partir de x, e x é dito ser um predecessor de y.

Diferentes tipos de estruturas de dados podem ser usadas para armazenar grafos em um sistema de computador. A matriz de adjacência é uma delas. Essa é uma matriz A *n* x *n*, onde *n* é o número de vértices no grafo. Se existe uma aresta de um vértice *x* para um vértice *y*, então o elemento *A(x,y)=1*, caso contrário é *0*. Em computação, esta matriz torna fácil a tarefa de encontrar subgrafos dentro de um grafo maior.

Além de representação da função Booleana, grafos também podem ser usados para representar redes de transistores. Neste caso, cada aresta representa um transistor e os vértices são os pontos de conexões (nodos) entre os transistores. Normalmente, as redes de transistores *CMOS* são construídas através de associações série-paralelo. Dessa forma, cada literal da expressão Booleana torna-se uma aresta (transistor) na representação do grafo. Portanto, se menor o número de literais em uma expressão, menor os transistores necessários para implementar essa função Booleana.

Uma função Booleana é uma função da forma $f: B^n \rightarrow B$, onde $B = \{0,1\}$ é um domínio Booleano e n é um inteiro não-negativo. No caso em que $n = 0$, a função é simplesmente um elemento constante do B. Mas geralmente, uma função Booleana de valores é uma função do tipo $f: X \rightarrow B$, onde X é um conjunto arbitrário e onde B é um domínio Booleano. Se $X = [n] = \{1, 2, 3, ..., n\}$, então f é uma sequência binária de comprimento n. Portanto, existem mais de 2^{2^n} funções.

A representação de uma função Booleana é feita por um conjunto de variáveis que podem alterar o valor de saída de uma função. Por exemplo, considere a função *f (a, b, c) = a.b +!a.c.* O seu conjunto de variáveis é o conjunto *{a, b, c}*. Um vetor de entrada é um elemento definido no domínio Booleano e indica o valor de cada variável que define o espaço Booleano. Um vetor de entrada $v \in B^n$ pertence ao conjunto de ON-set se, e somente se, *f(v) = 1*. Caso contrário, se *f (v) = 0*, então *v* pertence ao OFF-set de *f*. Embora os vetores *v1 = {1,1,0}* e *v2 = {0,1,1}* pertencem ao ON-set definido da função *f (a, b, c) = a.b +! a.c*, o vetor *v3 = {1,0,0}* vai para o OFF-set de *f (a, b, c)*.

Uma expressão lógica é uma representação da função Booleana. Cada função é única para qualquer aplicação $f: B^n \rightarrow B$ em todo o espaço Booleano. No entanto, uma função Booleana tem representações infinitas. Todas as funções Booleanas podem ser expressas em uma forma canônica através de soma de produtos (SOP) ou produto de somas (POS).

A SOP é dita canônica quando todas as variáveis aparecem em todos os produtos. Cada instância de uma variável Booleana é chamada de literal. Um produto de literais é formalmente chamado de cubo. Por exemplo, *{a, b, c}* é um cubo interpretado como *a.b.c* ou apenas como *abc*. Um mintermo é um cubo que contém todas as variáveis que representam a função. Mapas de *Karnaugh* [Karnaugh 1953] e o método de *Quine-McCluskey*, segundo [Quine 1955] e [McCluskey 1956], são as principais técnicas de otimização exaustiva para dois níveis de minimização. Embora eles não sejam algoritmos práticos para circuitos de grande porte, eles são fáceis de usar e simples de entender. O algoritmo *Expresso* [Mcgeer 1993] é um método heurístico para dois níveis de minimização que é computacionalmente menos custoso e apresenta bons resultados. Esses métodos têm como objetivo a redução do número de literais, eliminando redundâncias em uma SOP.

Geralmente, o número de literais de uma expressão SOP-irredundante (ISOP), não é mínimo. Neste sentido, poderíamos utilizar a forma fatorada para representar funções Booleanas. De acordo com [Brayton 1987], uma forma fatorada pode ser definida como uma representação de uma função lógica que tenha um único literal ou uma soma de produtos em forma fatorada. É muito semelhante a uma expressão algébrica com parênteses. Essa representação com parênteses parece ser a representação mais adequada para uso na síntese lógica multinível.

Existem vários métodos de fatoração [Brayton 1987], [Mintz 2005], [Yoshida 2006], [Golumbic 2008] para a obtenção de diferentes formas fatoradas de uma função lógica. Esses métodos variam desde os puramente algébricos, que são bastante rápidos, aos chamados métodos Booleanos, que são mais lentos, mas são capazes de obter melhores resultados. Uma vez que a obtenção de uma forma fatorada mínima para uma função Booleana arbitrária é um problema NP-difícil [Mintz 2005], todos os algoritmos práticos para fatoração são heurísticos e fornecem uma solução, correta e logicamente equivalente, mas não necessariamente uma solução ótima. As formas fatoradas são especialmente interessantes para gerar redes de transistores, que posteriormente irão compor as portas lógicas ou células lógicas. Normalmente, portas lógicas são criadas seguindo algum estilo de topologia. Os estilos mais comuns são a *Pass-Transistor Logic* (PTL) e a *Complementary Séries-Paralell* (CSP) ou *CMOS* (também conhecida como *CMOS* estática). Independentemente da topologia, a saída da célula é ligada a VDD ou GND através de um ou mais caminhos de transistores interconectados.

Um caminho de VDD para a saída da célula é chamado de *Pull-up*, enquanto um caminho de GND para a saída da célula é chamado de *Pull-down*. Existem várias técnicas para uma geração automatizada de redes de transistores. Alguns métodos para geração de redes PTL são encontrados em [Buch 1997], [Hsiao 2000], [Shelar 2001], [Shelar 2002], [Avci 2003], [Da Rosa Jr. 2006] e [Da Rosa Jr. 2007]. A maioria deles é baseado em grafos, tais como diagramas de decisão binária (BDDs). Neste caso, cada nó de um BDD é um ponto de decisão que corresponde a um multiplexador de duas entradas.

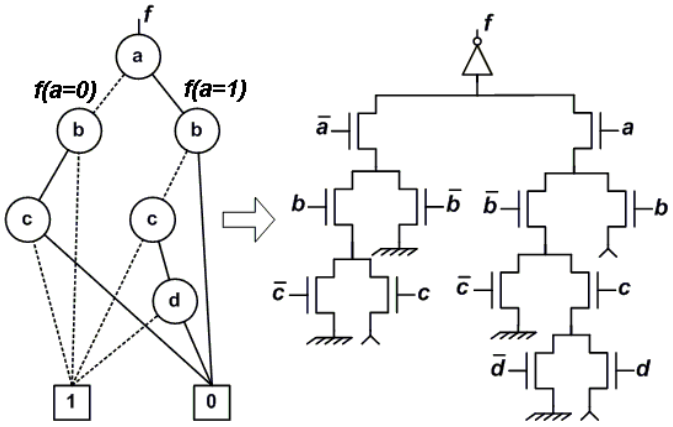


Figura 1. Rede de transistores PTL derivada de um BDD.

Considere a função *f (a, b, c, d)* tal que o ON-set e o OFF-set são representados pelas Equações 1 e 2, respectivamente.

$$\text{ON-set } (f) = !a.!b+!a.!c+!b.c+!b.!d \quad (1)$$

$$\text{OFF-set } (f) = a.b+b.c+a.c.d \quad (2)$$

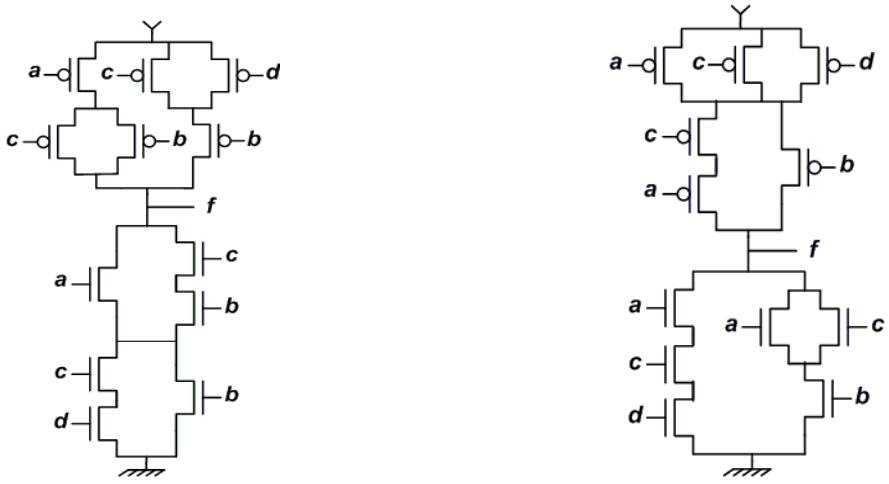
A Figura 1 mostra um BDD que representa o ON-set da função *f*, e sua rede de transistor derivada considerando a topologia PTL.

Na topologia CSP os planos complementares das células são implementados usando planos disjuntos de transistores série-paralelo. O plano *Pull-up* (*Pull-down*) corresponde ao conjunto de transistores interconectados entre a saída da célula e VDD (GND). Enquanto o plano *Pull-up* é composto apenas por transistores *PMOS*, o plano de *Pull-down* é composto por transistores *NMOS*. Quando o plano *Pull-up* é derivado da equação ON-set de uma função Booleana, a topologia do plano *Pull-down* é o complemento em série-paralelo do plano *Pull-up*. De maneira semelhante, as células CSP também podem ser derivadas a partir da equação de OFF-set. Neste caso, o plano *Pull-down* é derivado diretamente da equação e o *Pull-up* é o complemento em série-paralelo do plano *Pull-down*. Considere as formas fatoradas expressas pelas Equações 3 e 4:

$$\text{ON-set } (f) = !a.(!b+!c)+!b.(!c+!d) \quad (3)$$

$$\text{OFF-set } (f) = b.(a+c)+a.c.d \quad (4)$$

A Figura 2 mostra células CSP derivadas do conjunto ON-set e OFF-set das Equações 3 e 4, respectivamente. Observe que cada literal destas equações corresponde a um par de transistores na porta lógica. Portanto, o menor número de literais na forma fatorada, implica em um menor número de transistores na célula lógica.



a) Célula derivada do ON-set*(f)* b) Célula derivada do OFF-set*(f)*

Figura 2. Rede de transistores CSP CMOS de *(f).*

Ao invés de usar uma forma fatorada para derivar uma rede de transistores, existem técnicas, tais como [Kohavi 1970], [Dietmeyer 1971], [Wu 1985], [Zhu 1993], [Kagaris 2007] e [Da Rosa Jr. 2009], as quais aplicam um conjunto de regras e operações para eliminar arestas de um grafo. Cada aresta do grafo corresponde a um transistor. Portanto, o compartilhamento das arestas e operações aplicadas no grafo

resulta num grafo mais otimizado para implementar portas lógicas. A principal contribuição destas técnicas é que elas são capazes de atingir redes de transistores com arranjos *bridge*, também chamados de *Wheatstone Bridge*. Esse tipo de associação, quando atingida, leva a uma contagem de transistores menor se comparado à implementação série-paralelo.

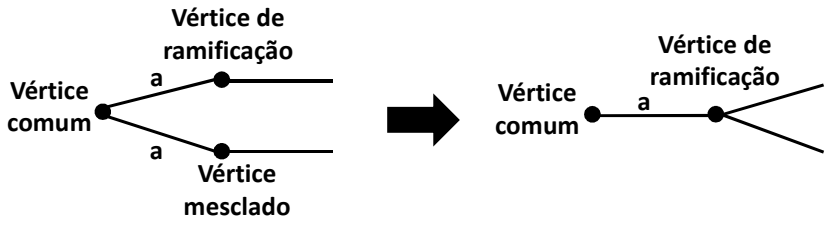
3. Algoritmo Proposto para Otimização de Redes de Transistores

O grafo inicial representa uma SOP, tanto para o conjunto ON-set ou OFF-set de uma função Booleana. Cada produto de uma SOP representa um único caminho entre dois nós terminais (vértices) que são rotulados como *T0* e *T1*. A abordagem proposta realiza otimizações em duas etapas. Primeiramente, o método de compartilhamento de arestas é aplicado atravessando o grafo a partir do nó terminal *T0* para o nó terminal *T1*. Em uma segunda etapa, o processo de otimização é aplicado na direção oposta, a partir do terminal *T1* para o terminal de *T0*. Usando esta estratégia, é possível alcançar associações de transistores do tipo *Wheatstone Bridge.*

Alguns métodos, tais como [Kohavi 1970], [Dietmeyer 1971], [Wu 1985] e [Zhu 1993], tentam compartilhar as arestas de uma forma gulosa, sem prestar atenção para a melhor seleção de arestas que serão unidas em cada iteração. Outros métodos tentam construir o grafo através de um processo gradual de inserção de arestas [Kagaris 2007]. Esses métodos também podem atingir arranjos *Wheatstone Bridge*. No entanto, temos observado empiricamente que é oportuno para minimizar o grafo, aplicar compartilhamento série-paralelo antes de tentar alcançar associações *Wheatstone Bridge*. Esta é a principal diferença entre a abordagem proposta e os métodos disponíveis na literatura.

3.1 Compartilhamento de Arestas

Como primeiro passo do método de compartilhamento de arestas, todos os caminhos no grafo são percorridos a fim de reconhecer arestas idênticas (arestas que são representadas pelo mesmo literal). Quando essa condição é verificada, as arestas idênticas são movidas para a extremidade do grafo onde posteriormente serão compartilhadas. Esse processo de reposicionamento de arestas é realizado com o objetivo de fazer com que as arestas candidatas a otimização possuam pelo menos um vértice em comum. Assim, cada iteração de compartilhamento é bem definida, mantendo um vértice em comum entre as arestas candidatas, outro vértice que será mesclado e um vértice que posteriormente será o ponto de ramificação entre os caminhos. A Figura 3 ilustra os vértices e arestas envolvidos durante um processo de compartilhamento.



a) b)

Figura 3. Vértices e arestas envolvidos durante o processo de compartilhamento de arestas.

Considere a Equação 5, que representa a função lógica *XOR* de quatro entradas. O grafo inicial para esta função é ilustrado na Figura 4.a, onde também é possível ver as arestas que fazem conexões entre os pares de vértices.

$$f = !a.!b.!c.d + !a.!b.c.!d + !a.b.!c.!d + !a.b.c.d + a.!b.!c.!d + a.!b.c.d + a.b.!c.d + a.b.c.!d \quad (5)$$

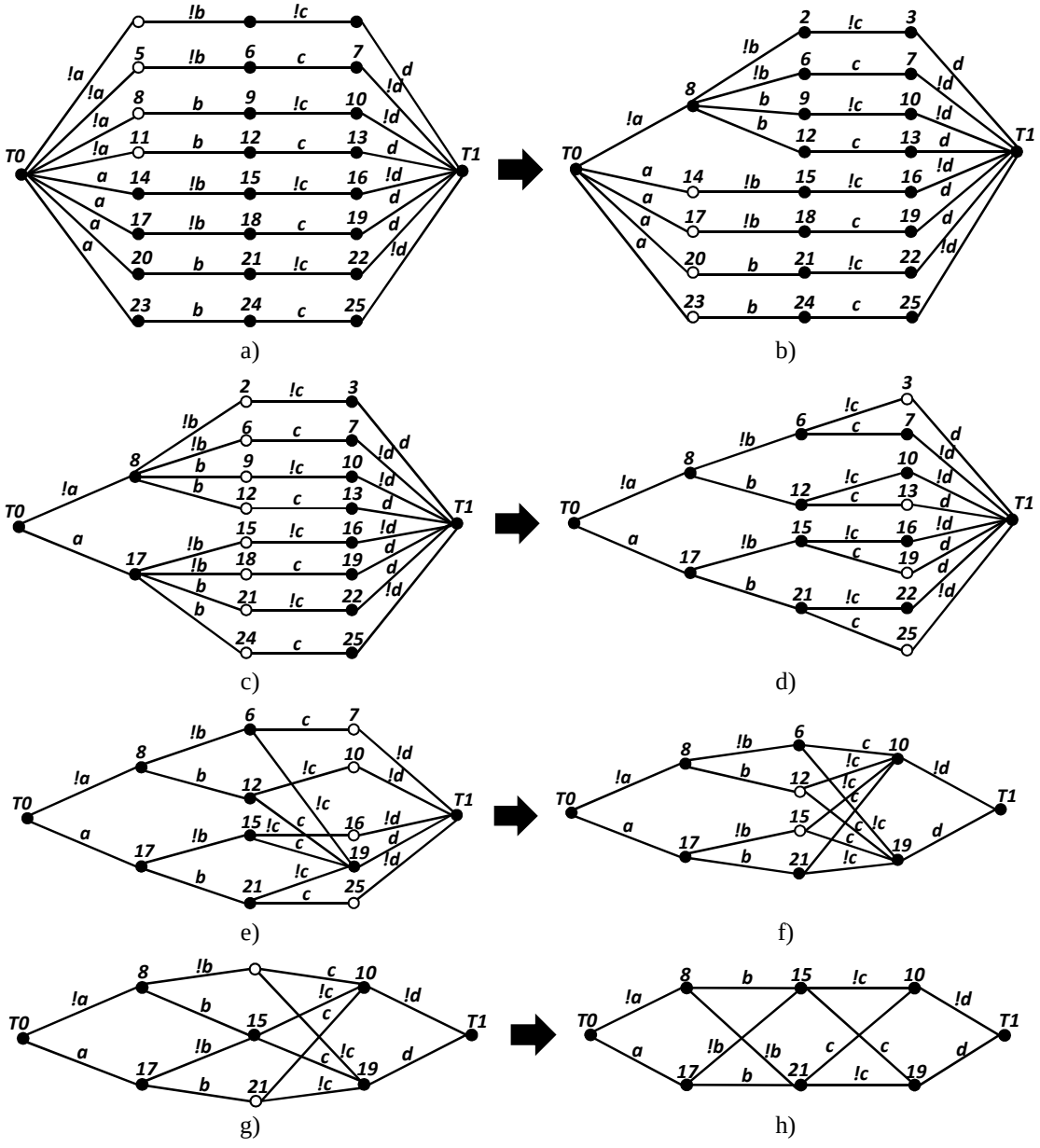


Figura 4. Compartilhamento de arestas aplicado a função lógica *XOR* de 4 entradas.

O compartilhamento de arestas leva a diminuição na contagem de arestas como ilustra a Figura 4.b. Nas figuras abaixo, os vértices que serão mesclados são destacados por círculos, enquanto os vértices sem alteração são representados por pontos pretos. Por exemplo, na Figura 4.b, a aresta *!a* foi compartilhada e os vértices 1, 5, 8 e 11 foram mesclados em um único vértice. Na sequência, a aresta *a* é compartilhada, o que resulta no grafo mostrado na Figura 4.c. Posteriormente, os vértices 8 e 17, um de cada vez, são

considerados o novo ponto de partida do processo de otimização, onde o algoritmo procura arestas idênticas entre esses dois vértices e o terminal *T1*. Desta forma, as arestas *!b* ligadas ao vértice 8 serão compartilhadas e, na sequência, isso ocorre com as arestas *b*. Posteriormente, o mesmo processo é aplicado para as arestas *!b* e *b* ligados ao vértice 17. Isto é demonstrado na Figura 4.d.

Como pode ser visto na Figura 4.d, partindo dos vértices 6, 12, 15 e 21, um de cada vez, e percorrendo o grafo em direção ao terminal *T1*, não é possível realizar novas otimizações, já que arestas idênticas não são encontradas entre estes vértices. Neste momento, a segunda etapa de otimização é iniciada. O grafo é percorrido partindo do terminal *T1* em direção ao terminal *T0*. Assim as arestas *d* são identificadas e são compartilhadas, como demonstrado na Figura 4.e. Na seqüência, as arestas *!d* serão compartilhadas, resultando no grafo da Figura 4.f.

Agora, considere os vértices 10 e 19 como novos pontos de partida para o método de compartilhamento. Há duas arestas *!c* conectadas no vértice 10, como mostrado na Figura 4.f. Então é possível remover a aresta *!c* conectada aos vértices 10 e 12 unindo os vértices 12 e 15. Neste caso, a união dos vértices 12 e 15 resulta em duas arestas *c* conectadas entre os vértices 19 e 15. Quando essa condição é detectada, apenas uma dessas arestas permanece no grafo, como ilustra a Figura 4.g. Este processo é aplicado novamente, mas desta vez para as arestas *c* que estão conectadas ao vértice 10, unindo os vértices 6 e 21. Esse processo deriva duas arestas *!c* conectadas entre os vértices 19 e 21, uma destas arestas será removida resultando no grafo final apresentado na Figura 4.h. Como pode ser visto na Figura 4.h, não é possível realizar outras otimizações. Assim, o processo de otimização é concluído e o grafo final é entregue. Caso alguma das operações de otimização gere um caminho inválido, uma rotina de recuperação é invocada e o compartilhamento inválido é revertido. Esta rotina é apresentada na próxima subseção.

Outro exemplo de execução do algoritmo proposto é ilustrado pela Figura 5 e pela Figura 6. Considerando a Equação (6) como entrada do algoritmo, após algumas iterações de compartilhamento, o método é capaz de atingir a rede mínima ilustrada pela Figura 6.c. A rede derivada pelo método proposto implementa um arranjo *Wheatstone Bridge* com o mesmo número de chaves da rede obtida pelo método descrito em [Zhu, 1993], utilizando a mesma expressão Booleana como entrada.

$$f = a.b.!e.f + a.b.h + a.c.e.g + a.!b.c.e + b.c.d.!e.f + b.c.d.h + d.e.g + !b.d.e \qquad (6)$$

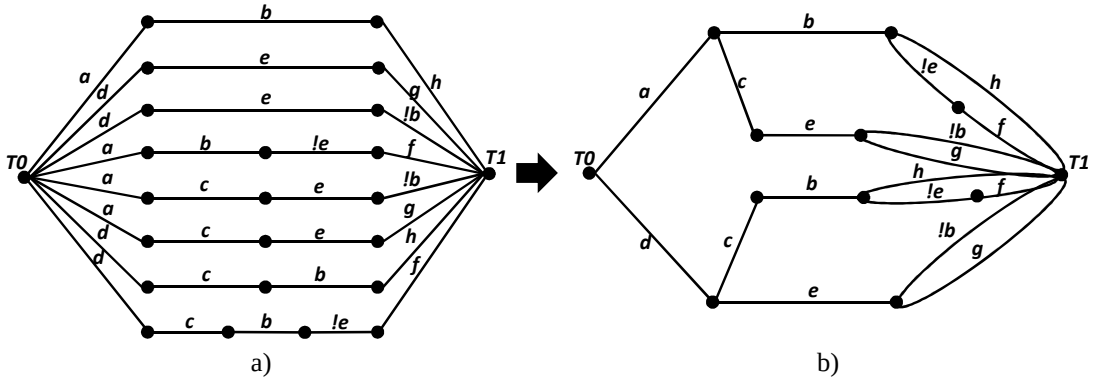


Figura 5. (a) grafo obtido a partir da Equação (6) e (b) grafo resultante após algumas iterações.

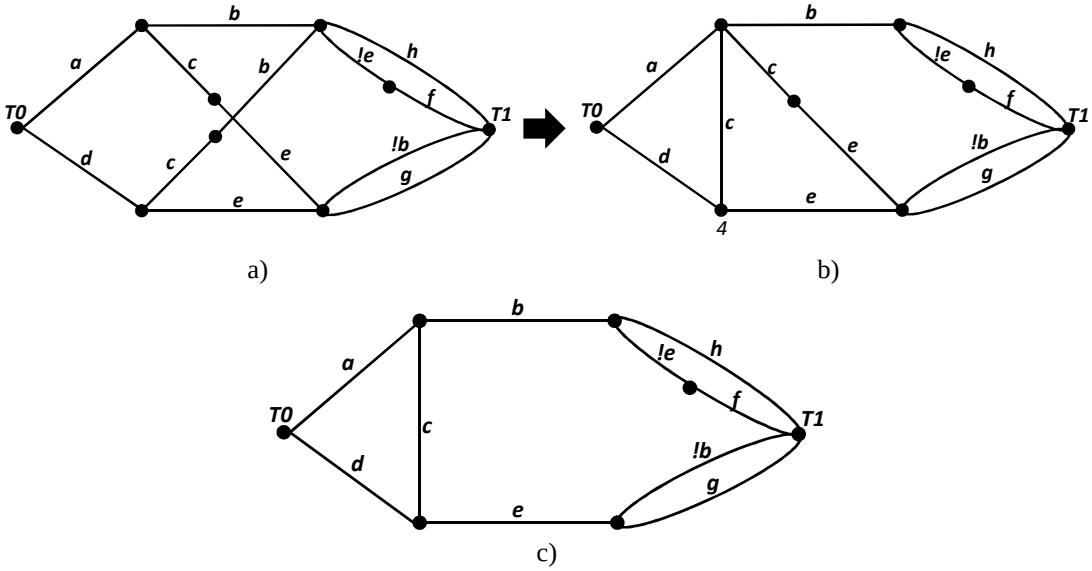


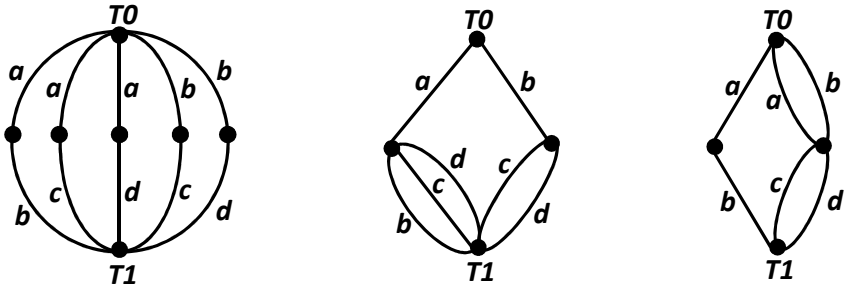
Figura 6. Passos intermediários (a) e (b) do processo de otimização para Equação (6) e solução mínima encontrada em (c).

Como pode ser visto, para o exemplo da função *XOR* e para o exemplo da Equação (6), o método proposto realizou as melhores escolhas de arestas candidatas aos compartilhamentos, resultando na implementação mínima para cada exemplo. Porém, dependendo da SOP de entrada, podem existir diversas possibilidades de escolha de arestas candidatas. Diferentes escolhas podem resultar em diferentes soluções. Assim, o processo para selecionar os candidatos que levarão à solução ideal é uma tarefa difícil. Portanto, a fim de fazer a melhor escolha, algumas heurísticas devem ser aplicadas.

O método proposto implementa uma estratégia gulosa para selecionar arestas. Esta estratégia consiste em selecionar os candidatos que irão resultar na maior redução de arestas. Na maioria dos casos, esta estratégia leva a bons resultados. Em alguns casos, diferentes escolhas por arestas candidatas resultarão no mesmo número de arestas eliminadas do grafo. Então, a estratégia gulosa não consegue distinguir quais são os melhores candidatos. Pois, no compartilhamento atual o resultado será o mesmo independente da escolha. Porém, uma escolha ruim nesse momento pode afetar compartilhamentos futuros. Assim, esta abordagem é fortemente dependente da ordem das arestas. Este problema de decisão é demonstrado a seguir.

Considere a Equação (7), que é representada pelo grafo ilustrado na Figura 7.a. Aplicando a escolha gulosa, será selecionada a aresta candidata *a*. O processo de otimização converge para a solução apresentada na Figura 7.b. No entanto, realizando uma outra escolha, o processo de otimização pode alcançar a solução ideal, conforme apresenta a Figura 7.c.

$$f = a.b + a.c + a.d + b.c + b.d \quad (7)$$

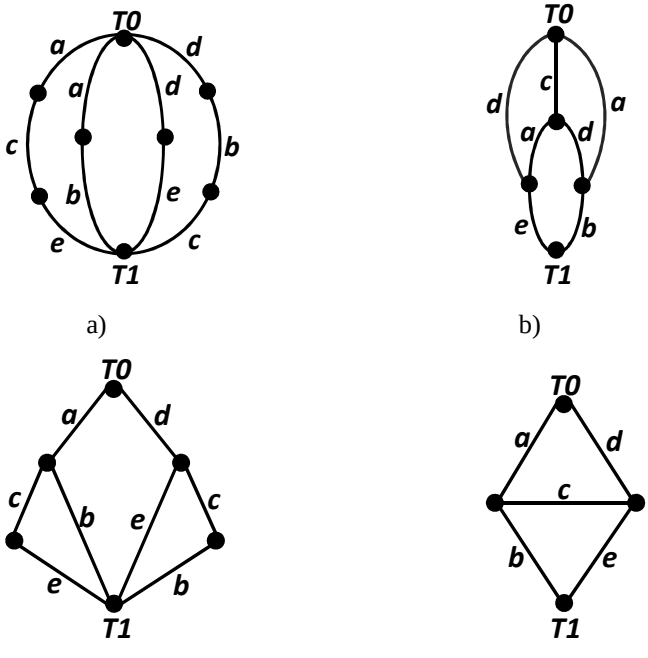


a) b) c)

Figura 7. Algumas possibilidades de solução para a Equação (7).

Outro exemplo é demonstrado na Figura 8 que representa a Equação (8). Neste caso, independente dos candidatos à otimização escolhidos, o número de arestas a ser reduzido será idêntico, o que torna a decisão ainda mais difícil. Note que, quando a aresta *c* é compartilhada antes das outras, o resultado final é o grafo da Figura 8.b. Por outro lado, se o processo de otimização é iniciado compartilhando as arestas *a*, e em uma etapa posterior, as arestas *d*, o grafo resultante é o ilustrado na Figura 8.c. Assim, as arestas *b* podem ser compartilhadas. Posteriormente, é possível compartilhar arestas *e*. E, por fim, é necessário eliminar uma das arestas redundantes *c*. Desta forma, é possível atingir a solução ótima ilustrada pela Figura 8.d.

$$f = a.c.e + a.b + d.e + b.c.d \tag{8}$$



a) b)

c) d)

Figura 8. Algumas possibilidades de solução para a Equação (8).

3.2 Verificação de Equivalência Lógica

Para garantir que a rede de transistores otimizada seja equivalente a SOP de entrada, um procedimento de validação é aplicado para certificar que todos os mintermos estão representados no grafo resultante. Além disso, é preciso garantir que falsos caminhos (*Sneak-Paths*) não sejam introduzidos na rede. Um *Sneak-Path* é um caminho que não coincide com qualquer um dos mintermos expressos na SOP de entrada.

Este procedimento consiste em comparar cada caminho com os produtos que compõem a SOP original. Se um caminho que não pertence à SOP foi introduzido, uma rotina verifica se este caminho é sensibilizável ou não-sensibilizável através de um vetor de entrada. Em redes de transistores um caminho é dito não-sensibilizável se contiver transistores controlados por uma variável com ambas as polaridades, por exemplo, *a* e *!a*. Em outras palavras, este caminho não é um caminho válido na rede. Se o caminho introduzido não é sensibilizável, então ele é aceito, uma vez que não altera o comportamento lógico do circuito. Caso contrário, o grafo precisa ser restaurado a uma etapa anterior a otimização que gerou o caminho inválido, descartando as alterações que tenham levado ao problema. Neste sentido, uma rotina de restauração é invocada. Esta rotina é responsável por recuperar as arestas e vértices que foram eliminados do grafo e reconectá-los na rede. Basicamente, a função dessa rotina é executar o processo inverso ao ilustrado na Figura 3. O algoritmo de recuperação parte de uma configuração como a ilustrada pelo grafo da Figura 3.a e volta para o estado ilustrado pelo grafo da Figura 3.b. Assim, a arresta que derivou o compartilhamento inválido é ignorada nas próximas buscas por candidatos a otimização.

Note que, para o exemplo da função *XOR*, todos os produtos originais da Equação 5 estão presentes no grafo representado pela Figura 4.h. No entanto, através do compartilhamento de arestas, alguns novos caminhos também foram introduzidos. Esses caminhos são permitidos porque não são caminhos sensibilizáveis.

4. Algoritmo Proposto para Extração da Expressão Booleana

O algoritmo discutido até a seção anterior recebe uma expressão como entrada, instancia o grafo e aplica as otimizações. Contudo, estas otimizações são realizadas diretamente sob o grafo. Uma vez que se pode desejar obter a expressão otimizada que representa este grafo, um segundo algoritmo deverá ser aplicado. Esta seção descreve um algoritmo para extrair as expressões Booleanas a partir do grafo que representa as redes de transistores otimizadas.

A extração de uma expressão Booleana requer travessias sucessivas em um grafo de um nó terminal para outro. O algoritmo executa essas travessias aplicando compactação de arestas no grafo. A cada iteração do algoritmo, as arestas adjacentes associadas em série são compactadas, resultando em uma única aresta. Se não houver mais arestas em série a serem compactadas, o algoritmo percorre o grafo novamente, mas desta vez visando à compactação de arestas associadas em paralelo. Quando não há mais arestas em paralelo a serem compactadas, o algoritmo é reiniciado à procura de compactação em série. Este processo é iterativo e pára quando não há possibilidade de aplicar compactação tanto em série quanto em paralelo.

Os passos de compactação do grafo são demonstrados na Figura 9. O grafo inicial é apresentado na Figura 9.a. O processo inicia pesquisando arestas adjacentes ligadas em

série em subgrafos. Neste exemplo, os primeiros conjuntos de arestas a serem compactadas estão ligados ao nó terminal *T0*. Por exemplo, as arestas *!f, !e, !c* estão associadas em série, e podem ser compactadas para uma única aresta. Portanto, o subgrafo composto por essas arestas é substituído por uma nova aresta, que representa um conjunto de arestas. Neste caso, esta nova aresta é representada pelo produto *!f.!e.!c*. De maneira semelhante, as arestas *f.e.c* e *d.b.a* são criadas. Este processo é ilustrado na Figura 9.b.

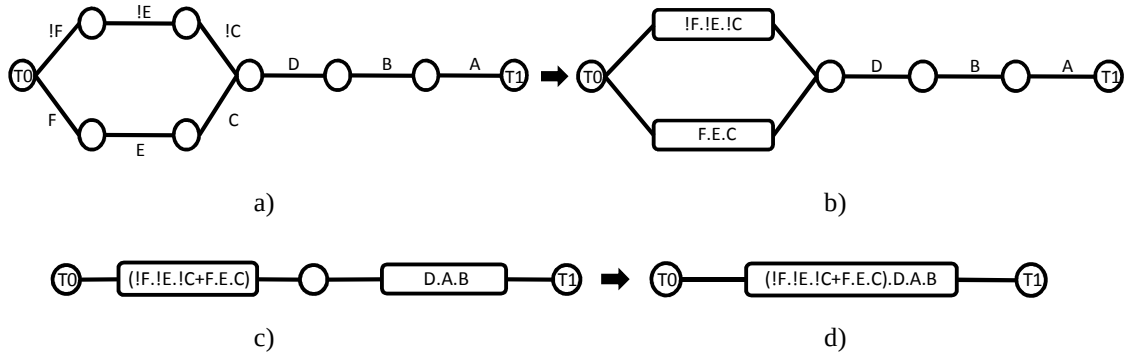
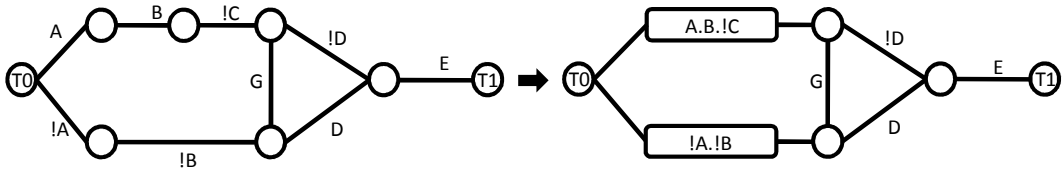


Figura 9. Passos do algoritmo de compactação de arestas.

Depois de executar a primeira iteração da compactação em série, o algoritmo procura por associações paralelas. Na Figura 9.b, existem dois subgrafos em paralelo. Como ilustrado na Figura 9.c, as arestas *!f.!e.!c* e *f.e.c* podem ser compactadas em uma única aresta que será representada como *!f.!e.!c + f.e.c*. Neste ponto, não há mais arestas em paralelo. Então, uma nova iteração tentando encontrar arestas em série é aplicada novamente. Dessa forma, diversas iterações de compactação série/paralelo são aplicadas, finalizando quando não acontece a diminuição no número de arestas do grafo. Assim, o grafo apresentado na Figura 9.c pode ser compactado, resultando no grafo ilustrado pela Figura 9.d. O grafo final apresenta a capacidade de representar a função Booleana através da expressão otimizada (fatorada) *((!f.!e.!c + f.e.c) .d.b.a)*.

Usando compactação no grafo também torna-se possível identificar as configurações *Wheatstone bridge*. Uma vez que nas configurações *bridge* existe pelo menos uma aresta que não encontra-se nem em série e nem em paralelo com as demais, o grafo final obtido após a compactação possuirá ao menos cinco arestas remanescentes (a construção mínima de uma rede *bridge*).

É importante relembrar que o objetivo principal do algoritmo consiste em extrair a expressão Booleana que representa exatamente o grafo otimizado. Quando um grafo pode ser compactado em uma única aresta, isto significa que existem apenas associações de transistores puramente série-paralelo. Isto não acontece no caso de redes com associações *bridge.* Para redes *bridge* um passo adicional deverá ser executado para que a expressão Booleana possa ser derivada. Um exemplo de compactação de grafo que representa uma rede do tipo *bridge* pode ser visto na Figura 10.



a) b)

Figura 10. Passos da compactação aplicados em redes do tipo *bridge.*

Considerando a compactação de grafo de uma rede do tipo *bridge*, é necessário aplicar travessias em todos os caminhos possíveis do grafo, a fim de extrair a expressão Booleana que representa este grafo. Este procedimento final é realizado utilizando uma matriz de adjacência para representar o grafo compactado. A expressão pode ser extraída aplicando um algoritmo baseado no algoritmo de busca em profundidade sobre a matriz de adjacência. A pesquisa inicia sempre em um nodo terminal e finaliza no outro nodo terminal. A Figura 11 apresenta todas as pesquisas sobre a matriz de adjacência que representa o grafo compactado da Figura 10.

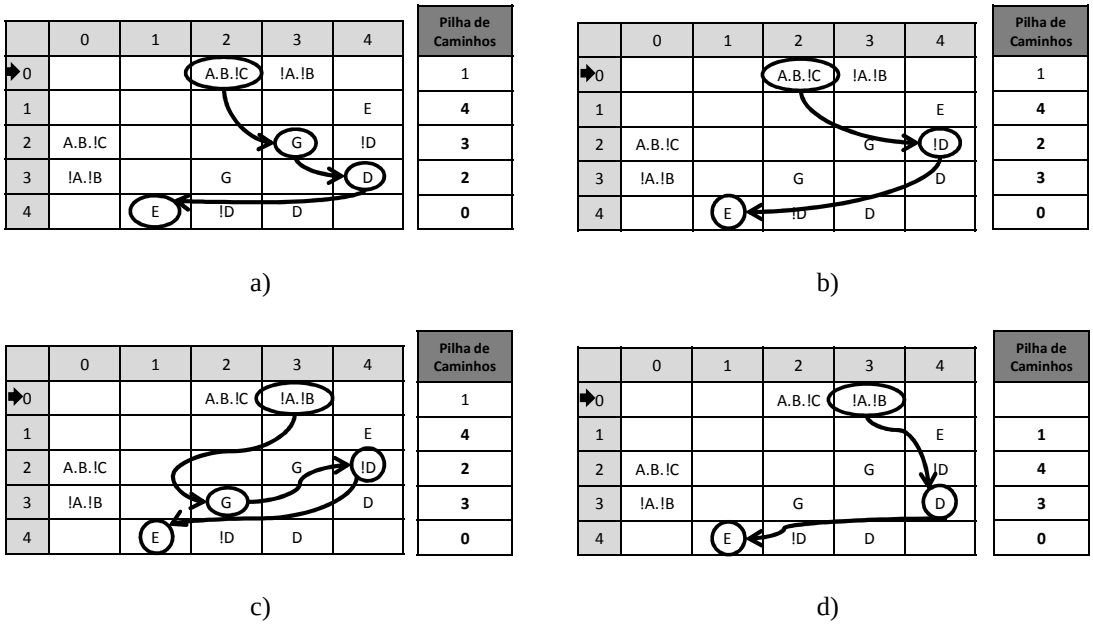


Figura 11. Caminhos na matriz de adjacência no caso de rede *bridge*.

A busca em profundidade inicia-se no nodo terminal *T0*, ou seja, sobre o índice zero da matriz de adjacência. Esse algoritmo utiliza uma pilha para manter o caminho percorrido. Esta pilha armazena os índices da matriz de adjacência que representa os nodos do grafo. A partir da linha zero, o algoritmo procura índices onde são representadas arestas adjacentes no grafo. Neste ponto, a coluna 2, que representa a aresta *a.!b.c*, é a primeira escolha. Portanto, o índice 2 é adicionado à pilha. Através de uma rotina recursiva, a linha 2 é o próximo ponto para pesquisa. Nesta linha, a primeira coluna que indica uma adjacência é a coluna 0. No entanto, esta linha já foi visitada, pois seu índice já está na pilha de vértices visitados. Desta forma, a próxima aresta a ser visitada é a *g*. Portanto, a próxima linha é a de índice 3. Este processo é repetido através dos vértices interligados, e ele é finalizado quando o terminal oposto *T1* é atingido. No exemplo apresentado na Figura 11, o índice 1 representa o nodo terminal *T1*.

No final, a pilha conterá uma sequência de índices, os quais representam um caminho entre ambos os terminais. Cada caminho construído simboliza um produto entre as arestas, expressando um caminho entre os terminais. No exemplo da Figura 11, há quatro caminhos resultantes. A composição das subexpressões obtidas resulta em uma expressão Booleana que representa a rede de transistores. Alguns produtos em um caminho são equivalentes aos produtos de outros caminhos. Assim, técnicas associativas podem ser aplicadas para reduzir o número de literais da expressão final. Neste exemplo, o resultado deste processo é apresentado na Equação (9).

$$f = a.b.!c.(!d.e+g.d.e)+!a.!b.(g.!d.e+d.e) \quad (9)$$

5. Resultados Experimentais

A ferramenta *Soptimizer* foi totalmente desenvolvida na linguagem Java. Os algoritmos propostos foram implementados através de um conjunto de classes Java e foram integrados na ferramenta, assim como a biblioteca Prefuse [Prefuse.org 2011], que é usada para visualização dos grafos gerados. A fim de validar as expressões geradas pelos métodos propostos, experimentos foram realizados em um computador com processador Core2Duo e 4GB de memória RAM, rodando o sistema operacional Ubuntu 10.10-64-bits.

O primeiro experimento realizado utilizou um conjunto de 10 funções Booleanas com 7 variáveis de entrada, que foram escolhidas aleatoriamente para avaliar o comportamento do método proposto neste artigo. Expressões lógicas foram criadas para representar cada uma das 10 funções. Esta expressões serviram de entrada à ferramenta SIS [Sentovich 1992] para que, então, pudéssemos obter suas versões equivalentes em somas de produtos. A Tabela 1 apresenta os resultados obtidos, comparando o método de fatoração da ferramenta SIS com a metodologia proposta considerando o número de literais. O número total de literais da SOP em sua forma canônica é mostrado na segunda coluna. A terceira coluna mostra o número de literais obtidos nas expressões fatoradas usando o algoritmo *Quick-factor* do SIS. A quarta coluna apresenta o número de arestas do grafo resultante da ferramenta *Soptmizer*. Por fim, o percentual de ganho do *Soptimizer* em relação ao SIS é demonstrado na última coluna.

Considerando que o número de literais, assim como o número de arestas do grafo, indicam o número de transistores de uma porta lógica, pode-se dizer que em todos os casos a abordagem proposta obteve resultados melhores. Analisando as redes obtidas pelo *Soptimizer*, foram identificadas diversas configurações do tipo *bridge*. O SIS gera apenas expressões com as primitivas lógicas básicas: *E, OU* e inversão. Logo, redes de transistores geradas a partir destas equações terão apenas associações do tipo série-paralelo. Deste modo, um dos fatores que contribui para o aumento na contagem de transistores é a impossibilidade do uso de associações do tipo *bridge*.

Em um segundo experimento, o mesmo tipo de análise do primeiro experimento foi realizado para o conjunto de funções lógicas de 4 entradas da *classe P*. Este conjunto de funções é composto por 3982 funções lógicas. Cada SOP foi fatorada com a ferramenta SIS utilizando os métodos de fatoração *Quick-factor* e *Good-factor*. Além disso, as SOPs também foram sintetizadas com o *Soptimizer*. O método proposto foi capaz de proporcionar melhores soluções, reduzindo o número total de transistores nas redes finais, conforme descrito na Tabela 2.

Tabela 1. Comparação entre SIS e o *Soptimizer*

Função	Número de literais da SOP	Número de literais resultantes do *Quick-factor*/SIS	Arestas do grafo no Soptimizer	Ganho (%)
F1	133	76	61	19.73
F2	92	56	44	21.42
F3	78	48	39	18.75
F4	150	86	67	22.09
F5	119	68	58	14.70
F6	71	39	36	7.69
F7	170	99	80	19.19
F8	135	85	64	24.70
F9	111	70	56	20.00
F10	97	56	46	17.85

Do total de 3982 funções lógicas, o *Soptimizer* apresentou 1966 redes de transistores menores quando comparadas com as soluções do SIS. Em 1973 redes de transistores a contagem de transistores é exatamente igual ao SIS. Nas outras 43 redes, o método proposto apresentou uma contagem de transistores ligeiramente maior que o SIS. A Figura 12 mostra a distribuição dos ganhos ou perdas da abordagem proposta em relação ao SIS (tanto para o algoritmo *Quick-factor* quanto para *Good-factor*). Como pode ser visto, o método de compartilhamento de arestas do *Soptimizer* é capaz de reduzir até 5 transistores em algumas redes. Por outro lado, o SIS alcança melhores resultados para algumas funções. Nestes casos, o algoritmo implementado no *Soptimizer* não foi capaz de gerar associações do tipo *bridge*.

Tabela 2. Contagem total de transistores para o conjunto de funções *classe P* de 4 entradas

	Total de Transistores	
Soptimizer	34902	
Good-factory	37723	
Quick-factory	38341	
Soptimizer X SIS	**Nº de Funções Lógicas**	**% da *classe P***
Número de Transistores Menor	1966	49.37%
Número de Transistores Igual	1973	49.54%
Número de Transistores Maior	43	1.07%

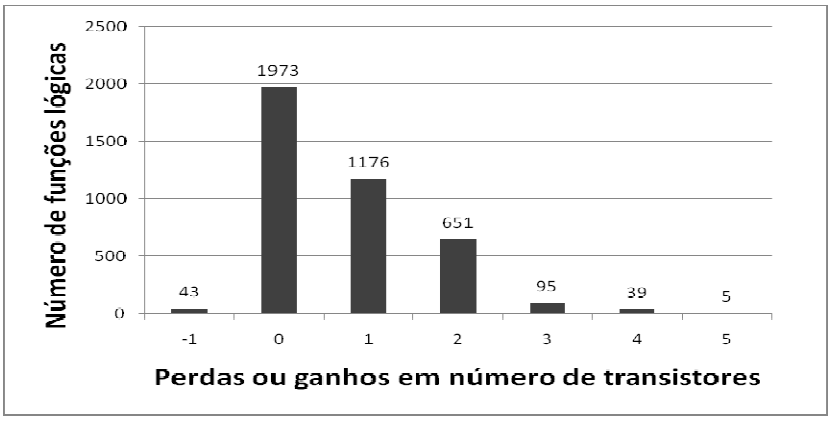


Figura 12. Distribuição de funções Boolenas da *classe P* considerando perdas e ganhos em número de transistores obtidos pelo *Soptimizer* tendo o SIS como referência.

A validação do algoritmo de geração de expressões Booleanas proposto neste trabalho também foi considerada. O resultado da integração do algoritmo de geração de expressões com o *Soptimizer* recebeu o nome de *Soptimizer+* nos resultados experimentais.

O primeiro experimento com o foco na validação do *Soptimizer+* consistiu em aplicar o método proposto no conjunto de funções de 4 entradas da *classe P*. Todas as 3982 expressões foram fatoradas por meio do método de compartilhamento de arestas do *Soptimizer,* sendo que as expressões resultantes foram geradas através do *Soptimizer+* em cerca de 8 minutos. Os resultados obtidos demonstraram que todas as expressões geradas pelo *Soptimizer+* foram lógicamente equivalentes às expressões de entrada.

A Tabela 3 apresenta uma comparação entre a fatoração do SIS e os algoritmos do *Soptimizer e Soptimizer+*, considerando um subconjunto de 10 funções das 3982 da *classe P*. A segunda coluna mostra o número de literais da SOP que representa cada função. A terceira coluna mostra o número de literais da forma fatorada gerada pelo SIS. A quarta coluna apresenta o número de arestas geradas pelo *Soptimizer*. A última coluna apresenta o número de literais alcançado pelo *Soptimizer+*.

Após esses experimentos, o número de transistores (número de arestas dos grafos ou número de literais das expressões) foi contabilizado, a fim de permitir uma comparação do algoritmo de compartilhamento de arestas (*Soptimizer),* com o algoritmo de geração da expressão Booleana a partir do grafo (*Soptimizer+*).

Como esperado, em 2192 funções (de 3982), os números são os mesmos quando o grafo resultante representa somente associações de transistores do tipo série-paralelo. No entanto, quando o grafo resultante é uma associação do tipo *bridge*, as expressões do *Soptimizer+* têm um maior número de literais. Isso se deve ao fato, das primitivas lógicas *E* e *OU* representarem associações série-paralelo nas redes de transistores. Deste modo, termos redundantes são inseridos na expressão Booleana que representam redes com associações do tipo *bridge* no caso das expressões do *Soptimizer+*.

Além disso, também comparou-se as expressões geradas pelo *Soptimizer+* e a fatoração do SIS, usando como entrada as mesmas funções de 4 entradas da *classe P*. O algoritmo proposto neste trabalho obteve melhores resultados em um conjunto de 107 funções. Em 2011 casos, os resultados obtidos foram os mesmos.

Tabela 3. Relação entre 10 funções da *classe P* na contagem de literais e arestas

Função	Número de literais da SOP	Número de literais resultantes do *Quick-factor*/SIS	Arestas do grafo no Soptimizer	Número de literais Soptimizer+
87	14	11	10	10
100	7	6	6	6
126	12	9	9	9
231	8	7	6	9
393	13	10	8	8
594	11	9	8	8
1183	13	10	8	8
2284	16	13	12	12
3622	12	10	9	9
3635	22	15	13	36

No entanto, para 1864 funções, o algoritmo do SIS obteve melhores resultados. Isso pode ser explicado porque atualmente o algoritmo proposto para geração de expressões fatoradas do *Soptimizer+* não é capaz de eliminar caminhos não–sensibilizávies em uma rede de transistores. Desta forma, mesmo que o grafo resultante seja menor em número de arestas, no momento da travessia para a geração da expressão fatorada, torna-se necessário replicar caminhos. Isso acarreta em um aumento no número de literais da expressão final. Um exemplo desse problema é apresentado na Figura 13. O caminho formado pelas arestas *a*, *g* e *!a* não é sensibilizável, dado que a variável *a* aparece em ambas as polaridades. O método proposto gera a expressão *a.(g.!a+d)+!b(g.d+!a)*, com 8 literais, quando poderia ser reduzida para *a.d +!b.(g.d +!a)*, com 6 literais.

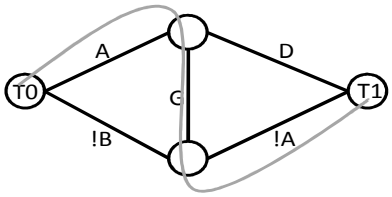


Figura 13. Exemplo de caminho não-sensibilizável.

6. Conclusões e Trabalhos Futuros

Este trabalho apresentou algoritmos baseados em grafos para gerar redes de transistores e extrair expressões Booleanas que representam essas redes. Tais algoritmos demonstram-se factíveis, pois são capazes de tratar redes do tipo *Wheatstone bridge*, diferentemente de técnicas de fatoração, que só podem gerar redes com configuração do tipo série-paralelo. Quando comparado com a configuração série-paralelo, redes *bridge* são capazes de reduzir a contagem de transistores em várias funções lógicas. Esta é uma questão importante, uma vez que a redução do número de transistores em portas lógicas na tecnologia CMOS pode reduzir o consumo de energia e/ou o atraso do circuito.

O método de geração de expressões fatoradas com base no grafo preenche uma lacuna entre o *Soptimizer* e outras ferramentas que utilizam expressões como entradas. O algoritmo proposto foi capaz de gerar as expressões das redes com associações do tipo série-paralelo e associações do tipo *Wheatstone Bridge* representadas por grafos. Expressões Booleanas não tem um operador para denotar associações do tipo *Wheatstone Bridge*. Desta forma, quando este tipo de associação é expresso por operadores primitivos, termos redundantes são adicionados na expressão Booleana.

Como trabalhos futuros, pretende-se eliminar caminhos não-sensibilizáveis das expressões geradas pelo *Soptimizer+*. Isso pode ser feito utilizando métodos Booleanos durante a manipulação do grafo. Desta forma, será possível reduzir a contagem de transistores e alcançar expressões menores em termos de literais.

Referências

Avci, M., Yildirim, T. (2003). General design method for complementary pass transistor logic circuits. *Electronics Letters*, [S.l.], v. 39, n. 1, p.46–48.

Brayton, R. K. (1987). Factoring logic functions. *IBM Journal of Research and Development*, v. 31, n. 2, p. 187-198.

Buch, P. et al. (1997). Logic synthesis for large pass transistor circuits. In: ICCAD, p. 663-670.

Da Rosa Jr., L. S., Marques, F., Cardoso, T., Ribas, R., Sapatnekar, S., Reis, A. (2006). Fast Disjoint Transistor Networks from BDDs. In: SBCCI, p. 137-142.

Da Rosa Jr., L. S., Marques, F., Schneider, F., Ribas, R., Reis, A. (2007). A Comparative Study of CMOS Gates with Minimum Transistor Stacks. In: SBCCI, p. 93-98.

Da Rosa Jr., L. S., Schneider, F., Ribas, R. P., Reis, A. I. (2009). Switch Level Optimization of Digital CMOS Gate Networks. In: ISQED, p. 324-329.

Dietmeyer, D. L. (1971). *Logic design of digital systems*. Allyn and Bacon.

Hsiao, S., Yeh, J., Chen, D. (2000). High performance multiplexer-based logic synthesis using pass-transistor logic. In: ISCAS, p. 325-328.

Golumbic, M. C., Mintz, A., Rotics, U. (2008). An improvement on the complexity of factoring read-once Boolean functions. *Discrete Appl. Math*, Vol. 156, n. 10, 1633-1636.

Kagaris, D. et al. (2007). A Methodology for Transistor-Efficient Supergate Design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 488-492.

Karnaugh, M. (1953). The Map Method for the Synthesis of Combinational Circuits. *AIEE Transactions*, [S.l.], p. 593-599.

Kohavi, Z. (1970). *Switching and Finite Automata Theory*. New York: McGraw-Hill.

McCluskey, E. J. (1956). Minimization of Boolean Functions. *Bell Systems Technical Journal*, [S.l.], v. 35, p. 1417-1444.

Mcgeer, P. et al. (1993). Espresso-Signature: A new exact minimizer for logic functions. In: DAC, p. 432-440.

Mintz, A., Golumbic, M. C. (2005). Factoring boolean functions using graph partitioning. *Discrete Appl. Math*, 149, 1-3, 131-153.

Possani, V. N., Timm, E. F., Agostini, L. V., Da Rosa Junior, L. S. (2011). A Graph-based Technique to Optimize Transistor Networks. In: LASCAS, p. 1-4.

Prefuse.org. The Prefuse Visualization Toolkit. [Online] Avaliable: http://prefuse.org/ [Acessed: Mar. 19, 2011].

Quine, W. V. (1955). A Way To Simplify Truth Functions. *American Mathematical Monthly*, [S.l.], v. 62, p. 627-631.

Sentovich, E., Singh, K., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P., Brayton, R., Sangiovanni-Vincentelli, A. (1992). SIS: A system for sequential circuit synthesis. Tech. Rep. UCB/ERL M92/41. UC Berkeley.

Shelar, R., Sapatnekar, S. (2001). Recursive bipartitioning of BDDs for performance driven synthesis of pass transistor logic circuits. In: ICCAD, p. 449-452.

Shelar, R., Sapatnekar, S. (2002). An efficient algorithm for low power pass transistor logic synthesis. In: ASPDAC, p. 87-92.

Wu, M., Shu, W., Chan, S. (1985). A unified theory for MOS circuit design switching network logic. *Int. J. Electron*., vol. 58, no. 1, pp. 1-33.

Yoshida, H., Ikeda, M., Asada, K. (2006). Exact Minimum Logic Factoring via Quantified Boolean Satisfiability. In: ICECS, p. 1065-1068.

Zhu, J., Abd-El-Barr, M. (1993). On the optimization of MOS circuits. *IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications, Theory Appl.*, vol. 40, no. 6, pp. 412–422.