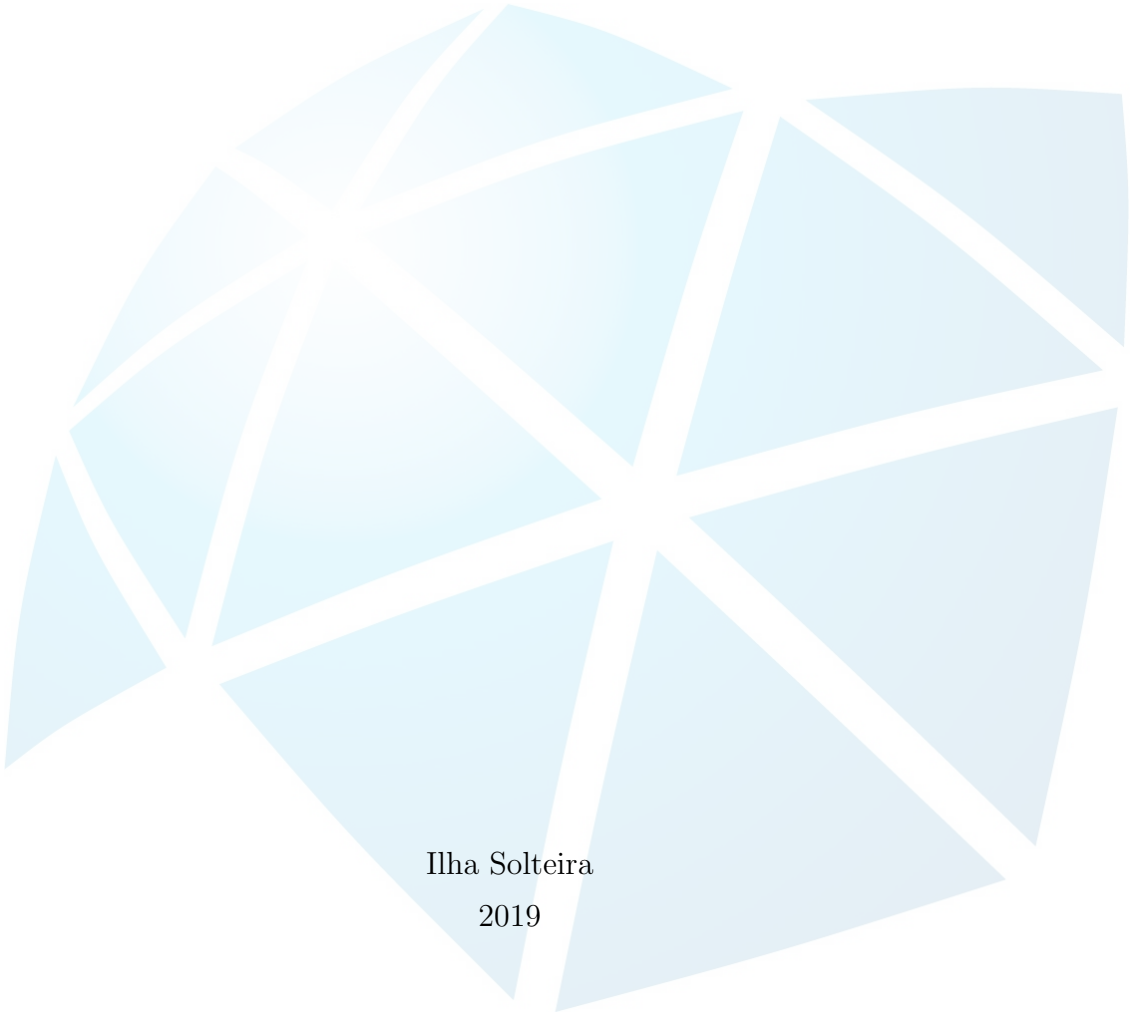
UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”

FACULDADE DE ENGENHARIA

CAMPUS DE ILHA SOLTEIRA

DOUGLAS UKA RENNÓ

OTIMIZAÇÃO PÓS-SÍNTESE DE CIRCUITOS REVERSÍVEIS UTILIZANDO MÉTODOS HEURÍSTICOS



Ilha Solteira

2019

DOUGLAS UKA RENNÓ

Otimização pós-síntese de circuitos reversíveis utilizando métodos heurísticos

Dissertação apresentada à Faculdade de Engenharia – UNESP – Campus de Ilha Solteira como parte dos requisitos exigidos para obtenção do título de Mestre em Engenharia Elétrica. Área de Conhecimento: Automação.

Prof. Dr. Alexandre César Rodrigues da Silva
Orientador

Ilha Solteira
2019

FICHA CATALOGRÁFICA
Desenvolvido pelo Serviço Técnico de Biblioteca e Documentação

Rennó, Douglas Uka.
R414o Otimização pós-síntese de circuitos reversíveis utilizando métodos heurísticos / Douglas Uka Rennó. -- Ilha Solteira: [s.n.], 2019
66 f. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia . Área de conhecimento: Automação, 2019

Orientador: Alexandre César Rodrigues da Silva
Inclui bibliografia

1. Otimização pós-síntese. 2. Método de otimização. 3. Métodos heurísticos. 4. Simulated annealing. 5. Variable neighbourhood descent.

Sandra Maria Clementa de Souza
CRB 8-4740

unesp

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO DA DISSERTAÇÃO: Otimização pós-síntese de circuitos reversíveis utilizando métodos heurísticos

AUTOR: DOUGLAS UKA RENNÓ
ORIENTADOR: ALEXANDRE CESAR RODRIGUES DA SILVA

Aprovado como parte das exigências para obtenção do Título de Mestre em ENGENHARIA ELÉTRICA, área: Automação pela Comissão Examinadora:

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

Prof. Dr. RUBEN AUGUSTO ROMERO LAZARO
Departamento de Engenharia Elétrica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. EVANDRO MAZINA MARTINS
Faculdade de Engenharias, Arquitetura e Urbanismo / Universidade Federal de Mato Grosso do Sul

Ilha Solteira, 18 de janeiro de 2019

AGRADECIMENTOS

Agradeço a todos que, de alguma maneira, me auxiliaram, ajudaram e deram forças a finalizar este trabalho. Primordialmente a minha família pelo apoio e incentivo nos momentos difíceis.

Ao Professor Dr. Alexandre César Rodrigues da Silva pela oportunidade de trabalhar junto ao grupo do Laboratório de Processamento de Sinais e Sistemas Digitais (LPSSD). Agradeço também pela confiança dada mesmo diante das dificuldades.

Em especial, ao aluno de mestrado Marcos Gomes Vieira por me incentivar no início do processo de mestrado.

Agradeço aos membros do laboratório pela ótima convivência durante o tempo de pesquisa. Sobretudo, aos Doutorandos Alexandre A. A. de Almeida e Willian Ferreira.

Agradeço à CAPES pelo fomento nesta pesquisa, ao CNPq, processo n ° 309193/2015-0 e à Pós-graduação em Engenharia Elétrica da Faculdade de Engenharia de Ilha Solteira - UNESP (Universidade Estadual Paulista) ao suporte concedido a este trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Neste trabalho foram programados dois algoritmos descritos na literatura denominados de XOR e MDM que realizam a síntese de circuitos reversíveis a partir da tabela verdade. Programou-se também algoritmos relacionados com a otimização pós-síntese, denominados *Greedy*, *Simulated Annealing* e *Variable Neighbourhood Descent*, que empregam métodos heurísticos e regras de reescrita, cujo objetivo é reduzir a quantidade de portas lógicas reversíveis do circuito sintetizado. A contribuição deste trabalho foi o emprego do método Divisão que divide o circuito sintetizado em vizinhanças e aplica o método *Simulated Annealing* ou *Variable Neighbourhood Descent* nas partes do circuito. Os métodos de otimização implementados foram comparados utilizando como testes 42 circuitos. Constatou-se que os métodos *Simulated Annealing* e *Variable Neighbourhood Descent* em conjunto com o método Divisão geraram circuitos menores. Além disso, o algoritmo que aplica a meta-heurística *Simulated Annealing* comparado ao *Variable Neighbourhood Descent* obteve menor quantidade de portas em 7 dos 42 circuitos, mesmo custo em 29 circuitos e pior custo em 6.

Palavras-chave: Otimização pós-síntese. Método de otimização. Métodos heurísticos. *Simulated Annealing*. *Variable Neighbourhood Descent*.

ABSTRACT

In this work, two algorithms described in the literature denominated of XOR and MDM are programmes that realize the synthesis of reversible circuits from the truth table. It has been programmed also algorithms related to the post-synthesis optimization, called Greedy, Simulated Annealing and Variable Neighbourhood Descent, which use heuristic methods and rewriting rules, whose objective is to reduce the number of reversible logic gates of the synthesized circuit. The contribution of this work was the use of the Division method that divides the synthesized circuit into neighborhoods and applies the Simulated Annealing or Variable Neighbourhood Descent method in the circuit parts. The implemented optimization methods were compared using 42 circuits as a test. It was found that the Simulated Annealing and Variable Neighborhood Descent methods together with the Division method generated smaller circuits. Furthermore, the algorithm that applies the Simulated Annealing meta-heuristic compared to the Variable Neighbourhood Descent obtained the lowest number of gates in 7 of the 42 circuits, even cost in 29 circuits and the worst cost in 6.

Keywords: Post-synthesis optimization. Optimization method. Heuristic methods. Simulated Annealing. Variable Neighborhood Descent.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AG	*Adaptive Genetic*
BDD	*Binary Decision Diagram*
CCNOT	*Controlled-Controlled*-NOT
CNOT	*Controlled*-NOT
DCs	Don't Cares
DT	*Dynamic Template*
ESOP	*Exclusive-or sum-of-products*
GA	*Genetic Algorithm*
MB	Megabytes
MCT	*Multiple-Control* Toffoli *gates*
MPMCT	*Mixed-Polarity Multiple-Control* Toffoli *gates*
MTG	*Modified* Toffoli *Gate*
NCT	NOT CNOT TOFFOLI
PSO	*Particle Swarm Optimization*
QTS	*Quantum Tabu search*
SA	*Simulated Annealing*
SAT	*Boolean Satisfiability*
TS	*Tabu Search*
VND	*Variable Neighbourhood Descent*
XOR	*Exclusive-OR*

SUMÁRIO

1 INTRODUÇÃO

Devido à grande quantidade de transistores integrados em uma mesma área de um circuito integrado (CI), a dissipação de energia na forma de calor se tornou um grande problema para o avanço da tecnologia. Esse conceito afirma que utilizando a lógica convencional (irreversível) há perda de informação (bit) (LANDAUER, 1961).

Hänninen, Lent e Snider apresentaram um estudo para estimar a quantidade de informação perdida em circuitos que empregam a lógica irreversível, sendo que a cada bit perdido tem-se dissipação de $k*T*\ln(2)$ Joules de energia térmica, sendo k a constante Boltzmann e T a temperatura ambiente em que computação é executada. Apesar de relativamente pequena, não se pode negligenciar, levando-se em consideração a grande quantidade de transistores presentes nos computadores atuais (HÄNNINEN; LENT; SNIDER, 2014).

Com o intuito de solucionar esse problema, Bennet propôs o emprego da lógica reversível com o objetivo de diminuir ou até mesmo, teoricamente, zerar a dissipação de calor e consequentemente a perda de energia. Uma das características da lógica reversível é a relação biunívoca entre as variáveis de entrada e as funções de saída, ou seja, para um determinado vetor de entrada tem-se um determinado vetor de saída e vice-versa (BENNETT, 1973).

A lógica reversível vem se tornando prioridade no projeto de circuitos digitais quando o assunto é computação quântica. Na literatura, mostrou-se que as portas reversíveis podem ser construídas em tecnologias como CMOS (DESOETE; VOS, 1999; VASUDEVAN; LALA; PARKERSON, 2005; CHACKO; WHIG, 2016), computação ótica (PICTON, 1991; SHUKLA *et al.*, 2017) e nanotecnologia (MERKLE, 1993; GOVINDAPRIYA; PERIYASAMY, 2016). No entanto, a maioria das portas utilizadas em projetos de circuitos digitais não são reversíveis como, por exemplo, as portas OR, AND e XOR. Portanto, ao longo do tempo foram criadas portas reversíveis, entre elas estão as portas CNOT (FEYNMAN, 1985) e TOFFOLI (TOFFOLI, 1980).

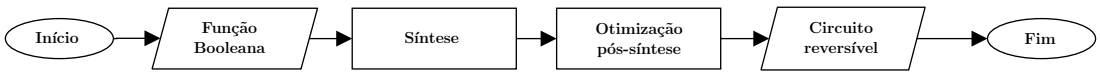
Neste trabalho, estudou-se sobre a lógica reversível e as etapas de síntese e otimização pós-síntese. Na Figura 1, apresenta-se o fluxograma simplificado para a síntese de circuitos reversíveis.

A síntese é a etapa mais importante, consequentemente a mais complexa. O objetivo é transformar uma função irreversível para reversível e posteriormente efetuar a construção do circuito (MILLER; MASLOV; DUECK, 2003; SARKAR; GHOSAL; MOHANTY, 2013; SOEKEN *et al.*, 2017).

A estrutura de um circuito que emprega a lógica reversível é composta por uma sequência de portas reversíveis, isto é, não é possível conter *fan-out* ou realimentação.

A otimização pós-síntese é a etapa final e tem como funcionalidade reduzir o custo de circuitos reversíveis. Tal custo pode ser calculado de duas maneiras: quantidade de portas no circuito (SOEKEN; THOMSEN, 2013; RAHMAN; SOEKEN; DUECK, 2015) e custo quântico (DATTA; SENGUPTA; RAHAMAN, 2015; ABDESSAIED *et al.*, 2015).

Figura 1 – Fluxograma simplificado das etapas para a síntese de circuitos reversíveis.



Fonte: Elaborado pelo próprio autor.

O objetivo desta pesquisa foi desenvolver métodos de otimização para reduzir a quantidade de portas em circuitos reversíveis, utilizando como metodologia os métodos heurísticos junto às regras de reescrita apresentadas por Soeken e Thomsen (2013). Os métodos heurísticos utilizados foram o *Greedy*, *Simulated Annealing* (SA) e *Variable Neighbourhood Descent* (VND).

Para melhorar o desempenho dos métodos SA e VND, desenvolveu-se também um método denominado Divisão que foi adicionado ao processo de otimização para diminuir o espaço de busca. A estratégia é dividir o circuito em várias vizinhanças e fazer uma busca pelo ótimo local em cada uma.

Cabe salientar que a implementação dos métodos foi realizada no Revkit, um $framework$ *Open Source* que utiliza a linguagem C++ para escrever os algoritmos. Este $framework$ foi disponibilizado por Wille *et al.* (2008) com o propósito de armazenar todos os testes e métodos da lógica reversível que foram desenvolvidos.

Nas Seções 1.1 e 1.2 são apresentados respectivamente trabalhos recentes que aplicam a lógica reversível e a descrição do conteúdo de cada capítulo do trabalho.

1.1 TRABALHOS RELACIONADOS À LÓGICA REVERSÍVEL

Nesta seção, apresentam-se os trabalhos recentes relacionados à lógica reversível e que foram estudados, servindo como base para o desenvolvimento dos métodos desenvolvidos. As subseções são separadas conforme as etapas da lógica reversível, ou seja, a síntese e a otimização pós-síntese.

1.1.1 Síntese

Bahauddin e Irfan (2012) apresentaram um estudo sobre as propriedades de ciclo e transposição para revelar o potencial da álgebra de permutação na lógica reversível. Também foi apresentado um método utilizando as expressões de ciclo e transposições para a construção de circuitos reversíveis.

Datta, Sengupta e Rahaman (2012) aplicaram a meta-heurística *Particle Swarm ptimization* (PSO) na síntese para a construção de circuitos reversíveis. Como resultado, comparou-se o método PSO com outros algoritmos como, por exemplo, *MOSAIC algorithm* (SAEEDI; ZAMANI; SEDIGHI, 2008), *Reversible Logic Synthesis* (DATTA *et al.*, 2012) e etc.

Datta *et al.* (2013) utilizaram uma técnica de síntese para a construção de circuitos reversíveis com base na teoria de ciclo de permutação. O método aplicado utiliza regras de decomposição para dividir ciclos maiores em menores; em seguida, os ciclos menores com diferença menor que 3 bits são sintetizados utilizando um algoritmo apresentado por Datta *et al.* (2012).

Bandyopadhyay *et al.* (2013) desenvolveram uma técnica de síntese baseada em *Exclusive-or sum-of-products* (ESOP) para a construção de circuitos reversíveis utilizando uma lista de cubos melhorada para funções com até 16 variáveis de entrada.

Sarkar, Ghosal e Mohanty (2013) apresentaram um método estocástico para sintetizar um circuito reversível. Este procedimento é baseado em uma versão modificada do método clássico de Quine-McCluskey e está sendo utilizado o envoltório de duas técnicas inteligentes da busca estocástica: *Simulated Annealing* e *Ant Colony Optimization*.

Zeng *et al.* (2015) propuseram uma nova técnica para reduzir a quantidade de entrada de funções baseada na relação entre hipercubo e portas. O algoritmo possibilita classificar funções com as mesmas propriedades em uma classe isomórfica que se aplica a qualquer valor lógico de entrada, podendo não apenas reduzir a dimensão da função, mas também

acelerar o tempo de execução do algoritmo.

Sasamal, Singh e Mohan (2015) apresentaram um algoritmo para sintetizar circuitos reversíveis utilizando a meta-heurística *Adaptive Genetic* (AG). O algoritmo produz uma cascata de portas Toffoli para uma determinada especificação do circuito.

Kole *et al.* (2017) desenvolveram dois métodos para aplicar na síntese da lógica reversível ternária. O primeiro método é baseado em *Boolean Satisfiability* (SAT) e o segundo método na técnica de busca heurística de nível limitado.

1.1.2 Otimização pós-síntese

Wang *et al.* (2012) apresentaram uma técnica para otimizar circuitos reversíveis baseado em *Quantum Tabu search* (QTS). *Quantum Tabu Search* é um algoritmo avançado que utiliza as estratégias de diversificação e intensificação baseado na computação quântica.

Soeken e Thomsen (2013) descreveram as regras básicas que são necessárias para executar a reescrita de circuitos reversíveis. A maior diferença com relação às abordagens existentes é o uso de controles negativos. Essas regras são denominadas de regras de reescrita e possuem a finalidade de modificar o circuito sem alterar a respectiva funcionalidade, ou seja, a tabela verdade.

Sasamal, Singh e Mohan (2015) apresentaram um algoritmo utilizando a meta-heurística *Genetic Algorithm* (GA) para a otimização de circuitos reversíveis. Essa técnica foi aplicada em funções reversíveis com uma quantidade de variáveis menor que 3.

Rahman, Soeken e Dueck (2015) propuseram uma heurística de combinações de modelos dinâmicos para otimizar circuitos reversíveis, possibilitando encontrar novas regras de reescrita para cada situação.

Datta, Sengupta e Rahaman (2015) propuseram uma técnica de otimização para portas que apresentam controle positivo e negativo. Essa técnica baseia-se em aplicar um pequeno conjunto de regras repetidas vezes. Essas regras têm por objetivo fundir ou substituir portas adjacentes no circuito reversível.

Zeng *et al.* (2015) desenvolveram uma técnica para efetuar a otimização antes de gerar uma ESOP para a função. O intuito desse método é gerar um melhor circuito inicial em comparação à utilização de técnicas que tendem a minimizar uma ESOP apenas.

Bandyopadhyay, Parekh e Rahaman (2016) propuseram um esquema de síntese de

circuitos reversíveis que utiliza uma lista de cubos e efetua o compartilhamento das linhas de saída em pares, isto é, combinam as linhas de saída para encontrar o melhor vizinho e assim, projetar circuitos menores. Essa aplicação é utilizada basicamente para circuitos reversíveis baseados na representação ESOP.

Podlaski (2016) apresentou uma nova implementação de algoritmo baseada em transformação, levando a importantes restrições de memória do algoritmo. Por outro lado, qualquer função Booleana pode ser representada utilizando o Binary Decision Diagram (BDD).

Abubakar *et al.* (2016) propuseram uma nova biblioteca de portas reversíveis para otimização de circuitos reversíveis. O experimento realizado com a meta-heurística GA possibilita escolher o tipo de porta a ser utilizada para reduzir o custo do circuito.

Deshpande, Jayashree e Agrawal (2017) propuseram trabalhar com novos modelos e regras para o processo de otimização de circuitos reversíveis. O intuito do algoritmo *Modified* Toffoli *Gate* (MTG) é trabalhar com controles negativos, além dos positivos.

Almeida, Dueck e Silva (2018) apresentaram um método que aplica a meta-heurística *Tabu Search* para reduzir a quantidade de portas nos circuitos reversíveis. Comparou-se os resultados obtidos com os apresentados por Rahman, Soeken e Dueck (2015), em que obteve-se circuito com custo menor ou igual em 39 dos 42 casos testados.

Renno *et al.* (2018) desenvolveram um método para reduzir a quantidade de portas nos circuitos reversíveis, que emprega como metodologia a meta-heurística *Simulated Annealing* e as regras de reescrita. Constatou-se que comparado aos resultados disponibilizados por Rahman, Soeken e Dueck (2015), o método Divisão com o *Simulated Annealing* obteve 93,3% dos circuitos com custo equivalente ou menor.

1.2 DESCRIÇÃO DO TRABALHO

Nesta seção, descreve-se sobre cada capítulo do trabalho.

No capítulo 2 são apresentadas as características de uma função reversível, as portas reversíveis, a estrutura de um circuito reversível e como gerar a tabela verdade de um circuito reversível.

No capítulo 3, apresenta-se algoritmos para a síntese de circuitos reversíveis. O processo de síntese é dividido em duas fases: a transformação da função irreversível para reversível e a construção do circuito a partir da tabela verdade. Para detalhar

sobre essas etapas foram programados dois algoritmos descritos na literatura: o XOR (MASLOV; DUECK, 2004) e o MDM (MILLER; MASLOV; DUECK, 2003). O XOR é utilizado para preencher a tabela verdade na transformação da função irreversível para reversível e o algoritmo MDM para efetuar a construção do circuito reversível de uma função reversível a partir da tabela verdade.

No capítulo 4 é apresentada a otimização pós-síntese para circuitos reversíveis. Neste capítulo são apresentados os três métodos de otimização desenvolvidos denominados de método *Greedy*, *Simulated Annealing* e *Variable Neighbourhood Descent*. Além disso, apresenta-se também o método Divisão, adicionado ao processo de otimização para diminuir o espaço de busca.

No capítulo 5 são disponibilizados os resultados obtidos com a aplicação dos métodos propostos no capítulo 4, tendo como base de teste 42 circuitos reversíveis. Além disso, comparou-se os métodos com os algoritmos apresentados por Rahman, Soeken e Dueck (2015) e Almeida, Dueck e Silva (2018), que utilizam respectivamente a heurística *Dynamic Template* e a meta-heurística *Tabu Search*.

No capítulo 6, apresenta-se uma descrição do que foi realizado durante a pesquisa e a decisão final sobre o método mais eficiente para reduzir a quantidade de portas nos circuitos reversíveis.

2 LÓGICA REVERSÍVEL

Uma das características da lógica reversível é a relação biunívoca entre as variáveis de entrada e as funções de saída, ou seja, para um determinado vetor de entrada tem-se um determinado vetor de saída e vice-versa. Cabe salientar que a lógica Booleana convencional é considerada irreversível.

Neste capítulo apresenta-se conceitos sobre a lógica reversível. Na Seção 2.1 são apresentadas as definições de uma função reversível. Na seção 2.2 são apresentadas as portas reversíveis. E para finalizar, apresenta-se a estrutura de um circuito reversível e como gerar a tabela verdade de um circuito reversível.

2.1 FUNÇÃO REVERSÍVEL

Na literatura descreve-se dois tipos de funções: as funções denominadas de saída simples e as funções denominadas de múltiplas saídas.

Uma função Booleana de saída simples é definida matematicamente por $f : \mathbb{B}^n \rightarrow \mathbb{B}$, com $\mathbb{B} \in \{0,1\}$ e $n \in \mathbb{N}^*$. Além disso, na forma algébrica é representada por $f(x_1, x_2, ..., x_n) = f'(x)$, com n entradas e uma saída. Dessa forma, a tabela verdade é constituída por $(n + 1)$ colunas e 2^n linhas (l). Por exemplo, as funções OR($\vee$), AND($\wedge$) e NOT(-) apresentadas na Tabela 1.

Tabela 1 – Tabelas verdade das operações básicas OR, AND e NOT.

(a) OR.

Entrada		Saída
x_1	x_2	$f'(x) = x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

(b) AND.

Entrada		Saída
x_1	x_2	$f'(x) = x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

(c) NOT.

Entrada	Saída
x_1	$f'(x) = \bar{x_1}$
0	1
1	0

Fonte: Adaptado de (TOCCI; WIDMER; MOSS, 2017).

Cabe salientar que qualquer função Booleana pode ser derivada das funções básicas OR, AND e NOT. Por exemplo, a função XOR apresentada na Tabela 2.

Tabela 2 – Tabela verdade de uma função XOR.

Entrada		Saída
x_1	x_2	$f'(x) = (x_1 \wedge \bar{x_2}) \vee (\bar{x_1} \wedge x_2)$
0	0	0
0	1	1
1	0	1
1	1	0

Fonte: Adaptado de (TOCCI; WIDMER; MOSS, 2017).

Cada conjunto de entrada pode ser convertido e representado na forma decimal. Por exemplo, na Tabela 2, o conjunto $f(1,1) = f'(0)$ é equivalente a $f(3) = f'(0)$ (TOCCI; WIDMER; MOSS, 2017).

Uma função Booleana de múltiplas saídas é definida matematicamente por $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$, com $\mathbb{B} \in \{0,1\}$ e n, $m \in \mathbb{N}^*$. Além disso, na forma algébrica é representada por $f(x_1, x_2, ..., x_n) = (f'_1(x_1, x_2, ..., x_n), f'_2(x_1, x_2, ..., x_n),..., f'_m(x_1, x_2, ..., x_n))$, com n entradas e m saídas.

Dessa forma, a tabela verdade é constituída por $(n + m)$ colunas e 2^n linhas (l). Por exemplo, a função apresentada na Tabela 3 que possui com 3 entradas e 2 saídas.

Tabela 3 – Exemplo de uma função de múltiplas saídas.

$f(x_1, x_2, x_3)$	Entrada			Saída	
	x_1	x_2	x_3	x'_1	x'_2
$f(0)$	0	0	0	0	0
$f(1)$	0	0	1	1	0
$f(2)$	0	1	0	1	1
$f(3)$	0	1	1	1	0
$f(4)$	1	0	0	0	1
$f(5)$	1	0	1	0	1
$f(6)$	1	1	0	1	1
$f(7)$	1	1	1	1	1

Fonte: Elaborado pelo próprio autor.

Nesta pesquisa, trabalhou-se com a lógica reversível em funções Booleanas. Uma função Booleana é reversível se:

1. A quantidade de entradas e saídas são iguais $(n = m)$;

2. A função é bijetora. Uma função é bijetora ou bijetiva se cada entrada é mapeada para uma única saída e vice-versa, para todas as combinações possíveis.

Na Tabela 4 apresentam-se respectivamente as tabelas verdade de uma função irreversível e reversível. Na Tabela 4(a) é representada uma função irreversível, visto que os conjuntos de entrada $f(6)$ e $f(7)$ são mapeados para o mesmo conjunto de saída $f'(6)$. Por outro lado, na Tabela 4(b) é exibida uma função reversível, pois cada conjunto de entrada é direcionado para um único conjunto de saída. Para facilitar, o valor decimal de cada conjunto de saída é exposto ao lado direito da tabela.

Tabela 4 – Tabelas verdade de uma função irreversível e reversível.

(a) Função irreversível.

$f(x_1, x_2, x_3)$	Entrada			Saída		
	x_1	x_2	x_3	x'_1	x'_2	x'_3
$f(0)$	0	0	0	0	0	0
$f(1)$	0	0	1	1	0	0
$f(2)$	0	1	0	1	1	1
$f(3)$	0	1	1	1	0	1
$f(4)$	1	0	0	0	1	1
$f(5)$	1	0	1	0	1	0
$f(6)$	1	1	0	1	1	0
$f(7)$	1	1	1	1	1	0

(b) Função reversível.

$f(x_1, x_2, x_3)$	Entrada			Saída		
	x_1	x_2	x_3	x'_1	x'_2	x'_3
$f(0)$	0	0	0	0	0	0
$f(1)$	0	0	1	1	0	0
$f(2)$	0	1	0	1	1	1
$f(3)$	0	1	1	1	0	1
$f(4)$	1	0	0	0	1	1
$f(5)$	1	0	1	0	1	0
$f(6)$	1	1	0	1	1	0
$f(7)$	1	1	1	0	0	1

Fonte: Elaborado pelo próprio autor.

Nesta seção foram apresentadas as características das funções Booleanas, denominadas de funções de saída simples, funções de múltiplas saídas e funções reversíveis. Na Seção 2.2 são apresentadas as portas lógicas que operam de acordo com as definições de reversibilidade, denominadas de portas reversíveis.

2.2 PORTAS REVERSÍVEIS

Dentre várias portas reversíveis disponibilizadas na literatura, como as portas *Double Feynman*, *Fredkin*, *Peres* e outras (YELEKAR; CHIWANDE *et al.*, 2011), neste trabalho utilizam-se apenas três, sendo elas as portas reversíveis NOT, CNOT e Toffoli.

Essas portas pertencem a biblioteca NCT (NOT CNOT TOFFOLI) e podem conter os seguintes itens:

- controle positivo (●): é ativo se a variável de entrada for 1 (ativo em 1);

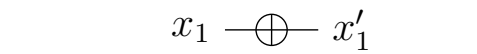
- controle negativo (○): é ativo se a variável de entrada for 0 (ativo em 0);

- alvo (⊕): tem como finalidade negar o valor lógico da variável de entrada apenas se todos os controles forem ativos.

2.2.1 Porta NOT

Uma porta NOT, conforme é apresentada na Figura 2, é composta apenas pelo alvo e é considerada naturalmente reversível, pois adequa-se a todas as características da reversibilidade.

Figura 2 – Porta NOT.



Fonte: Elaborado pelo próprio autor.

2.2.2 Porta CNOT

Uma porta CNOT, também conhecida por FEYNMAN, é composta pelo alvo e por apenas um controle, positivo ou negativo. Nas Figuras 3(a) e 3(b), apresentam-se respectivamente portas CNOT com controle positivo e negativo (FEYNMAN, 1985).

2.2.3 Porta Toffoli

Uma porta Toffoli, também conhecida por *Controlled-Controlled*-NOT (CCNOT) é composta por dois ou mais controles e o alvo.

Existem dois tipos de portas Toffoli reconhecidas na bibliografia denominadas

Figura 3 – Porta CNOT.

(a) Controle positivo. (b) Controle negativo.

Fonte: Adaptado de (FEYNMAN, 1985).

Multiple-Control Toffoli *gates* (MCT) e *Mixed-Polarity Multiple-Control* Toffoli *gates* (MPMCT) (TOFFOLI, 1980).

A diferença entre as portas é relacionada de acordo com o controle utilizado. A porta MCT contém apenas um tipo de controle (positivo ou negativo), enquanto a porta MPMCT apresenta os dois tipos de controle. Nas Figuras 4(a) e 4(b) apresentam-se as portas MCT com controle positivo e MPMCT, respectivamente.

Figura 4 – Porta Toffoli.

(a) Porta MCT com controle positivo. (b) Porta MPMCT.

Fonte: Adaptado de (TOFFOLI, 1980).

Apresentadas as portas reversíveis, na Seção 2.3, apresenta-se a estrutura de um circuito reversível e suas restrições.

2.3 CIRCUITO REVERSÍVEL

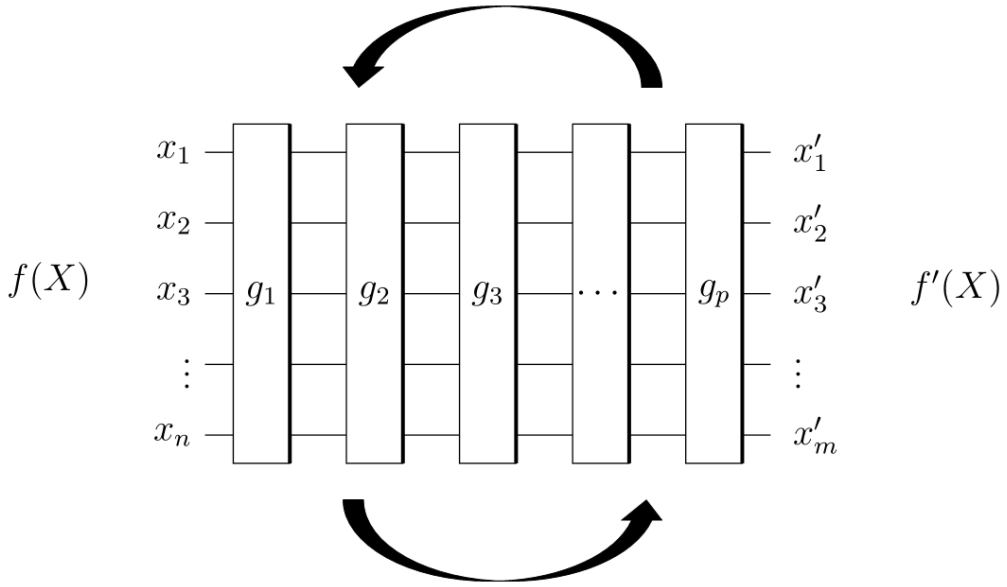
Um circuito reversível é composto por uma sequência de portas reversíveis que permite encontrar um conjunto de saídas para cada conjunto de entradas e vice-versa, para todas as combinações possíveis de uma função. Isto é, deve ser possível encontrar o estado inicial a partir do estado final, ou seja, o modo inverso do circuito.

Apresenta-se na Figura 5 um circuito reversível generalizado com p portas reversíveis

e n entradas, que é definido matematicamente por G = $g_1 g_2 ... g_p$ (WILLE; DRECHSLER, 2010).

Os circuitos reversíveis não podem apresentar *fan-out* ou realimentação (NIELSEN; CHUANG, 2000), pois são construídos por uma sequência de portas reversíveis.

Figura 5 – Estrutura generalizada de um circuito reversível.



Fonte: Adaptado de (WILLE; DRECHSLER, 2010).

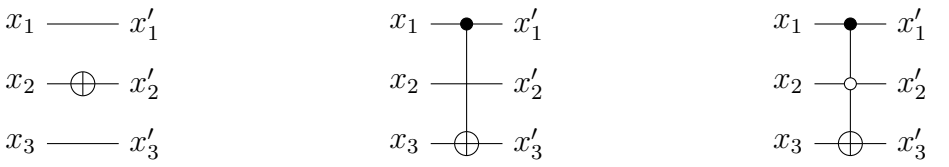
Fan-out, também conhecido como fator de carga, é definida pela quantidade máxima de entradas de portas que uma saída pode alimentar. Por exemplo, uma porta lógica com *fan-out* pode alimentar até 3 portas lógicas. Realimentação é quando a saída é utilizada como a própria entrada da porta lógica.

Na seção 2.3.1 descreve-se um método que tem como objetivo gerar a tabela verdade da função a partir do circuito.

2.3.1 Geração da tabela verdade de um circuito reversível

Para descrever o método que gera a tabela verdade de um circuito reversível, na Figura 5 apresentam-se três exemplos de circuitos que são nomeados de acordo com as posições do controle e o alvo. Primeiro são descritas as variáveis que contêm os controles e em seguida o alvo. Por exemplo, na Figura 6(c) é apresentada uma porta Toffoli com controle positivo em x_1, controle negativo em x_2 e o alvo na variável x_3; portanto, o nome é dado por $(x_1, \bar{x_2}; x_3)$.

Figura 6 – Exemplos de circuitos reversíveis.



(a) $(;x_2)$. (b) $(x_1;x_3)$. (c) $(x_1,\bar{x_2};x_3)$.

Fonte: Elaborado pelo próprio autor.

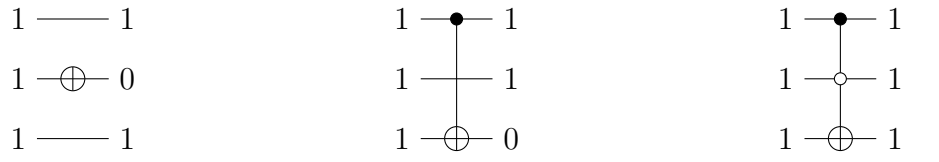
Na literatura existem dois métodos para gerar a tabela verdade. Entre eles está o método ESOP, na qual uma função é representada utilizando um conjunto de termos de produtos separados por operadores XOR (FAZEL; THORNTON; RICE, 2007). No entanto, o método escolhido para este trabalho consiste em gerar a tabela verdade utilizando as características da porta reversível, ou seja, pela configuração dos controles e o alvo (SOEKEN; THOMSEN, 2013). A seguir são listadas as especificações necessárias:

1. A saída pode resultar na negação da entrada apenas se apresentar o alvo na respectiva linha. Caso contrário, a saída é igual a entrada;
2. Se a porta reversível apresentar apenas o alvo, a saída resulta na negação da entrada automaticamente, como na porta NOT apresentada na Figura 7(a);
3. O controle positivo é ativo apenas se a variável de entrada for 1;
4. O controle negativo é ativo apenas se a variável de entrada for 0;
5. Se a porta reversível apresentar 1 ou mais controles, a função de saída resulta na negação da variável de entrada apenas se todos os controles forem ativos;
6. Se o circuito reversível apresentar mais de uma porta reversível, as m saídas da porta reversível atual se tornam as n entradas da próxima porta reversível. Esse processo é realizado até o final do circuito.

Na Figura 7 são apresentados os exemplos de circuitos utilizando o conjunto de entrada $f(7)$. Nota-se na Figura 7(c) que o conjunto de entrada não é alterado, pois a entrada x_2 apresenta valor lógico 1 e o controle é negativo.

Depois de aplicado o método para todos os conjuntos de entrada foram adquiridas as tabelas verdade, conforme apresentadas na Tabela 5.

Figura 7 – Exemplos de circuitos reversíveis utilizando o conjunto de entrada $f(7)$.



(a) $(; x_2)$. (b) $(x_1; x_3)$. (c) $(x_1, \bar{x_2}; x_3)$.

Fonte: Elaborado pelo próprio autor.

Tabela 5 – Tabelas verdade dos circuitos reversíveis da Figura

(a) (x_2).

Entrada			Saída		
x_1	x_2	x_3	x'_1	x'_2	x'_3
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	1	0	1

(b) $(x_1; x_3)$.

Entrada			Saída		
x_1	x_2	x_3	x'_1	x'_2	x'_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	1	0

(c) $(x_1, \bar{x_2}; x_3)$.

Entrada			Saída		
x_1	x_2	x_3	x'_1	x'_2	x'_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	1
1	0	1	1	0	0
1	1	0	1	1	0
1	1	1	1	1	1

Fonte: Elaborado pelo próprio autor.

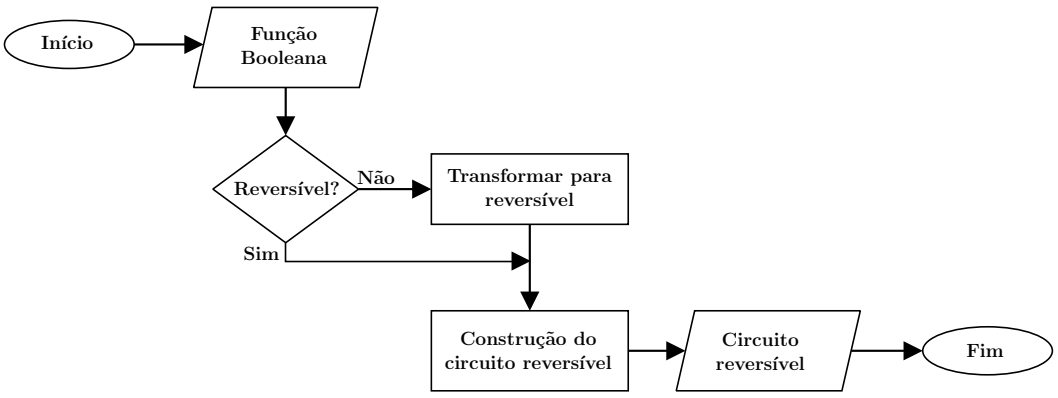
No capítulo 3 apresenta-se a síntese de circuitos reversíveis, cuja finalidade é transformar uma função Booleana irreversível para reversível e construir o circuito a partir da tabela verdade de uma função reversível.

3 SÍNTESE DE UM CIRCUITO REVERSÍVEL

Neste capítulo, apresenta-se a primeira etapa para a síntese de circuitos reversíveis. O objetivo desta etapa é transformar uma função irreversível em reversível e posteriormente construir o circuito reversível a partir da função reversível obtida, conforme é apresentado no fluxograma na Figura 8.

De acordo com Datta *et al.* (2013), a síntese é considerada a etapa mais difícil se comparada com a lógica convencional. Por exemplo, a quantidade de entradas deve ser igual a de saídas. Além disso, um circuito reversível é construído por uma cascata de portas reversíveis, ou seja, *fan-out* ou realimentação não são permitidos (DATTA *et al.*, 2013).

Figura 8 – Etapas da síntese de um circuito reversível.



Fonte: Elaborado pelo próprio autor.

Para explicar o processo da síntese de circuitos reversíveis, utiliza-se a função AND apresentada na Tabela 6 como exemplo. Nota-se que essa função é irreversível, pois a quantidade de entradas e saídas são diferentes.

Na Seção 3.1 é apresentado como transformar uma função Booleana irreversível em reversível e na Seção 3.2 é apresentado como construir o circuito reversível a partir da função Booleana reversível.

Tabela 6 – Tabela verdade da função AND.

Entrada		Saída
x_1	x_2	x'
0	0	0
0	1	0
1	0	0
1	1	1

Fonte: Elaborado pelo próprio autor.

3.1 TRANSFORMAR UMA FUNÇÃO BOOLEANA IRREVERSÍVEL EM REVERSÍVEL

A transformação de uma função Booleana irreversível em reversível é muito importante e significante, pois influencia diretamente na qualidade do circuito. A transformação divide-se em duas etapas: expansão da função e atribuição de bits nos Don't Cares (DCs), conforme descrito a seguir:

1) Expansão da função:

A expansão de uma função é fundamental nesse processo para que seja possível gerar novos conjuntos de saída e, consequentemente, tornar a função bijetiva. Assim, para expandir uma função são adicionadas saídas *garbages* e, se necessário, entradas *constant*.

Saída *garbage (s)* é uma saída adicionada na função para possibilitar que gere todas as condições de entrada. Após acrescentar saídas *garbages*, gera-se lacunas na coluna saída da tabela verdade que são preenchidas com DCs (WILLE; DRECHSLER, 2010).

Entrada *constant (e)* é uma entrada acrescentada na função para igualar a quantidade de entradas e saídas. Após adicionar entradas *constant*, gera-se lacunas na coluna entrada da tabela verdade que são preenchidas com os valores lógicos 0 ou 1. A coluna entrada deve conter as 2^n combinações possíveis, podendo estar em ordem crescente ou não (WILLE; DRECHSLER, 2010).

De acordo com Maslov e Dueck (2004), devem ser adicionadas pelo menos $\lceil log_2(\mu) \rceil$ saídas *garbages* para que seja possível gerar os conjuntos de saída necessários, sendo μ o número máximo de repetições dos conjuntos de saída na tabela verdade. Por exemplo, considerando a Tabela 6, devem ser adicionadas $\lceil log_2(3) \rceil = 2$ saídas *garbages*, pois a função AND é constituída pelo conjunto de saída 0 por três vezes repetidas.

A partir dessa definição, pode-se atribuir bits nos DCs, uma vez que quanto maior a quantidade de saídas *garbages* maior a probabilidade de aumentar a quantidade de

entradas e, consequentemente, o número de linhas na tabela verdade.

O próximo passo é verificar se a função respeita à primeira definição de reversibilidade, isto é, a equivalência entre a quantidade de entradas e saídas. Dessa forma, Maslov e Dueck (2004) afirmaram que a quantidade de entradas *constant* é calculada por:

$$\text{Entrada } \textit{constant(e)} + \text{Entrada}\textit{(n)} = \text{Saída}\textit{(m)} + \text{Saída } \textit{garbage(s)} \tag{1}$$

Portanto, para igualar a quantidade de entradas e saídas da função AND é necessário que $e + 2 = 1 + 2$, ou seja, $e = 1$. Como resultado, tem-se a tabela verdade da função AND apresentada na Tabela 7 com uma entrada *constant* e duas saídas *garbages*.

Nota-se que a tabela verdade está incompleta, pois contém DCs devido a inclusão de entrada *constant* e saídas *garbages*. Para finalizar esse processo é necessário efetuar a atribuição nos DCs.

Tabela 7 – Tabela verdade da função AND depois de adicionada uma entrada *constant* e duas saídas *garbages*.

Entrada			Saída		
e_1	x_1	x_2	x'_1	s_1	s_2
0	0	0	0	-	-
0	0	1	0	-	-
0	1	0	0	-	-
0	1	1	1	-	-
1	0	0	-	-	-
1	0	1	-	-	-
1	1	0	-	-	-
1	1	1	-	-	-

Fonte: Elaborado pelo próprio autor.

2) Atribuição nos Don't Cares:

Nesta fase atribui-se os bits nos DCs, influenciando diretamente na qualidade do circuito.

Para demonstrar a complexidade da atribuição nos DCs, considera-se a tabela verdade apresentada na Figura 9. Na parte superior da tabela ($2^n/2 = 4$ primeiras linhas) é possível atribuir os valores lógicos de '4 x 3 x 2 x 4' maneiras, totalizando-se em 96 possibilidades. Além disso, a parte inferior da tabela ($2^n/2 = 4$ últimas linhas) pode ser preenchidas por 4! maneiras (2^2 linhas), porém, também deve-se considerar que as saídas podem ser permutadas em 3! maneiras (3 saídas), totalizando 144 possibilidades. Por fim, multiplicando os campos superior e inferior da tabela verdade, são 13.824 possibilidades

de preencher a tabela verdade de uma função com apenas 3 entradas.

Figura 9 – Demonstração parcial da complexidade de preencher a tabela verdade de uma função com apenas 3 entradas.

Entrada			Saída			
e_1	x_1	x_2	x'_1	s_1	s_2	
0	0	0	0	-	-	$\rightarrow 2^2 = 4$
0	0	1	0	-	-	$\rightarrow 2^2$ - 1 = 3
0	1	0	0	-	-	$\rightarrow 2^2$ - 2 = 2
0	1	1	1	-	-	$\rightarrow 2^2 = 4$
1	0	0	-	-	-	
1	0	1	-	-	-	
1	1	0	-	-	-	
1	1	1	-	-	-	

Fonte: Elaborado pelo próprio autor.

Com intuito de resolver esse problema, foram desenvolvidos métodos para atribuir os bits nos DCs na tentativa de gerar circuitos menores como, por exemplo, os algoritmos *Greedy* e *Hungarian* apresentados por Miller, Wille e Dueck (2009).

Neste trabalho utilizou-se o método XOR como exemplo. Esse método foi apresentado por Maslov e Dueck (2004) e tem como funcionalidade preencher os DCs efetuando a operação XOR (*Exclusive-OR*) junto às variáveis de entrada. A seguir são apresentados os passos do método XOR:

1. Para cada linha da tabela verdade i completa:

   a. Conjunto $k = i$, então k representa o conjunto de entrada;

   b. Conjunto p e variável auxiliar $q = 0$;

   c. Para cada saída f_j da função ($0 \leq j < n$):

      i. Se f_j é uma saída garbage, $q = \text{q} \oplus k_j$ e $p_j = \text{q}$;

      ii. Caso contrário, $p_j = f_i$.

   d. Se p for igual a qualquer outro conjunto de saída f, incremente k e repita o passo 1.b;

   e. O conjunto de saída p é colocado na tabela verdade.

Considerando o conjunto de entradas $f(0)$ da função AND apresentada na Tabela 6, a seguir é listado detalhadamente o processo de execução do método XOR:

Passo 1 - Identificar as posições dos vetores de entrada e saída. Na Tabela 8(a),

apresentam-se as posições da linha 0.

Passo 2 - Efetuar os cálculos conforme descritos a seguir:

I - Dados iniciais: $k = 0$; $q = 0$.

II.a - Posição atual (p_3) - Saída *garbage*:

$p_3 = q \oplus k_3$;

$p_3 = 0 \oplus 0 = 0$.

II.b - Posição atual (p_2) - Saída *garbage*:

$p_2 = q \oplus k_2$;

$p_2 = 0 \oplus 0 = 0$.

II.c - Posição atual (p_1) - Não é saída *garbage*;

$p_1 = f_1$;

II.d - O conjunto p é definido por 000.

Passo 3. Verificar se existe um conjunto f igual ao conjunto p. Se o conjunto p for igual a qualquer conjunto de saída f é necessário incrementar k e calcular p novamente. Caso contrário, deve-se passar p para f_i na tabela verdade e prosseguir para a próxima linha. Na Tabela 8(b), apresenta-se o conjunto de saída de $f(0)$ depois de aplicado o método XOR.

Tabela 8 – Exemplo de aplicação do método XOR referente ao conjunto de entrada $f(0)$ da função reversível AND.

(a) Posições.

Entrada			Saída		
e_1	x_1	x_2	x_1'	s_1	s_2
k_1	k_2	k_3	f_1	f_2	f_3

(b) Valores atribuídos.

Entrada			Saída		
e_1	x_1	x_2	x_1'	s_1	s_2
0	0	0	0	0	0

Fonte: Elaborado pelo próprio autor.

Apresenta-se na Tabela 9 a tabela verdade completa da função reversível AND após efetuado o processo de atribuição de bits nos DCs. Nota-se que os valores iniciais permanecem na tabela verdade, conforme destacados em vermelho.

Nesta seção, apresentou-se uma metodologia para transformar uma função Booleana irreversível para reversível. Na Seção 3.2, apresenta-se um procedimento para obter o circuito reverísvel a partir da tabela verdade de uma função reversível.

Tabela 9 – Tabela verdade da função AND utilizando o método *XOR*.

Entrada			Saída		
e_1	x_1	x_2	x_1'	s_1	s_2
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	1	0
0	1	1	1	0	1
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	1

Fonte: Elaborado pelo próprio autor.

3.2 OBTENÇÃO DE CIRCUITO REVERSÍVEL A PARTIR DA TABELA VERDADE

Depois de transformar uma função irreversível para reversível é possível efetuar a construção do circuito reversível a partir da tabela verdade. Entre vários métodos existentes na literatura para a construção de um circuito reversível, como apresentados por Datta *et al.* (2013) e Zeng *et al.* (2015), neste trabalho foi implementado o método *Transformation Based Synthesis*, também conhecido por método MDM.

O método MDM, apresentado por Miller, Maslov e Dueck (2003), consiste em adicionar uma porta reversível ao circuito, com apenas controles positivos, até que se torne uma função identidade. Obtém-se uma função identidade quando cada conjunto de entradas e saídas são iguais. Por exemplo, o conjunto de entradas $f(0) = f'(0)$, $f(1) = f'(1)$ e assim por diante.

Vale ressaltar que as portas são escolhidas de modo que não altere as linhas anteriores, iniciando-se a análise pela primeira linha.

Outro fator importante é que as portas devem ser adicionadas no início do circuito, ou seja, depois de adicionada a primeira porta deve-se acrescentar a próxima no lado esquerdo da última porta adicionada no circuito.

A seguir são apresentadas as etapas do método MDM:

1. Se $f(0) \neq 0$, inverte-se o valor lógico da saída que apresenta valor lógico diferente da entrada. A cada inversão adiciona-se a porta NOT na linha da saída em que apresenta valor diferente. Esse passo deve ser realizado até que $f(0) = 0$. Adiciona-se a porta NOT no início do circuito;

2. Considera-se cada i entre $1 \leq i < 2^{\frac{n}{2}}$ - 1. Se $f(i) =$ i, não é necessário modificar

o conjunto. Por outro lado, se $f(i) \neq i$, adiciona-se uma porta reversível ao circuito considerando a variável que deseja alterar como alvo e a variável com bit 1 como controle positivo; no entanto, considera-se apenas uma variável, de preferência a que apresenta um índice maior que o alvo. Adiciona-se a porta CNOT no início do circuito;

3. Considera-se cada i entre $2^{\frac{n}{2}}$ - 1 $\leq i \leq 2^n$ - 1. Se $f(i) =$ i, não é necessário modificar o conjunto. Por outro lado, se $f(i) \neq i$, adiciona-se uma porta reversível ao circuito considerando todas as variáveis com bit 1 como controle e a variável que deseja alterar como alvo. Adiciona-se a porta Toffoli no início do circuito.

Como exemplo, considera-se a função descrita na Tabela 9 para aplicar o método MDM, em que foram necessárias duas portas para encontrar uma função identidade. Para melhor entendimento, as variáveis de entrada e as funções de saída são representadas, respectivamente, por a, b e c. Por exemplo, e_1 e x_1' são representados por a. Além disso, apresenta-se uma coluna nomeada de Linha para facilitar a visualização da tabela verdade. Nota-se que a numeração da linha também representa o valor do conjunto f. Como, por exemplo, a linha 0 representa $f(0)$.

Na coluna Passo 1 da Tabela 10, o primeiro conjunto alterado é $f(1) = f'(3)$, pois o valor lógico b na linha 1, da coluna Passo 1 e da entrada, são diferentes ($0\mathbf{0}1 \neq 0\mathbf{1}1$). Portanto, adiciona-se uma porta CNOT com controle positivo e o alvo, respectivamente, na variável representada por c e b, conforme apresenta-se na Figura 10(a).

Nota-se que a cada linha da tabela o conjunto de saída é comparado com o conjunto de entrada até que se torne uma função identidade.

Tabela 10 – Processo de aplicação do método MDM na função AND.

Linha	Entrada			Saída			Passo 1			Passo 2			Saída		
(i)	a	b	c	a	b	c	a	b	c	a	b	c	a	b	c
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	1	0	$\boxed{1}$	1	0	0	1	0	0	1
2	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
3	0	1	1	1	0	1	1	0	1	$\boxed{1}$	1	1	0	1	1
4	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
5	1	0	1	1	1	1	1	1	1	1	0	1	1	0	1
6	1	1	0	1	1	0	1	1	0	1	1	0	1	1	0
7	1	1	1	0	0	1	0	0	1	0	1	1	1	1	1

Fonte: Elaborado pelo próprio autor.

O segundo conjunto alterado apresentado na coluna Passo 2 é $f(3) = f'(7)$, pois o valor lógico a na linha 3, da coluna Passo 2 e da entrada, são diferentes ($\mathbf{0}11 \neq \mathbf{1}11$). Portanto, foi adicionada uma porta Toffoli com alvo na variável representada por a e os

controles positivos nas variáveis representadas por b e c, conforme apresenta-se na Figura 10(b). Cabe salientar que a porta deve ser adicionada no início do circuito, ou seja, no lado esquerdo da última porta adicionada no circuito.

Figura 10 – Processo de construção do circuito reversível da função AND.

(a) Passo 1. (b) Passo 2.

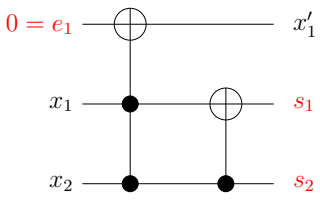
Fonte: Elaborado pelo próprio autor.

Como resultado da aplicação do método MDM, apresenta-se na Figura 11 o circuito reversível da função AND com duas portas. Observa-se que as variáveis de entrada e as funções de saída estão representadas de acordo com as definições iniciais, isto é, e_1, x_1, etc.

Cabe salientar que a entrada *constant* e as saídas *garbages* estão destacadas em vermelho, pois apenas as variáveis de entrada x_1 e x_2 e a função de saída x'_1 estão presentes na função AND original, conforme apresentada na Tabela 6.

Observa-se também que a entrada *constant* foi definida pelo valor lógico 0, pois a função original se localiza nas $\frac{2^n}{2}$ ($\frac{2^3}{2} = 4$) primeiras linhas da tabela verdade, conforme apresentada na Tabela 9.

Figura 11 – Circuito reversível da função AND.



Fonte: Elaborado pelo próprio autor.

Neste capítulo foi apresentada a etapa da síntese de um circuito reversível, utilizando como exemplo os métodos XOR e MDM aplicados na função AND, originalmente irreversível. No Capítulo 4, aborda-se a otimização pós-síntese na lógica reversível, cuja finalidade é reduzir a quantidade de portas no circuito reversível.

4 OTIMIZAÇÃO PÓS-SÍNTESE DE UM CIRCUITO REVERSÍVEL

No Capítulo 3 foi apresentada a etapa da síntese de circuitos reversíveis, isto é, como transformar uma função irreversível para reversível e posteriormente como construir um circuito reversível. No entanto, apesar de muitas pesquisas efetuadas sobre esse assunto, não foi desenvolvido um método capaz de encontrar o ótimo global, isto é, gerar um circuito reversível com menor custo.

Na otimização pós-síntese é possível reduzir o custo de um circuito reversível que pode ser quantificado por custo quântico ou pela quantidade de portas utilizadas no circuito. Conforme apresentado na literatura, existem vários estudos relacionados à otimização de circuitos reversíveis. Por exemplo, Rahman, Soeken e Dueck (2015) que abordaram a heurística *Dynamic Template* para redução da quantidade de portas no circuito e Abdessaied *et al.* (2015) que implementaram um método baseado na meta-heurística *Simulated Annealing* para otimizar o custo quântico.

A metodologia deste trabalho foi estudar algoritmos para reduzir a quantidade de portas no circuito. Foram desenvolvidos métodos de otimização que utilizam diferentes métodos heurísticos junto às regras de reescrita. Vale ressaltar que foi utilizada tanto a heurística como a meta-heurística. Heurística é um método para se obter resultados ótimos em um tempo razoável, enquanto a meta-heurística utiliza estratégias para escapar de ótimos locais na procura do ótimo global.

Além dos métodos de otimização, desenvolveu-se um método denominado Divisão, que é adicionado ao processo de otimização para diminuir o espaço de busca. A estratégia é dividir o circuito em várias vizinhanças e fazer uma busca pelo ótimo local em cada uma.

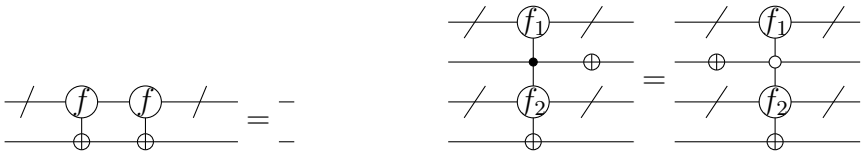
Na Seção 4.1 são apresentadas as regras de reescrita utilizadas nos métodos de otimização. Na Seção 4.2, apresentam-se os métodos de otimização que aplicam a heurística *Greedy* e meta-heurísticas *Simulated Annealing* e *Variable Neighbourhood Descent*. Por fim, apresenta-se o método Divisão.

4.1 REGRAS DE REESCRITA

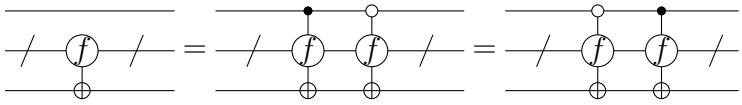
Segundo Soeken e Thomsen (2013), as regras de reescrita são utilizadas para moldar o circuito de diferentes maneiras por meio de movimentos simples. Foram definidas 7 regras de reescrita que podem alterar o custo do circuito e o mais importante, sem alterar a tabela verdade. Entre essas regras encontram-se as que aumentam, as que diminuem e as que simplesmente modificam o circuito, conforme são apresentadas nas Figuras 12(a) a 12(f).

Para a apresentação das regras de reescrita, utilizou-se f para representar o conjunto de controles apresentados no circuito. Por exemplo, na Regra D1, f pode ser representado tanto por um controle negativo quanto por um controle positivo.

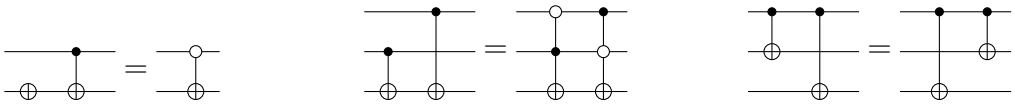
Figura 12 – Regras de reescrita.



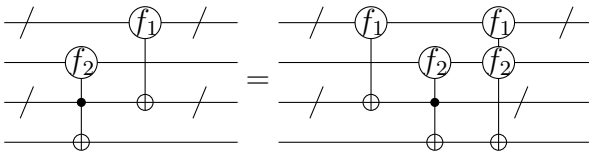
(a) Regra D1. (b) Regra D2.



(c) Regra D3.



(d) Regra D4. (e) Regra D5. (f) Regra D6.



(g) Regra D7.

Fonte: Adaptado de (SOEKEN; THOMSEN, 2013).

Na Regra D1 é possível verificar que duas portas adjacentes idênticas podem ser

removidas. Nas Regras D2, D5 e D6, nota-se que o custo dos circuitos não são alterados, no entanto, apresentam diferentes características, isto é, os controle positivos e negativos. Por exemplo, na Regra D2 as portas reversíveis trocam de posição e na variável que contém o alvo, altera-se o controle para que seja possível efetuar a troca de posição sem alterar a tabela verdade. Nas Regras D3, D4 e D7, verifica-se que é possível aumentar e diminuir o custo de um circuito. No entanto, neste trabalho apenas a regra D7 é aplicada para diminuir ou aumentar o custo. Ou seja, as regras D3 e D4 apenas reduz o custo.

4.2 MÉTODOS DE OTIMIZAÇÃO

Os métodos de otimização foram desenvolvidos para reduzir o custo de circuitos reversíveis utilizando metodologias baseadas em métodos heurísticos. Esses métodos utilizam as regras de reescrita apresentadas na Seção 4.1 como estrutura para modificar o circuito.

4.2.1 Método *Greedy*

Para algumas soluções, a heurística *Greedy* pode encontrar soluções ótimas. Por outro lado, para problemas pertencentes a classe NP-completo ou NP-difícil, essa metodologia tenta obter resultados próximos ao ótimo em tempo razoável. A grande desvantagem é a dificuldade de escapar de ótimos locais (GENDREAU; POTVIN, 2010).

Na otimização de circuitos reversíveis, o método *Greedy* consiste em otimizar o circuito em um tempo razoável. A cada iteração é aplicada uma regra de reescrita em duas portas sequencialmente, até percorrer todo o circuito. Isto é, a cada duas portas são listadas todas as regras possíveis de serem aplicadas e selecionado apenas uma das regras, isto é, se apresentar custo menor ou igual. As regras são escolhidas na seguinte ordem: D1 $\rightarrow$ D2 $\rightarrow$ D3 $\rightarrow$ D4 $\rightarrow$ D5 $\rightarrow$ D6 $\rightarrow$ 7.

A regra é aceita apenas se $\Delta_{custo} \leq 0$, conforme é apresentada na Equação 2. A solução atual (S) representa as duas portas reversíveis e a solução vizinha (S'), a solução atual depois de aplicada uma das regras.

$$\Delta_{custo} = S' - S \tag{2}$$

Depois de percorrer o circuito inteiro é verificado se houve otimização. Caso apresente redução no custo do circuito, o método *Greedy* é aplicado novamente.

4.2.2 Método *Simulated Annealing*

A meta-heurística *Simulated Annealing* é uma das técnicas para resolver o problema de métodos heurísticos simples, pois possuem características que possibilitam escapar de ótimos locais (GENDREAU; POTVIN, 2010).

Na otimização de circuitos reversíveis, realizam-se l iterações em cada camada de temperatura. É importante afirmar que a variável temperatura é considerada apenas como um parâmetro para o desenvolvimento do método, ou seja, o valor é adimensional. A cada iteração tem-se duas soluções, a atual (S) e vizinha (S'):

- solução atual: circuito atual;
- solução vizinha: solução atual modificada, isto é, depois de aplicada uma das regras de reescrita no circuito.

Para gerar a solução vizinha, são listadas todas as regras possíveis e sorteada apenas uma. Se $\Delta_{custo} \leq 0$, a solução S' é aceita. Por outro lado, se $\Delta_{custo} > 0$ a solução vizinha ainda pode ser aceita, podendo assim evitar uma convergência prematura.

A estratégia da *Simulated Annealing* é aceitar soluções piores no início da execução devido a temperatura elevada (T), que é definida pela probabilidade apresentada na Equação 3. No entanto, conforme a temperatura diminui, apenas as soluções melhores são aceitas.

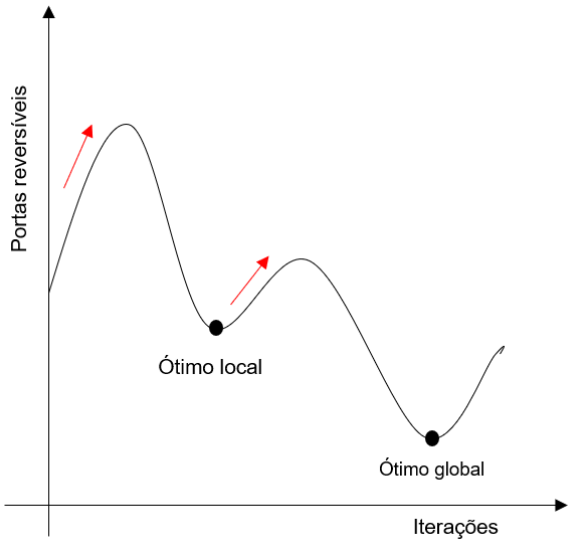
$$p(\Delta_{custo}) = e^{-\Delta_{custo}/T} \tag{3}$$

Na Figura 13 apresenta-se um gráfico da análise do comportamento da *Simulated Annealing* na otimização de circuitos reversíveis. Nota-se que conforme executam-se as iterações, a quantidade de portas no circuito aumenta e diminui para escapar do ótimo local e tentar atingir o ótimo global.

Outro fator importante aplicado no método SA é a variável *Limite*. Esse parâmetro foi atribuído ao método para definir a quantidade máxima de portas que o circuito pode alcançar, evitando assim que cresça de uma maneira que não possa ser otimizado posteriormente.

Para demonstrar a importância da variável *Limite*, na Figura 14 é apresentado o desempenho em ambos os casos, sem e com a variável *Limite*. Na Figura 14(a) é

Figura 13 – Análise do comportamento da *Simulated Annealing* na otimização de circuitos reversíveis.



Fonte: Elaborado pelo próprio autor.

exibido o comportamento sem a variável *Limite*, sendo possível observar o crescimento descontrolado do circuito e a incapacidade de reduzir a quantidade de portas no decorrer do processo. Por outro lado, na Figura 14(b) é apresentada a efetividade da variável *Limite* no método, tendo como resultado uma redução de 5 portas reversíveis, equivalente a aproximadamente 24% do custo inicial do circuito.

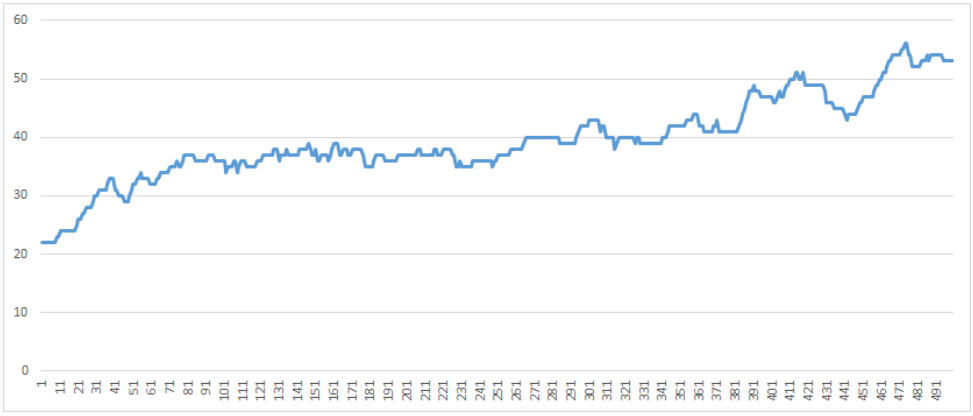
Apresentada a estrutura da meta-heurística na otimização de circuitos reversíveis, na Tabela 11 são apresentadas as variáveis e funções utilizadas no pseudocódigo do método SA apresentado na Figura 15.

Tabela 11 – Variáveis e funções utilizadas no método SA e as respectivas aplicações.

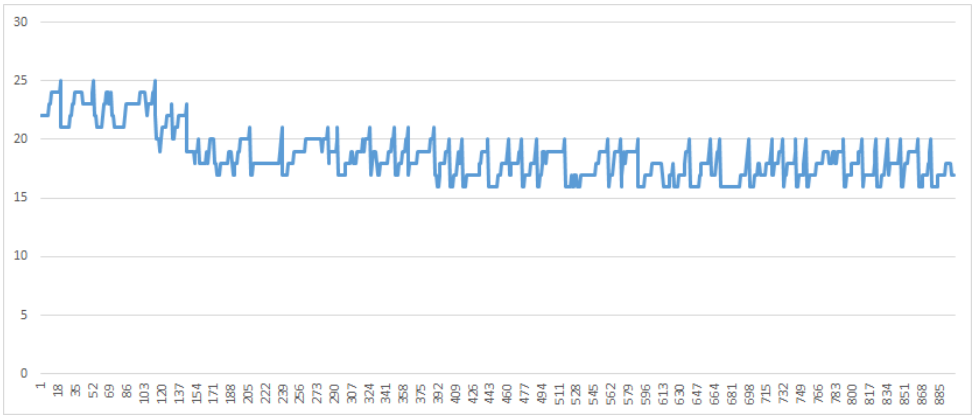
Variável	**Aplicação**
T	Temperatura
l	Quantidade de iterações na camada de tempo.
$Parada$	Critério que finaliza o algoritmo.
$Limite$	Quantidade máxima de portas que o circuito pode alcançar.
x	Regras de reescritas que podem ser aplicadas.
α	Fator de redução da temperatura.
Funções	**Aplicação**
Aplica_regra	Lista todas as x regras possíveis e sorteia apenas uma.

Fonte: Elaborado pelo próprio autor.

Figura 14 – Análise da variável *Limite* no método *Simulated Annealing* aplicado no circuito mod5adder_127.



(a) Sem limite.



(b) Com limite.

Fonte: Elaborado pelo próprio autor.

4.2.3 Método *Variable Neighbourhood Descent*

O método *Variable Neighbourhood Descent* obtém um ótimo local para uma certa estrutura de vizinhança, porém não necessariamente é um ótimo local para outra estrutura de vizinhança. Assim, a funcionalidade desse método é atingir o mínimo local a cada vizinhança e quando alcançar, passar para a próxima vizinhança (POSSAGNOLO, 2015).

Para entendimento da metodologia do VND, na Figura 16 é apresentado o processo de estrutura. Na primeira vizinhança $N_1(x_1)$ foi encontrada uma solução melhor $N_1(x_2)$. Ainda na mesma vizinhança, obteve-se várias outras soluções, como por exemplo, a solução x_3; no entanto, nenhuma solução melhor. Sendo assim, avança-se para a próxima vizinhança $N_2(x_2)$, em que foi encontrada uma solução melhor $N_2(x_4)$. Por fim, como foi

Figura 15 – Pseudocódigo do método *Simulated Annealing*.

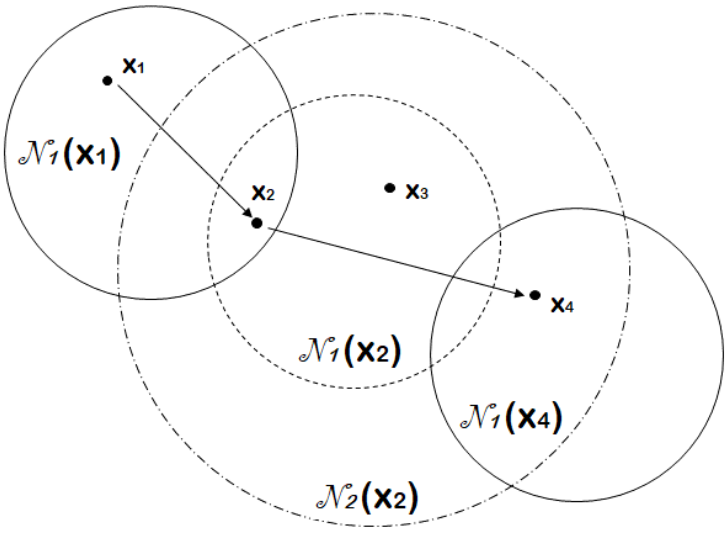
```
1  procedimento Método Simulated Annealing(circ);
2  //S = Circuito atual;
3  //S' = Circuito auxiliar;
4  //M = Circuito mínimo;
5  //*S = Quantidade de portas no circuito atual;
6  //*S' = Quantidade de portas no circuito auxiliar;
7  //*M = Quantidade de portas no circuito mínimo;
8  início
9      enquanto Parada < 5 faça
10         Otimizado ← falso;
11         para i = 1 at l faça
12             se S > Limite então
13                 S → M;
14             fim
15             Aplica_regra(S,S');
16             se Δ_custo ≤ 0 então
17                 S' → S;
18             fim
19             senão
20                 q ← Aleatório(0,1);
21                 se q < e^(−Δ_custo/T) então
22                     S' → S;
23                 fim
24             fim
25             se S < M então
26                 S → M;
27                 Otimizado ← verdadeiro;
28             fim
29         fim
30         T ← T.α;
31         se !Otimizado então
32             Parada++;
33         fim
34         senão
35             Parada ← 0;
36         fim
37     fim
38 fim
```

Fonte: Adaptado de (ABDESSAIED *et al.*, 2015).

obtida uma nova solução, retorna-se para a primeira estrutura de vizinhança $N_1(x_4)$.

Na otimização de circuitos reversíveis, a estrutura de vizinhança é definida pela quantidade de regras de reescrita que são aplicadas na solução atual (S). Por exemplo, na primeira vizinhança (N_1) é aplicada uma regra no circuito, na segunda (N_2) são aplicadas

Figura 16 – Estrutura da meta-heurística *Variable Neighbourhood Descent*.



Fonte: Adaptado de (POSSAGNOLO, 2015).

duas regras e assim sucessivamente.

Como no método SA, são realizadas l iterações para cada estrutura de vizinhança. A cada iteração tem-se duas soluções: a solução atual (S) e a solução vizinha (S'). A solução atual representa o circuito atual e a solução vizinha, o circuito modificado devido a aplicação da regra. No entanto, a solução vizinha é aceita apenas se $\Delta_{custo} \leq 0$, conforme apresenta-se no Algoritmo 17.

Na Tabela 12 são apresentadas as variáveis e funções utilizadas no pseudocódigo do método VND, conforme apresenta-se na Figura 17.

Tabela 12 – Variáveis e funções utilizadas no método *Variable Neighbourhood Descent* e as respectivas aplicações.

Variável	**Aplicação**
l	Quantidade de iterações na camada de tempo.
N	Vizinhança máxima.
v	Vizinhança atual
x	Regras de reescrita que podem ser aplicadas.
Funções	**Aplicação**
Aplica_regra	Lista todas as x regras possíveis e sorteia apenas uma.

Fonte: Elaborado pelo próprio autor.

Figura 17 – Pseudocódigo do método *Variable Neighbourhood Search*.

```
1  procedimento Método Variable Neighbourhood Descent(circ);
2  //S = Circuito atual;
3  //S' = Circuito auxiliar;
4  //*S = Quantidade de portas no circuito atual;
5  //v ← 1;
6  início
7      enquanto v ≤ N faça
8          Otimizado ← falso;
9          S' → S
10         s para i = 1 até l faça
11             S' → S
12             Aplica_regra(S,S');
13         fim
14         se S' ≤ S então
15             S' → S
16             Otimizado ← verdadeiro;
17         fim
18         se Otimizado então
19             v ← 1;
20         fim
21         senão
22             v++;
23         fim
24     fim
25 fim
```

Fonte: Elaborado pelo próprio autor.

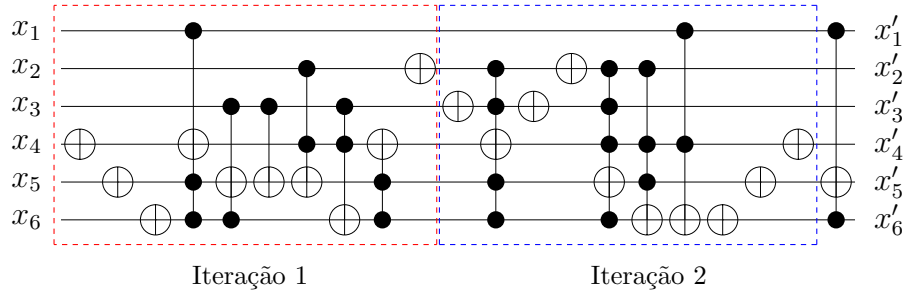
4.3 MÉTODO DIVISÃO

Neste trabalho desenvolveu-se o método Divisão para solucionar o problema dos métodos SA e VND, que perdem a efetividade conforme o aumento na quantidade de portas iniciais no circuito. Esse método tem a funcionalidade de reduzir o espaço de busca que é aplicado no método de otimização, isto é, dividir o circuito em várias vizinhanças e realizar uma busca pelo ótimo local em cada uma delas.

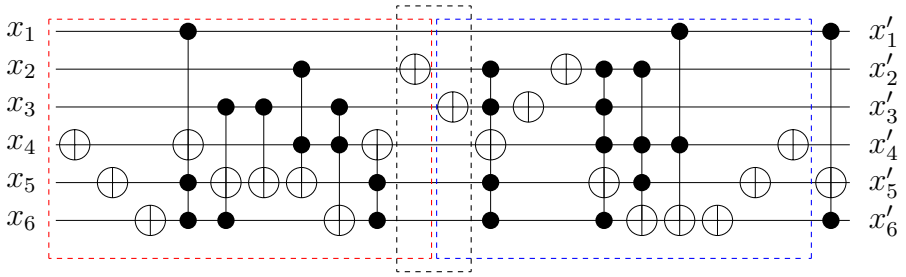
Como exemplo de aplicação, apresenta-se na Figura 18(a) o método Divisão aplicado no circuito denominado mod5adder_127, constituído por 21 portas e 6 variáveis. Considerando um espaço de busca de 10 portas, o método de otimização aplicado a cada 10 portas, conforme destacado nas iterações 1 e 2 em vermelho e azul, respectivamente. No entanto, notou-se que portas adjacentes que poderiam ser otimizadas são separadas

nesse processo e impossibilitadas de otimização, conforme destacado na Figura 18(b).

Figura 18 – Estudo do algoritmo Divisão utilizando o circuito reversível mod5adder_127.



(a) Primeiro teste.



(b) Deficiência identificada.

Fonte: Elaborado pelo próprio autor.

Como solução para esse problema, a partir da segunda iteração são reutilizadas portas do circuito otimizado na iteração anterior. Essa variável é denominada de "*Reut*". O parâmetro "*Reut*" define a quantidade de portas que são reutilizadas. Sendo assim, as portas não reutilizadas são adicionadas no circuito final até que seja efetuado todo o processo.

Por exemplo, na Figura 19 é apresentado o processo de execução do método Divisão com os parâmetros EDB e *Reut* definidos respectivamente por 10 e 5. Na Figura 19(a) apresenta-se o circuito original.

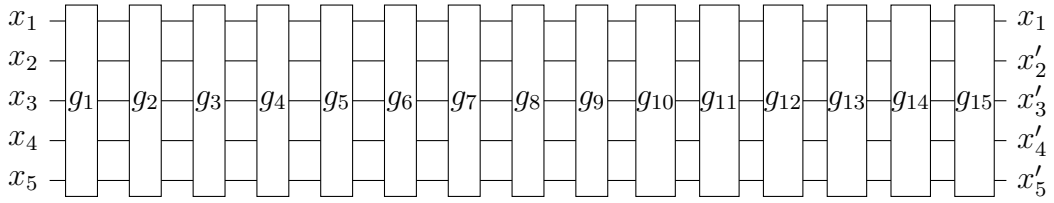
Na primeira iteração são selecionadas as portas g_1 até g_{10} do circuito original e posteriormente aplicado o método de otimização desejado. Considerando que foram reduzidas 6 portas, na Figura 19(b) apresenta-se o circuito otimizado na primeira iteração com 9 portas. Nota-se que as portas estão nomeadas de g'_1 até g'_9.

Na segunda iteração são selecionadas as 5 últimas portas do circuito apresentado na Figura 19(c), isto é, as portas g'_5 até g'_9 e as próximas (EDB - *Reut*) portas do circuito original, ou seja, as portas g_{11} até g_{15} do circuito original. Cabe salientar que as portas

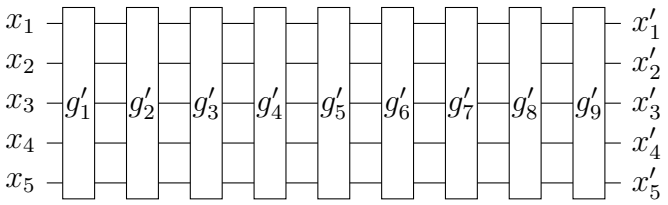
restantes são adicionadas no circuito final, conforme é apresentado na Figura 19(d).

Nota-se que todas as portas do circuito original já foram selecionadas; portanto, considerando que foram reduzidas 5 das 10 portas na segunda iteração, adicionaram-se as 5 portas restantes no circuito final, nomeadas de g_1'' até g_5''.

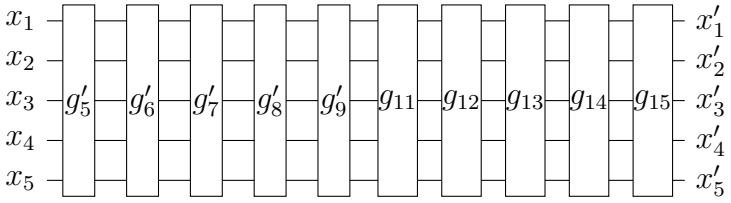
Figura 19 – Definição final do método Divisão.



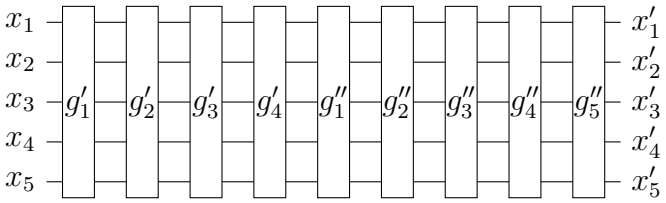
(a) Circuito original.



(b) Resultado da primeira iteração.



(c) Circuito selecionado para a segunda iteração.



(d) Circuito final.

Fonte: Elaborado pelo próprio autor.

Depois de percorrer o circuito inteiro é verificado se houve otimização. Caso apresente redução no custo do circuito, o método Divisão é aplicado novamente.

Na Tabela 13 são apresentadas as variáveis e funções do método Divisão e na Figura

20 é apresentado o pseudocódigo.

Tabela 13 – Variáveis e funções utilizadas no método Divisão e as respectivas aplicações.

Variável	**Aplicação**
EDB	Quantidade de portas em que é aplicado o método de otimização
Reut	Portas reutilizadas da iteração anterior
Funções	**Aplicação**
AdicionaPortas($circuito_1$, $circuito_2$)	Adiciona as portas do $circuito_1$ para o final do $circuito_2$.
LimpaPortas(*circuito*)	Deleta todas as portas do *circuito*.
CortaCircuito ($circuito_1$, pos_1, pos_2, $circuito_2$)	Realoca as portas entre as posições pos_1 e pos_2 do $circuito_1$ para o final do $circuito_2$.

Fonte: Elaborado pelo próprio autor.

Figura 20 – Pseudocódigo do método Divisão.

```
1  procedimento Método Divisão(circ, EDB, Reut);
2  //S = Circuito original;
3  //S' = Circuito auxiliar;
4  //S_o = Circuito auxiliar 2;
5  //S_f = Circuito final;
6  //*S = Quantidade de portas reversíveis do circuito original;
7  //*S' = Quantidade de portas reversíveis do circuito auxiliar;
8  //*S_o = Quantidade de portas reversíveis do circuito auxiliar 2;
9  //início = Início do circuito;
10 //fim = Fim do circuito;
11 término ← falso;
12 S_o ← 0;
13 S_f ← 0;
14 início ← 0;
15 fim ← EDB;
16 enquanto !termino faça
17 |  se ( *S == fim ) então
18 |  |  término ← verdadeiro;
19 |  fim
20 |  AdicionaPortas(S , S_o);
21 |  LimpaCircuito(S');
22 |  CortaCircuito(S, início, fim, S_o);
23 |  Algoritmo Simulated Annealing(S_o);
24 |  CortaCircuito(otimizado, *S_o - reut, *S_o, S');
25 |  CortaCircuito(otimizado, 0, *S_o - 1, S_f);
26 |  LimpaCircuito(otimizado);
27 |  início ← inicio + 1;
28 |  fim ← inicio + EDB - Reut - 1;
29 fim
30 AdicionaPortas(S', S_f);
```

Fonte: Elaborado pelo próprio autor.

Para avaliar o desempenho do método Divisão, comparou-se o método Divisão aplicado em conjunto com os dois algoritmos meta-heurísticos anteriormente descritos, utilizando-se um conjunto de 42 circuitos reversíveis. Os resultados dessas comparações são apresentados no capítulo 5.

5 RESULTADOS

Foram selecionados 42 circuitos reversíveis, conforme apresentados nas Tabelas 14 e 15, para serem usados como base de teste. Esses circuitos também foram utilizados por Rahman, Soeken e Dueck (2015) e Almeida, Dueck e Silva (2018). Cabe salientar que os circuitos foram colocados conforme a quantidade de portas, pois verificou-se que a qualidade dos métodos varia conforme a quantidade de portas no circuito inicial.

Tabela 14 – Dimensão dos 42 circuitos reversíveis selecionados para teste.

Circuito	Nome	Quantidade de variáveis	Quantidade de portas no circuito
1	4gt11-v1_85	5	4
2	4gt13-v1_93	5	4
3	4gt12-v1_89	5	5
4	4mod5-v0_19	5	5
5	4mod5-v1_24	5	5
6	mod5mils_65	5	5
7	mod5mils_71	5	5
8	3_17_13	3	6
9	3_17_14	3	6
10	4gt4-v0_72	5	6
11	decod24-v0_38	4	6
12	alu-v2_32	5	7
13	alu-v2_33	5	7
14	mod10_176	4	7
15	mod5d1_63	5	7
16	4mod5-v1_23	5	8
17	mod8-10_178	5	9
18	4gt12-v0_87	5	10
19	4gt13_91	5	10
20	j-e11_168	4	10
21	mod10_171	4	10
22	4_49_17	4	12
23	aj-e11_165	4	13
24	alu-v2_31	5	13

Fonte: Elaborado pelo próprio autor.

Tabela 15 – Dimensão dos 42 circuitos reversíveis selecionados para teste (continuação).

Circuit0	Nome	Quantidade de variáveis	Quantidade de portas no circuito
25	mod8-10_177	5	14
26	4_49_16	4	16
27	rd53_137	7	16
28	4gt4-v0_73	5	17
29	hwb4_49	4	17
30	alu-v2_30	5	18
31	mod5adder_127	6	21
32	ham7_106	7	25
33	rd53_131	7	28
34	rd53_130	7	30
35	sym6_145	7	36
36	hwb7_62	7	331
37	hwb8_116	8	749
38	hwb9_123	9	1959
39	urf2_152	8	5030
40	urf5_158	9	10276
41	urf1_149	9	11554
42	urf3_155	10	26468

Fonte: Elaborado pelo próprio autor.

A avaliação de cada método foi realizada de acordo com a quantidade de portas no circuito final, o tempo de execução em segundos (s), a redução do custo em porcentagem e a memória RAM consumida em Megabytes (MB).

Nesta seção foram efetuadas duas análises para verificar a qualidade de cada método. Na primeira avaliação, na Seção 5.1, comparou-se os métodos *Greedy*, *Simulated Annealing* e *Variable Neighbourhood Descent*. E em seguida, na Seção 5.2, analisou-se os métodos SA e VND combinados ao método Divisão em relação aos algoritmos que aplicam a heurística *Dynamic Template* (DT) (RAHMAN; SOEKEN; DUECK, 2015) e a meta-heurística *Tabu Search* (TS) (ALMEIDA; DUECK; SILVA, 2018).

5.1 PRIMEIRA AVALIAÇÃO

Na primeira avaliação são apresentados os resultados gerados com a aplicação dos métodos *Greedy*, *Simulated Annealing* e *Variable Neighbourhood Descent*. Cabe salientar que nos métodos SA e VND foram efetuados testes com e sem o uso do método Divisão. Posteriormente é efetuada uma análise final dos métodos de acordo com os resultados gerados.

5.1.1 Método *Greedy*

O método *Greedy* visa aceitar apenas soluções iguais ou melhores durante o processo, portanto, não utilizam as regras de reescrita que aumentam o respectivo custo. Cabe salientar que a estrutura aplicada no método *Greedy* é diferente dos métodos SA e VND, pois as regras são aplicadas a cada duas portas do início ao fim do circuito. Devido a isso, o método *Greedy* não é combinado com o método Divisão.

Nas Tabelas 16 e 17 são apresentados os resultados depois de aplicado o método *Greedy* nos 42 circuitos reversíveis.

Tabela 16 – Quadro de resultado do método *Greedy*.

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
1	4gt11-v1_85	4	4	0%	2 x 10^{-5}	1,17
2	4gt13-v1_93	4	3	25%	4 x 10^{-5}	1,18
3	4gt12-v1_89	5	5	0%	3 x 10^{-5}	1,33
4	4mod5-v0_19	5	3	40%	5 x 10^{-5}	1,23
5	4mod5-v1_24	5	5	0%	3 x 10^{-5}	1,20
6	mod5mils_65	5	3	40%	1 x 10^{-4}	1,15
7	mod5mils_71	5	3	40%	1 x 10^{-4}	1,14
8	3_17_13	6	4	33,3%	1 x 10^{-4}	1,16
9	3_17_14	6	5	16,7%	1 x 10^{-4}	1,39
10	4gt4-v0_72	6	5	16,7%	1 x 10^{-4}	1,23
11	decod24-v0_38	6	6	0%	4 x 10^{-5}	1,25
12	alu-v2_32	7	6	14,3%	1 x 10^{-4}	1,30
13	alu-v2_33	7	6	14,3%	1 x 10^{-4}	1,26
14	mod10_176	7	6	14,3%	1 x 10^{-4}	1,29
15	mod5d1_63	7	6	14,3%	1 x 10^{-4}	1,20
16	4mod5-v1_23	8	8	0%	1 x 10^{-4}	1,29
17	mod8-10_178	9	8	11,1%	1 x 10^{-4}	1,33
18	4gt12-v0_87	10	9	10%	1 x 10^{-4}	1,18
19	4gt13_91	10	9	10%	1 x 10^{-4}	1,18
20	aj-e11_168	10	9	10%	1 x 10^{-4}	1,66
21	mod10_171	10	8	20%	1 x 10^{-4}	1,29
22	4_49_17	12	10	16,7%	1 x 10^{-4}	1,45
23	aj-e11_165	13	11	15,4%	1 x 10^{-4}	1,29
24	alu-v2_31	13	10	23,1%	2 x 10^{-4}	1,27
25	mod8-10_177	14	13	7,1%	1 x 10^{-4}	1,28
26	4_49_16	16	13	18,8%	2 x 10^{-4}	1,37
27	rd53_137	16	15	6,3%	2 x 10^{-4}	1,45
28	4gt4-v0_73	17	13	23,5%	3 x 10^{-4}	1,43

Fonte: Elaborado pelo próprio autor.

Tabela 17 – Quadro de resultado do método *Greedy* (continuação).

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
29	hwb4_49	17	15	11,8%	2 x 10^{-4}	1,34
30	alu-v2_30	18	14	22,2%	3 x 10^{-4}	1,44
31	mod5adder_127	21	21	0%	1 x 10^{-4}	1,44
32	ham7_106	25	24	4%	3 x 10^{-4}	1,82
33	rd53_131	28	19	32,1%	4 x 10^{-4}	1,68
34	rd53_130	30	30	0%	1 x 10^{-4}	1,79
35	sym6_145	36	36	0%	2 x 10^{-4}	1,64
36	hwb7_62	331	324	2,1%	6,3 x 10^{-3}	2,30
37	hwb8_116	749	730	2,5%	2,4 x 10^{-2}	3,78
38	hwb9_123	1959	1932	1,4%	9,9 x 10^{-2}	7,43
39	urf2_152	5030	4693	6,7%	2,14	7,95
40	urf5_158	10276	9445	8,1%	10,93	15,37
41	urf1_149	11554	10749	7%	12,09	17,22
42	urf3_155	26468	24668	6,8%	57,66	24,91

Fonte: Elaborado pelo próprio autor.

5.1.2 Método *Simulated Annealing*

No método SA, os parâmetros utilizados foram baseados no trabalho de Abdessaied *et al.* (2015). Além desses parâmetros, foi adicionada a variável *Limite*, que demarca a quantidade máxima de portas que o circuito pode alcançar, sendo *min* o menor circuito encontrado durante a aplicação do método. Isso ocorre devido a estratégia empregada na meta-heurística SA, que aceita soluções piores no início do processo.

Outro fator importante é que nesta pesquisa foi aplicada apenas uma regra de reescrita a cada iteração, diferente do trabalho de Abdessaied *et al.* (2015) onde são aplicadas duas regras.

Na Tabela 18 são exibidas as variáveis e seus respectivos valores utilizados em ambos os testes. Cabe salientar que esses valores foram analisados de forma empírica.

O método SA foi executado nos 42 circuitos de duas maneiras. Nas Tabelas 19 e 20 são apresentados os resultados gerados sem o método Divisão e nas Tabelas 21 e 22 com o método Divisão. Nota-se que combinado com o método Divisão, o método SA foi mais eficiente ao encontrar circuitos menores.

Os parâmetros escolhidos para o método Divisão são EDB e *Reut* iguais a 10 e 5, respectivamente. Esses valores foram testados de forma empírica, tanto para o valor da

Tabela 18 – Parâmetros utilizados no método *Simulated Annealing*.

Variável	**Valor**
Temperatura inicial	100
Parada	5
Limite	$min + 3$
α	80%
Iterações	100

Fonte: Elaborado pelo próprio autor.

variável EDB quanto para o valor do *Reut*. De acordo com os testes, observou-se que quanto maior o valor de EDB menor é o tempo de execução, porém menor é a redução no custo do circuito.

Tabela 19 – Quadro de resultado do método *Simulated Annealing*.

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
1	4gt11-v1_85	4	3	25,0%	0,01	1,52
2	4gt13-v1_93	4	3	25,0%	0,01	1,58
3	4gt12-v1_89	5	4	20,0%	0,01	1,42
4	4mod5-v0_19	5	3	40,0%	0,01	1,49
5	4mod5-v1_24	5	4	20,0%	0,01	1,71
6	mod5mils_65	5	3	40,0%	0,01	1,48
7	mod5mils_71	5	3	40,0%	0,01	1,39
8	3_17_13	6	4	33,3%	0,01	1,38
9	3_17_14	6	5	16,7%	0,02	1,51
10	4gt4-v0_72	6	5	16,7%	0,01	1,32
11	decod24-v0_38	6	5	16,7%	0,02	1,59
12	alu-v2_32	7	6	14,3%	0,01	1,63
13	alu-v2_33	7	6	14,3%	0,01	1,43
14	mod10_176	7	5	28,6%	0,01	1,42
15	mod5d1_63	7	4	42,9%	0,03	1,56
16	4mod5-v1_23	8	4	50,0%	0,01	1,18
17	mod8-10_178	9	8	11,1%	0,02	1,61
18	4gt12-v0_87	10	9	10,0%	0,02	1,57
19	4gt13_91	10	9	10,0%	0,02	1,46
20	aj-e11_168	10	9	10,0%	0,02	1,55
21	mod10_171	10	6	40,0%	0,02	1,47
22	4_49_17	12	10	16,7%	0,02	1,45
23	aj-e11_165	13	11	15,4%	0,04	1,63
24	alu-v2_31	13	5	61,5%	0,08	1,62
25	mod8-10_177	14	9	35,7%	0,05	1,67
26	4_49_16	16	13	18,8%	0,04	1,59
27	rd53_137	16	15	6,3%	0,04	1,60

Fonte: Elaborado pelo próprio autor.

Tabela 20 – Quadro de resultado do método *Simulated Annealing* (continuação).

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
28	4gt4-v0_73	17	13	23,5%	0,03	1,89
29	hwb4_49	17	15	11,8%	0,03	1,55
30	alu-v2_30	18	14	22,2%	0,1	1,63
31	mod5adder_127	21	16	23,8%	0,05	1,77
32	ham7_106	25	25	0,0%	0,04	2,32
33	rd53_131	28	19	32,1%	0,08	1,61
34	rd53_130	30	30	0,0%	0,05	1,93
35	sym6_145	36	27	25,0%	0,16	1,54
36	hwb7_62	331	329	0,6%	1,04	2,68
37	hwb8_116	749	744	0,7%	5,01	4,09
38	hwb9_123	1959	1959	0,0%	3,26	7,58
39	urf2_152	5030	5030	0,0%	12,55	8,37
40	urf5_158	10276	10273	0,0%	53,85	15,55
41	urf1_149	11554	11554	0,0%	23,51	17,61
42	urf3_155	26468	26466	0,0%	98,28	26,51

Fonte: Elaborado pelo próprio autor.

Tabela 21 – Quadro de resultado do método *Simulated Annealing* combinado ao método Divisão.

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
1	4gt11-v1_85	4	3	25,0%	0,01	1,62
2	4gt13-v1_93	4	3	25,0%	0,02	1,42
3	4gt12-v1_89	5	4	20,0%	0,02	1,42
4	4mod5-v0_19	5	3	40,0%	0,02	1,57
5	4mod5-v1_24	5	4	20,0%	0,02	1,32
6	mod5mils_65	5	3	40,0%	0,02	1,36
7	mod5mils_71	5	3	40,0%	0,02	1,38
8	3_17_13	6	4	33,3%	0,02	1,38
9	3_17_14	6	5	16,7%	0,02	1,38
10	4gt4-v0_72	6	5	16,7%	0,02	1,56
11	decod24-v0_38	6	5	16,7%	0,02	1,54
12	alu-v2_32	7	6	14,3%	0,03	1,48
13	alu-v2_33	7	6	14,3%	0,02	1,56
14	mod10_176	7	5	28,6%	0,03	1,55
15	mod5d1_63	7	4	42,9%	0,03	1,59
16	4mod5-v1_23	8	4	50,0%	0,02	1,36
17	mod8-10_178	9	7	22,2%	0,06	1,58

Fonte: Elaborado pelo próprio autor.

Tabela 22 – Quadro de resultado do método *Simulated Annealing* combinado ao método Divisão (continuação).

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
18	4gt12-v0_87	10	9	10,0%	0,04	1,62
19	4gt13_91	10	9	10,0%	0,04	1,46
20	aj-e11_168	10	7	30,0%	0,07	1,85
21	mod10_171	10	6	40,0%	0,03	1,54
22	4_49_17	12	9	25,0%	0,05	1,48
23	aj-e11_165	13	11	15,4%	0,06	1,60
24	alu-v2_31	13	6	53,8%	0,06	1,63
25	mod8-10_177	14	10	28,6%	0,09	1,63
26	4_49_16	16	13	18,8%	0,08	1,62
27	rd53_137	16	13	18,8%	0,09	1,60
28	4gt4-v0_73	17	13	23,5%	0,07	1,74
29	hwb4_49	17	14	17,6%	0,09	1,60
30	alu-v2_30	18	14	22,2%	0,12	1,55
31	mod5adder_127	21	16	23,8%	0,11	1,60
32	ham7_106	25	23	8,0%	0,14	2,05
33	rd53_131	28	12	57,1%	0,12	1,92
34	rd53_130	30	18	40,0%	0,31	1,91
35	sym6_145	36	18	50,0%	0,48	1,72
36	hwb7_62	331	293	11,5%	6,97	2,73
37	hwb8_116	749	637	15,0%	12,84	4,19
38	hwb9_123	1959	1721	12,1%	75,74	7,70
39	urf2_152	5030	3144	37,5%	212,64	7,61
40	urf5_158	10276	5063	50,7%	942,36	15,55
41	urf1_149	11554	6853	40,7%	942,86	16,22
42	urf3_155	26468	14353	45,8%	4431,48	22,87

Fonte: Elaborado pelo próprio autor.

5.1.3 Método *Variable Neighbourhood Descent*

O método VND tem o intuito de aumentar a vizinhança em busca de um circuito menor. Cabe salientar que na otimização de circuitos reversíveis a vizinhança refere-se a quantidade de regra de reescrita aplicada.

Executou-se o método VND nos 42 circuitos de duas maneiras. Nas Tabelas 23 e 24 são apresentados os resultados gerados sem o método Divisão e nas Tabelas 25 e 26 com o método Divisão.

Os parâmetros utilizados também foram testados de forma empírica. Por exemplo, a quantidade de iterações efetuada a cada vizinhança, na qual observou-se que quanto

maior o número de iterações, maior é o tempo de execução. Para o método Divisão, os valores para EDB e *Reut* são 10 e 5, respectivamente. Como no método SA, verificou-se que quanto maior o EDB menor o tempo de execução. No entanto, menor é a redução no custo do circuito.

Tabela 23 – Quadro de resultado do método *Variable Neighbourhood Descent*.

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
1	4gt11-v1_85	4	3	25%	0,02	1,199
2	4gt13-v1_93	4	3	25%	0,02	1,199
3	4gt12-v1_89	5	4	20%	0,03	1,398
4	4mod5-v0_19	5	3	40%	0,02	1,309
5	4mod5-v1_24	5	4	20%	0,02	1,297
6	mod5mils_65	5	3	40%	0,02	1,141
7	mod5mils_71	5	3	40%	0,02	1,301
8	3_17_13	6	4	33,3%	0,05	1,355
9	3_17_14	6	5	16,7%	0,05	1,266
10	4gt4-v0_72	6	5	16,7%	0,02	1,238
11	decod24-v0_38	6	5	16,7%	0,02	1,195
12	alu-v2_32	7	6	14,3%	0,03	1,348
13	alu-v2_33	7	6	14,3%	0,02	1,172
14	mod10_176	7	5	28,6%	0,02	1,285
15	mod5d1_63	7	6	14,3%	0,03	1,203
16	4mod5-v1_23	8	4	50%	0,03	1,172
17	mod8-10_178	9	7	22,2%	0,04	1,211
18	4gt12-v0_87	10	9	10%	0,06	1,273
19	4gt13_91	10	9	10%	0,03	1,207
20	aj-e11_168	10	9	10%	0,03	1,367
21	mod10_171	10	8	20%	0,03	1,172
22	4_49_17	12	10	16,7%	0,04	1,324
23	aj-e11_165	13	11	15,4%	0,04	1,367
24	alu-v2_31	13	9	30,8%	0,05	1,414
25	mod8-10_177	14	7	50%	0,12	1,363
26	4_49_16	16	13	18,8%	0,04	1,328
27	rd53_137	16	12	25%	0,09	1,414
28	4gt4-v0_73	17	13	23,5%	0,04	1,246
29	hwb4_49	17	15	11,8%	0,05	1,359
30	alu-v2_30	18	14	22,2%	0,08	1,418
31	mod5adder_127	21	16	23,8%	0,13	1,441
32	ham7_106	25	24	4%	0,09	1,668
33	rd53_131	28	15	46,4%	0,21	1,676
34	rd53_130	30	30	0%	0,07	1,828
35	sym6_145	36	27	25%	0,22	1,633

Fonte: Elaborado pelo próprio autor.

Tabela 24 – Quadro de resultado do método *Variable Neighbourhood Descent* (continuação).

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
36	hwb7_62	331	331	0%	0,72	2,426
37	hwb8_116	749	738	1,5%	5,95	4,313
38	hwb9_123	1959	1959	0%	8,36	7,523
39	urf2_152	5030	5030	0%	18,8	8,113
40	urf5_158	10276	10276	0%	57,84	17,918
41	urf1_149	11554	11554	0%	69,42	19,488
42	urf3_155	26468	26466	0%	498,56	38,414

Fonte: Elaborado pelo próprio autor.

Tabela 25 – Quadro de resultado do método *Variable Neighbourhood Descent* combinado ao método Divisão.

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
1	4gt11-v1_85	4	3	25%	0,04	1,234
2	4gt13-v1_93	4	3	25%	0,05	1,418
3	4gt12-v1_89	5	4	20%	0,04	1,184
4	4mod5-v0_19	5	3	40%	0,04	1,281
5	4mod5-v1_24	5	4	20%	0,04	1,336
6	mod5mils_65	5	3	40%	0,04	1,309
7	mod5mils_71	5	3	40%	0,05	1,293
8	3_17_13	6	4	33,3%	0,04	1,270
9	3_17_14	6	5	16,7%	0,04	1,133
10	4gt4-v0_72	6	5	16,7%	0,05	1,336
11	decod24-v0_38	6	5	16,7%	0,04	1,332
12	alu-v2_32	7	6	14,3%	0,06	1,215
13	alu-v2_33	7	6	14,3%	0,06	1,293
14	mod10_176	7	5	28,6%	0,05	1,352
15	mod5d1_63	7	5	28,6%	0,11	1,340
16	4mod5-v1_23	8	4	50%	0,06	1,180
17	mod8-10_178	9	7	22,2%	0,07	1,273
18	4gt12-v0_87	10	9	10%	0,07	1,180
19	4gt13_91	10	9	10%	0,06	1,297
20	aj-e11_168	10	9	10%	0,06	1,328
21	mod10_171	10	6	40%	0,12	1,141
22	4_49_17	12	9	25%	0,12	1,371
23	aj-e11_165	13	10	23,1%	0,18	1,238

Fonte: Elaborado pelo próprio autor.

Tabela 26 – Quadro de resultado do método *Variable Neighbourhood Descent* combinado ao método Divisão (continuação).

Circuito	Nome	Quantidade de portas no circuito		Redução do custo	Tempo de execução (s)	Memória (MB)
		Inicial	Final			
24	alu-v2_31	13	6	53,8%	0,13	1,250
25	mod8-10_177	14	10	28,6%	0,12	1,371
26	4_49_16	16	13	18,8%	0,31	1,375
27	rd53_137	16	13	18,8%	0,29	1,617
28	4gt4-v0_73	17	13	23,5%	0,14	1,555
29	hwb4_49	17	14	17,6%	0,17	1,363
30	alu-v2_30	18	14	22,2%	0,21	1,289
31	mod5adder_127	21	16	23,8%	0,24	1,566
32	ham7_106	25	21	16%	0,65	1,859
33	rd53_131	28	13	53,6%	0,34	1,539
34	rd53_130	30	18	40%	0,48	1,672
35	sym6_145	36	20	44,4%	0,92	1,648
36	hwb7_62	331	297	10,3%	7,11	2,512
37	hwb8_116	749	639	14,7%	39,78	3,914
38	hwb9_123	1959	1702	13,1%	191,59	7,418
39	urf2_152	5030	3115	38,1%	355,40	7,805
40	urf5_158	10276	5098	50,4%	1117,04	14,965
41	urf1_149	11554	6785	41,3%	1126,40	15,711
42	urf3_155	26468	14334	45,8%	4227,41	23,922

Fonte: Elaborado pelo próprio autor.

5.1.4 Análise final

De acordo com os resultados apresentados foi verificado que os métodos SA e VND combinados ao método Divisão geraram circuitos menores.

Além disso, observou-se que o método SA comparado ao VND, ambos combinados com o método Divisão, apresentou menor quantidade de portas em 7 dos 42 circuitos, mesmo custo em 29 circuitos e pior custo em 6. Sendo assim, definiu-se que o método mais eficiente, desenvolvido nesta pesquisa, para reduzir a quantidade de portas em circuitos reversíveis é o método *Simulated Annealing* combinado com o método Divisão.

Outro fator de comparação é a média de redução do custo no circuito obtida ao aplicar cada método. Por exemplo, considerando dois circuitos com redução de custo respectivamente de 10% e 20% possui uma média de (10% + 20%)/2 = 15%. Assim, de acordo com os dados gerados com a aplicação do algoritmo *Greedy* e dos métodos SA e VND com e sem Divisão obteve-se as médias de redução de custo, conforme apresenta-se

na Tabela 27.

Tabela 27 – Média de redução do custo com a aplicação dos métodos desenvolvidos.

Métodos	*Greedy*	Sem Divisão		Com Divisão	
		SA	VND	SA	VND
Média	13%	19,5%	19,1%	27,9%	27,2%

Em relação a memória consumida, observou-se que a quantidade de memória varia de acordo com a quantidade de variáveis e portas no circuito. Isto é, as características do circuito reversível são adicionadas em uma matriz, sendo as linhas e colunas representadas respectivamente pelas variáveis e portas do circuito original. Cabe salientar que o circuito pode aumentar durante a execução do método, aumentando também a quantidade de memória.

5.2 SEGUNDA AVALIAÇÃO

Na segunda avaliação, apresentada nas Tabelas 28 e 29, efetuou-se a comparação entre os métodos SA e VND combinados com o Divisão e os dados apresentados por Rahman, Soeken e Dueck (2015) e Almeida, Dueck e Silva (2018), que aplicam respectivamente a heurística *Dynamic Template* e meta-heurística *Tabu Search*.

Tabela 28 – Quantidade de portas no circuito final ao aplicar os métodos DT, TS, SA e VND.

Circuito	Nome	Quantidade de portas no circuito				
		Inicial	Final			
			Dynamic Template	Tabu Search	Método Divisão	
					SA	VND
1	4gt11-v1_85	4	3	3	3	3
2	4gt13-v1_93	4	3	3	3	3
3	4gt12-v1_89	5	4	4	4	4
4	4mod5-v0_19	5	3	3	3	3
5	4mod5-v1_24	5	4	4	4	4
6	mod5mils_65	5	3	3	3	3
7	mod5mils_71	5	3	3	3	3
8	3_17_13	6	4	4	4	4
9	3_17_14	6	5	5	5	5
10	4gt4-v0_72	6	5	5	5	5
11	decod24-v0_38	6	5	5	5	5
12	alu-v2_32	7	6	6	6	6

Fonte: Elaborado pelo próprio autor.

Tabela 29 – Quantidade de portas no circuito final ao aplicar os métodos DT, TS, SA e VND (continuação).

Circuito	Nome	Quantidade de portas no circuito				
		Inicial	Final			
			Dynamic Template	Tabu Search	Método Divisão	
					SA	VND
13	alu-v2_33	7	6	6	6	6
14	mod10_176	7	5	5	5	5
15	mod5d1_63	7	5	4	4	5
16	4mod5-v1_23	8	4	4	4	4
17	mod8-10_178	9	8	6	7	7
18	4gt12-v0_87	10	9	9	9	9
19	4gt13_91	10	9	9	9	9
20	aj-e11_168	10	9	7	7	9
21	mod10_171	10	9	6	6	6
22	4_49_17	12	10	9	9	9
23	aj-e11_165	13	11	10	11	10
24	alu-v2_31	13	9	5	6	6
25	mod8-10_177	14	11	7	10	10
26	4_49_16	16	13	12	13	13
27	rd53_137	16	13	13	13	13
28	4gt4-v0_73	17	13	13	13	13
29	hwb4_49	17	14	13	14	14
30	alu-v2_30	18	15	15	14	14
31	mod5adder_127	21	16	10	16	16
32	ham7_106	25	22	22	23	21
33	rd53_131	28	13	13	12	13
34	rd53_130	30	22	18	18	18
35	sym6_145	36	31	19	18	20
36	hwb7_62	331	294	290	293	297
37	hwb8_116	749	638	631	637	639
38	hwb9_123	1959	1701	1715	1721	1702
39	urf2_152	5030	3274	3142	3144	3115
40	urf5_158	10276	5578	5183	5063	5098
41	urf1_149	11554	7347	6941	6853	6785
42	urf3_155	26468	15736	Indefinido	14353	14334

Fonte: Elaborado pelo próprio autor.

Cabe salientar que nesses trabalhos não foi apresentado o consumo de memória ao executar o método. Além disso, apenas no artigo apresentado por Almeida, Dueck e Silva (2018), foi disponibilizado o tempo de execução para alguns casos.

Apresenta-se na Tabela 30 as comparações efetuadas a cada dois métodos. Nota-se nas Tabelas 30(a) e 30(b) que tanto o SA quanto o VND geraram resultados melhores que o DT.

Tabela 30 – Comparação entre os métodos DT, TS e SA e VND combinados ao Divisão.

(a) SA/DT.

SA <DT	SA >DT	SA = DT
14	3	25

(b) VND/DT.

VND <DT	VND >DT	VND = DT
14	3	25

(c) SA/TS.

SA <TS	SA >TS	SA = TS
6	12	24

(d) VND/TS.

VND <TS	VND >TS	VND = TS
7	11	24

Fonte: Elaborado pelo próprio autor.

De acordo com os dados gerados, observou-se que o método TS obteve melhores resultados comparado aos método SA e VND, conforme apresentam-se nas Tabelas 30(c) e 30(d). No entanto, analisou-se que em circuitos com mais de 1000 portas o método VND gerou circuitos menores que o TS em 5 dos 5 casos.

Por fim, observou-se também que o resultado gerado com a aplicação do método TS no circuito urf3_155 está referenciado como indefinido, isso deve-se ao tempo elevado de mais de 10 horas ao executá-lo. Apresenta-se na Tabela 31 os dados obtidos com a execução dos métodos TS, SA e VND nos circuitos urf1_149 e urf3_155. É possível observar que além dos métodos SA e VND gerarem circuitos menores foram executados em menor tempo.

Tabela 31 – Análise do tempo de execução do método TS comparado ao SA e VND combinados ao Divisão.

Nome	Quantidade de portas no circuito final			Tempo de execução (s)		
	TS	SA	VND	TS	SA	VND
urf1_149	6941	6853	6785	5.118,86	942,86	1126,4
urf3_155	Indefinido	14353	14334	Indefinido	4431,48	4227,41

Fonte: Elaborado pelo próprio autor.

6 CONCLUSÃO

Neste trabalho foram estudados métodos aplicados para a otimização pós-síntese de circuitos reversíveis, denominados de métodos de otimização. Os métodos de otimização são operados em conjunto com as regras de reescrita, que possuem como funcionalidade modificar o circuito sem alterar a respectiva funcionalidade. Esses métodos operam utilizando os métodos heurísticos como metodologia, denominados heurística *Greedy* e meta-heurísticas *Simulated Annealing* e *Variable Neighbourhood Descent*.

Com o decorrer da pesquisa, verificou-se que os métodos de otimização que aplicam as meta-heurísticas *Simulated Annealing* e *Variable Neighbourhood Descent* perdem a eficiência conforme o aumento na quantidade de portas iniciais no circuito. Portanto, desenvolveu-se o método Divisão, que é adicionado ao processo de otimização para diminuir o espaço de busca. O objetivo é dividir o circuito em várias vizinhanças e fazer uma busca pelo ótimo local em cada uma.

Foram utilizados como teste 42 circuitos reversíveis com diferentes características, diferenciadas por quantidade de portas e variáveis. Para definir o método mais eficiente na redução de portas no circuito foram efetuadas duas avaliações.

Na primeira avaliação comparou-se os métodos *Greedy*, *Simulated Annealing* e *Variable Neighbourhood Descent*, na qual o método que apresentou melhor desempenho foi o *Simulated Annealing* junto ao método Divisão. Como forma de avaliação adotou-se a média de redução de custo dos 42 circuitos utilizados, na qual o método SA obteve uma média de 27,9% e o método VND 27,2%.

Na segunda avaliação, analisou-se os métodos SA e VND combinados ao método Divisão e os métodos *Dynamic Template* e *Tabu Search*, e o método que apresentou circuitos menores é o que emprega a meta-heurística *Tabu Search*.

No entanto, em circuitos com quantidade de portas iniciais maior, os métodos *Simulated Annealing* e *Variable Neighbourhood Descent* geraram circuitos menores em menor tempo de execução.

Cabe salientar que considerando apenas os circuitos com mais de 1000 portas reversíveis, o *Variable Neighbourhood Descent* apresentou menor custo que o *Tabu Search* e o *Simulated Annealing* em 5 dos 6 casos.

Para trabalho futuro tem-se como ideia testar afundo os parâmetros dos métodos SA e VND combinados ao método Divisão em busca de circuitos menores. Efetuar também a implementação do método *Tabu Search* combinado com o método Divisão para obter resultados em menor tempo de execução.

Além disso, ao invés de reduzir a quantidade de portas no circuito reversível, tem-se como intuito utilizar o custo quântico como parâmetro de otimização.

REFERÊNCIAS

ABDESSAIED, N. *et al.* Reversible circuit rewriting with simulated annealing. In: INTERNATIONAL CONFERENCE ON VERY LARGE SCALE INTEGRATION-VLSI-SOC, 23., 2015, Daejeon. **Anais...** Daejeon: IEEE, 2015. p. 286–291.

ABUBAKAR, M. Y. *et al.* New universal gate library for synthesizing reversible logic circuit using genetic programming. In: INTERNATIONAL CONFERENCE ON COMPUTER AND INFORMATION SCIENCES- ICCOINS, 3., 2016, Kuala Lumpur. **Anais...** Kuala Lumpur: IEEE, 2016. p. 316–321.

ALMEIDA, A. A. de; DUECK, G. W.; SILVA, A. C. da. Reversible circuit optimization based on tabu search. In: INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC- ISMVL, 48., 2018, Linz. **Anais...** Linz: IEEE, 2018. p. 103–108.

BAHAUDDIN, S. M.; IRFAN, A. A. M. Permutation algebra for constructing reversible circuits. **Journal of Quantum Information Science**, Dakha, v. 2, n. 3, p. 61, 2012.

BANDYOPADHYAY, C.; PAREKH, S.; RAHAMAN, H. A synthesis approach for esop-based reversible circuit. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS- ICACCI, 5., 2016, Jaipur. **Anais...** Jaipur: IEEE, 2016. p. 1741–1746.

BANDYOPADHYAY, C. *et al.* Esop-based synthesis of reversible circuit using improved cube list. In: INTERNATIONAL SYMPOSIUM ON ELECTRONIC SYSTEM DESIGN-ISED, 4., 2013, Singapore. **Anais...** Singapore: IEEE, 2013. p. 26–30.

BENNETT, C. H. Logical reversibility of computation. **IBM Journal of Research and Development**, Yorktown Heights, v.17, n. 6, p. 525–532, 1973.

CHACKO, J. B.; WHIG, P. Low delay based full adder/subtractor by mig and cog reversible logic gate. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL INTELLIGENCE AND COMMUNICATION NETWORKS- CICN, 8., 2016, Tehri. **Anais...** Tehri: IEEE, 2016. p. 585–589.

DATTA, K. *et al.* A cycle based reversible logic synthesis approach. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING AND COMMUNICATIONS- ICACC, 3., 2013, Cochin. **Anais...** Cochin: IEEE, 2013. p. 316–319.

DATTA, K. *et al.* Synthesis of reversible circuits using heuristic search method. In: INTERNATIONAL CONFERENCE ON VLSI DESIGN- VLSID, 25., 2012, Hyderabad. **Anais...** Hyderabad: IEEE, 2012. p. 328–333.

DATTA, K.; SENGUPTA, I.; RAHAMAN, H. Particle swarm optimization based circuit synthesis of reversible logic. In: INTERNATIONAL SYMPOSIUM ON ELECTRONIC SYSTEM DESIGN- ISED, 3., 2012, Kolkata. **Anais...** Kolkata: IEEE, 2012. p. 226–230.

DATTA, K.; SENGUPTA, I.; RAHAMAN, H. A post-synthesis optimization technique for reversible circuits exploiting negative control lines. **IEEE Transactions on Computers**, Shibpur, v. 64, n. 4, p. 1208–1214, 2015.

DESHPANDE, M. J.; JAYASHREE, H.; AGRAWAL, V. Reversible circuit optimization using modified toffoli templates. In: INTERNATIONAL CONFERENCE ON ADVANCES IN COMPUTING, COMMUNICATIONS AND INFORMATICS- ICACCI, 6., 2017, Udupi. **Anais...** Udupi: IEEE, 2017. p. 344–349.

DESOETE, B.; VOS, A. D. Feynman's reversible logic gates implemented in silicon. In: ADVANCED TRAINING COURSE ON MIXED DESIGN OF VLSI CIRCUITS, 6., 1999, Krakow. **Anais...** Krakow: Department of Electronics and information systems, 1999. p. 497–502.

FAZEL, K.; THORNTON, M. A.; RICE, J. Esop-based toffoli gate cascade generation. In: PACIFIC RIM CONFERENCE ON COMMUNICATIONS, COMPUTERS AND SIGNAL PROCESSING., 8., 2007, Hong Kong. **Anais...** Hong Kong: IEEE, 2007. p. 206–209.

FEYNMAN, R. P. Quantum mechanical computers. **Optics News**, California, v. 11, n. 2, p. 11–20, 1985.

GENDREAU, M.; POTVIN, J. Y. **Handbook of metaheuristics**. Heidelberg: Springer International Publishing, 2010. 648 p.

GOVINDAPRIYA, K.; PERIYASAMY, M. Nano design of communication parts with qca using reversible logic. In: INTERNATIONAL CONFERENCE ON ADVANCED COMMUNICATION CONTROL AND COMPUTING TECHNOLOGIES- ICACCCT, 27., 2016, Ramanathapuram. **Anais...** Ramanathapuram: IEEE, 2016. p. 819–823.

HÄNNINEN, I. K.; LENT, C. S.; SNIDER, G. L. Quantifying irreversible information loss in digital circuits. **ACM Journal on Emerging Technologies in Computing Systems- JETC**, Notre Dame, v. 11, n. 2, p. 10, 2014.

KOLE, A. *et al.* Exact synthesis of ternary reversible functions using ternary toffoli gates. In: INTERNATIONAL SYMPOSIUM ON MULTIPLE VALUED LOGIC- ISMVL, 47., 2017, Novi Sad. **Anais...** Novi Sad: IEEE, 2017. p. 179–184.

LANDAUER, R. Irreversibility and heat generation in the computing process. **IBM Journal of Research and Development**, Yorktown Heights, v. 5, n. 3, p. 183–191, 1961.

MASLOV, D.; DUECK, G. W. Reversible cascades with minimal garbage. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, California, v. 23, n. 11, p. 1497–1509, 2004.

MERKLE, R. C. Reversible electronic logic using switches. **Nanotechnology**, Bristal, v. 4, n. 1, p. 21, 1993.

MILLER, D. M.; MASLOV, D.; DUECK, G. W. A transformation based algorithm for reversible logic synthesis. In: DESIGN AUTOMATION CONFERENCE, 40., 2003, Anaheim. **Anais...** Anaheim: IEEE, 2003. p. 318–323.

MILLER, D. M.; WILLE, R.; DUECK, G. W. Synthesizing reversible circuits for irreversible functions. In: EUROMICRO CONFERENCE ON DIGITAL SYSTEM DESIGN, ARCHITECTURES, METHODS AND TOOLS- DSD, 12., 2009, Patras. **Anais...** Patras: IEEE, 2009. p. 749–756.

NIELSEN, M. A.; CHUANG, I. L. **Quantum computation and quantum information**. Cambridge: Cambridge University Press, 2000. 704 p.

PICTON, P. Optoelectronic multi-valued conservative logic. **International Journal of Optical Computing**, Chichester, v. 2, n. 1, p. 19–29, 1991.

PODLASKI, K. Reversible circuit synthesis using binary decision diagrams. In: INTERNATIONAL CONFERENCE MIXED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS- MIXDES, 23., 2016, Lodz. **Anais...** Lodz: IEEE, 2016. p. 235–238.

POSSAGNOLO, L. H. F. M. **Reconfiguração de sistemas de distribuição operando em vários níveis de demanda através de uma meta-heurística de busca em vizinhança variável**. Ilha solteira: Universidade Estadual Paulista- UNESP, 2015. 178 p.

RAHMAN, M. M.; SOEKEN, M.; DUECK, G. W. Dynamic template matching with mixed-polarity toffoli gates. In: INTERNATIONAL SYMPOSIUM ON MULTIPLE-VALUED LOGIC- ISMVL, 45., 2015, Waterloo. **Anais...** Waterloo: IEEE, 2015. p. 72–77.

RENNO, D. U. *et al.* Algoritmo divisão utilizando a meta-heurística simulated annealing aplicado na otimização de circuitos reversíveis. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA- CBA, 65., 2018, Paraíba. **Anais...** Paraíba: CBA, 2018. p. 1–7.

SAEEDI, M.; ZAMANI, M. S.; SEDIGHI, M. Moving forward: A non-search based synthesis method toward efficient cnot-based quantum circuit synthesis algorithms. In: SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE- ASP-DAC, 13., 2008, Seoul. **Anais...** Seoul: IEEE, 2008. p. 83–88.

SARKAR, M.; GHOSAL, P.; MOHANTY, S. P. Reversible circuit synthesis using aco and sa based quine-mccluskey method. In: CIRCUITS AND SYSTEMS (MWSCAS), INTERNATIONAL MIDWEST SYMPOSIUM ON, 56., 2013, Columbus. **Anais...** Columbus: IEEE, 2013. p. 416–419.

SASAMAL, T. N.; SINGH, A. K.; MOHAN, A. Reversible logic circuit synthesis and optimization using adaptive genetic algorithm. **Procedia Computer Science**, Amsterdam, v. 70, n. 3, p. 407–413, 2015.

SHUKLA, V. *et al.* Design and performance analysis for the reversible realization of adder/subtractor circuit. In: INTERNATIONAL CONFERENCE ON EMERGING TRENDS IN COMPUTING AND COMMUNICATION TECHNOLOGIES- ICETCCT, 8., 2017, Dehradun. **Anais...** Dehradun: IEEE, 2017. p. 1–6.

SOEKEN, M. *et al.* Hierarchical reversible logic synthesis using luts. In: DESIGN AUTOMATION CONFERENCE- DAC, 54., 2017, Austin. **Anais...** Austin: IEEE, 2017. p. 1–6.

SOEKEN, M.; THOMSEN, M. K. White dots do matter: rewriting reversible logic circuits. In: INTERNATIONAL CONFERENCE ON REVERSIBLE COMPUTATION, 5., 2013, Heidelberg. **Anais...** Heidelberg: Springer, 2013. p. 196–208.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Digital systems:principles and applications**. 12. ed. Dallas: Pearson Education, 2017. 57–58 p.

TOFFOLI, T. Reversible computing. In: INTERNATIONAL COLLOQUIUM ON AUTOMATA, LANGUAGES, AND PROGRAMMING, 7., 1980, Heidelberg. **Anais...** Heidelberg: Springer, 1980. p. 632–644.

VASUDEVAN, D. P.; LALA, P. K.; PARKERSON, J. P. Cmos realization of online testable reversible logic gates. In: COMPUTER SOCIETY ANNUAL SYMPOSIUM ON VLSI, 4., 2005, Tampa. **Anais...** Tampa: IEEE, 2005. p. 309–310.

WANG, W.-H. *et al.* Quantum-inspired tabu search algorithm for reversible logic circuit synthesis. In: INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS- SMC, 13., 2012, Seoul. **Anais...** Seoul: IEEE, 2012. p. 709–714.

WILLE, R.; DRECHSLER, R. **Towards a design flow for reversible logic**. Heidelberg: Springer Science & Business Media, 2010.

WILLE, R. *et al.* Revlib: An online resource for reversible functions and reversible circuits. In: INTERNATIONAL SYMPOSIUM ON MULTIPLE VALUED LOGIC-ISMVL, 38., 2008, Dallas. **Anais...** Dallas: IEEE, 2008. p. 220–225.

YELEKAR, P. R.; CHIWANDE, S. S. *et al.* Introduction to reversible logic gates & its application. In: NATIONAL CONFERENCE ON INFORMATION AND COMMUNICATION TECHNOLOGY- NCICT, 2., 2011, Singapore. **Anais...** Singapore: IEEE, 2011. p. 5–9.

ZENG, G.-J. *et al.* A novel classifying algorithm for reversible circuit synthesis. In: INTERNATIONAL CONFERENCE ON SYSTEMS, MAN, AND CYBERNETICS-SMC, 16., 2015, Kowloon. **Anais...** Kowloon: IEEE, 2015. p. 62–67.