COPPE
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia
UFRJ

SÍNTESE DE CIRCUITOS PARA COMPUTAÇÃO REVERSÍVEL USANDO PORTAS TOFFOLI GENERALIZADAS

Edinelço Dalcumune

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Franklin de Lima Marquezino
Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Rio de Janeiro
Julho de 2021

SÍNTESE DE CIRCUITOS PARA COMPUTAÇÃO REVERSÍVEL USANDO PORTAS TOFFOLI GENERALIZADAS

Edinelço Dalcumune

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientadores: Franklin de Lima Marquezino
Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Aprovada por: Prof. Franklin de Lima Marquezino
Prof. Celina Miraglia Herrera de Figueiredo
Prof. Luis Antonio Brasil Kowada
Prof. Márcia Helena Costa Fampa
Prof. Renato Portugal
Prof. Omar Paranaiba Vilela Neto

RIO DE JANEIRO, RJ – BRASIL
JULHO DE 2021

Dalcumune, Edinelço

Síntese de Circuitos para Computação Reversível usando Portas Toffoli Generalizadas/Edinelço Dalcumune. – Rio de Janeiro: UFRJ/COPPE, 2021.

XII, 57 p.: il.; 29, 7cm.

Orientadores: Franklin de Lima Marquezino
Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2021.

Referências Bibliográficas: p. 33 – 36.

1. Desenvolvimento de algoritmos. 2. Computação reversível. 3. Síntese de circuitos. 4. Representação de permutações através de ciclos. I. Marquezino, Franklin de Lima *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Dedico aos meus pais,
Maria Luiza e Francisco.

Agradecimentos

Agradeço aos meus pais e familiares pelo apoio.

Aos meus orientadores, Franklin, Celina e Luis Antonio, pela sugestão do tema, pela paciência, e pela dedicação e incentivo demonstrados durante o desenvolvimento desta tese. Não tenho palavras para expressar minha gratidão.

Ao PESC/COPPE/UFRJ pela estrutura oferecida, e a seus professores e demais funcionários pelo trabalho de excelência.

Aos colegas do Laboratório de Algoritmos e Combinatória, que me ajudaram muito durante esses anos e proporcionaram um bom ambiente de estudos e trabalho. Em especial, agradeço ao Alexsander de Melo que por diversas vezes se prontificou a ajudar com as dúvidas de programação e ao "não contemporâneo" André Ribeiro que também contribuiu muito para o desenvolvimento desta tese.

Aos professores Marcia Fampa, Renato Portugal e Omar Paranaiba por terem aceitado participar da banca examinadora e por suas contribuições. Ao professor Demerson Gonçalves pelas sugestões, no exame de qualificação.

Aos colegas do DCEX/UFVJM pelo incentivo e parceria.

A todas as pessoas que, de alguma forma, me ajudaram nesta caminhada. Em especial, agradeço à Mirelle e ao Elenilson pelas longas conversas e conselhos.

A todos os demais amigos, que prefiro não citar aqui, pois com certeza omitiria algum injustamente.

Agradeço a CAPES pelo apoio financeiro.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

SÍNTESE DE CIRCUITOS PARA COMPUTAÇÃO REVERSÍVEL USANDO PORTAS TOFFOLI GENERALIZADAS

Edinelço Dalcumune

Julho/2021

Orientadores: Franklin de Lima Marquezino
Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Programa: Engenharia de Sistemas e Computação

Apresentamos um novo algoritmo para síntese de circuitos reversíveis a partir de funções bijetivas. O algoritmo proposto usa portas Toffoli generalizadas, que incluem controles positivos e negativos. O algoritmo está dividido em duas partes. Primeiro, usamos portas Toffoli parcialmente controladas, com aumento progressivo do número de controles. Segundo, explorando propriedades de representação de permutações através de ciclos disjuntos, aplicamos portas Toffoli generalizadas com controles em todas as linhas exceto pela linha alvo. Portanto, uma das principais vantagens do algoritmo consiste no fato de obtermos circuitos que primeiro usam portas com custo baixo. Além disso, empregamos estratégias de síntese bidirecional para melhorar o número de portas. Comparamos os resultados obtidos pelo nosso algoritmo de síntese com os melhores resultados conhecidos usando a biblioteca de portas Toffoli generalizadas. Para o conjunto formado por todas as funções bijetivas com 3 bits, obtivemos a média de 5,23 portas por função, que é a melhor média de portas até onde sabemos, exceto para procedimentos que dão resultado exato mas não funcionam com funções bijetivas com mais bits. Isso significa uma melhora de 2,8% quando comparado ao melhor resultado conhecido obtido por uma heurística. Para os experimentos com vinte funções usadas como benchmark, obtivemos resultados semelhantes aos encontrados pelos melhores algoritmos da literatura. Além disso, nosso algoritmo de síntese funciona para um número n qualquer de bits. Propomos também uma nova regra e um novo algoritmo para otimização pós-síntese de circuitos reversíveis usando portas Toffoli generalizadas.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

CIRCUIT SYNTHESIS FOR REVERSIBLE COMPUTING USING GENERALIZED TOFFOLI GATES

Edinelço Dalcumune

July/2021

Advisors: Franklin de Lima Marquezino
Celina Miraglia Herrera de Figueiredo
Luis Antonio Brasil Kowada

Department: Systems Engineering and Computer Science

We present a new algorithm for synthesis of reversible circuits from bijective functions. This algorithm uses generalized Toffoli gates, which include positive and negative controls. Our algorithm is divided into two parts. First, we use partially controlled generalized Toffoli gates, progressively increasing the number of controls. Second, exploring the properties of the representation of permutations in disjoint cycles, we apply generalized Toffoli gates with controls on all lines except for the target line. Therefore, new in the method is the fact that the obtained circuits use first low cost gates and consider increasing costs towards the end of the synthesis. In addition, we employ two bidirectional synthesis strategies to improve the gate count, which is the metric used to compare the results obtained by our algorithm with the results presented in the literature. Our experimental results consider all 3-bit bijective functions and twenty widely used benchmark functions. The results obtained by our synthesis algorithm are competitive when compared with the best results known in the literature, considering as a complexity metric just the number of gates, as done by alternative best heuristics found in the literature. For example, for all 3-bit bijective functions using generalized Toffoli gates library, we obtained the best so far average count of 5.23. Our method gives an improvement of 2.8% over the best known result obtained by an heuristic. We also propose a new rule and a new algorithm for post-synthesis optimization of reversible circuits composed of generalized Toffoli gates.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos

C	Ciclo pertencente a uma permutação π, p. 8
$S(C)$	Soma das distâncias de Hamming entre elementos consecutivos de um ciclo C, p. 9
$X(i, j)$	Representação equivalente para porta GT totalmente controlada, p. 5
$\mathcal{C}$	Circuito reversível, p. 9
$\oplus$	Operação XOR bit a bit, p. 10
$d_H(\pi)$	Distância de permutações entre π e a identidade ι, p. 6
$d_H(\pi_j)$	Distância de Hamming entre π_j e j, p. 8
$d_H(i, j)$	Distância de Hamming entre dois valores i e j, p. 5
$d_H(\pi, \sigma)$	Distância de permutações entre π e σ, p. 6
$gc(\mathcal{C})$	Número de portas de um circuito reversível $\mathcal{C}$, p. 9
$qc(g)$	Custo quântico de uma porta reversível g, p. 9

Capítulo 1

Introdução

Nas últimas décadas, a síntese de circuitos reversíveis tem recebido atenção considerável devido à possibilidade de aplicações em várias áreas da ciência, tais como processamento de sinais, criptografia, computação quântica, computação gráfica, bioinformática entre muitas [27]. Uma das motivações principais para computação reversível reside no fato de que a computação quântica tem um dos seus fundamentos na reversibilidade de todas as portas, isto é, modelos para circuitos em computação quântica são reversíveis. Este fato se baseia no postulado da mecânica quântica, onde um operador unitário descreve a evolução de tempo do estado de um sistema fechado [21]. Outra motivação principal vem das importantes consequências físicas para computação reversível. LANDAUER [15] provou que o uso de portas lógicas irreversíveis necessariamente produz dissipação de calor independente da tecnologia utilizada. A razão é que cada bit apagado produz pelo menos $kT \ln 2$ de dissipação de energia, onde k é a constante de Boltzmann e T é a temperatura absoluta do circuito. Por outro lado, BENNETT [2] mostrou como usar portas lógicas reversíveis para reduzir ou mesmo eliminar a dissipação de calor em um circuito. De fato, em um circuito reversível— clássico ou quântico—podemos recuperar toda informação da entrada do circuito usando o que foi obtido na saída do circuito. Portanto, no processo nenhuma informação da entrada é apagada. Além disso, Bennett também mostrou que dissipação nula de calor em circuitos só é possível se o seu cálculo é reversível.

Um problema importante em computação reversível que tem sido intensamente estudado nas últimas décadas é a síntese de circuitos reversíveis. O problema pode ser enunciado como: dada uma função f, encontrar o melhor circuito reversível possível que implementa f, considerando uma métrica e uma biblioteca de portas reversíveis previamente definidas [9, 11, 14, 18, 20, 27, 29].

Nesta tese, apresentamos um novo algoritmo para síntese de circuitos reversíveis usando portas Toffoli com múltiplos controles [12, 34] com um número arbitrário de controles positivos e negativos, também chamadas portas Toffoli generalizadas [16,

17, 39]. Nosso método é um avanço sobre o algoritmo de síntese baseada em ciclos (CBS) [25]. Além de usar portas totalmente controladas—portas com controles em todas as linhas exceto na linha alvo—também incluímos portas parcialmente controladas [12].

Uma contribuição importante do nosso método é a busca inicial de portas de baixo custo. Para 3 bits, os circuitos obtidos pelo nosso novo algoritmo de síntese usam menos portas quando comparados com os circuitos obtidos pelo Algoritmo CBS, e mesmo quando comparados aos circuitos obtidos pelos principais algoritmos presentes na literatura, por exemplo ZHU *et al.* [40] e CHENG *et al.* [3] que usam portas Toffoli parcialmente controladas com controles positivos e negativos. Trabalhos anteriores [5, 24, 37] mostraram que bibliotecas de portas que incluem portas Toffoli com controles negativos podem ser mais eficientes para a síntese de circuitos. Nosso algoritmo proposto e algoritmos anteriores encontrados em MASLOV e DUECK [17], MASLOV *et al.* [19], RIBEIRO *et al.* [25] usam a distância de permutações entre a permutação de entrada e a permutação identidade, e procura minimizar tal distância fazendo movimentos (atribuições de portas). Começamos com portas com poucos controles e então aumentamos progressivamente o número de controles até que finalmente tenhamos portas totalmente controladas, onde podemos aplicar o algoritmo CBS [25], que explora propriedades da representação por ciclos das permutações. Uma representação alternativa por ciclos foi usada em SAEEDI *et al.* [28], onde os autores consideram a permutação como produto de pequenos ciclos não necessariamente disjuntos e exploram as propriedades de alguns tipos de produtos, tais como produtos de transposições, por exemplo.

Conforme o survey sobre síntese e otimização de circuitos reversíveis [27], uma síntese pode ser executada de forma ótima ou heurística. Comparamos nossos resultados heurísticos para a síntese de todas as funções bijetivas de 3 bits com os principais resultados atuais usando a biblioteca Toffoli Generalizada obtidos por ZHU *et al.* [40] e CHENG *et al.* [3]. O algoritmo de síntese baseado na decomposição por ciclos de ZHU *et al.* [40] é específico para funções bijetivas de 3 bits e sua ideia principal consiste em representar a permutação através de ciclos disjuntos e então usar o chamado "Head-Pointer-Adjust" que toma sobre todos os ciclos a maior distância de Hamming entre dois elementos consecutivos. Por outro lado, o algoritmo de CHENG *et al.* [3] também é específico para funções bijetivas de 3 bits e usa templates e método bidirecional. É importante mencionar que diferentemente dos algoritmos de ZHU *et al.* [40] e CHENG *et al.* [3], nosso algoritmo funciona para funções bijetivas arbitrárias de n bits. Além disso, obtém circuitos com menos portas em média, um modelo de custo adequado quando avaliamos circuitos reversíveis.

Existem algoritmos ótimos restritos a 3 bits [37] e a 4 bits [33]. Estes métodos principalmente formulam o problema de síntese como uma sequência de problemas

de decisão básicos, tais como Satisfatibilidade Booleana. Observamos que somente um número pequeno de bits podem ser manipulados por estes métodos.

Além disso, executamos uma série de experimentos usando funções bijetivas como *benchmark*. Comparamos os resultados obtidos pelo nosso algoritmo de síntese com os melhores resultados disponíveis na literatura, por ZAKABLUKOV [39] e MASLOV *et al.* [18, 19], para o problema de contagem de portas.

O presente trabalho está organizado da seguinte forma. Nas Seções 1.1 e 1.2, apresentamos resumidamente os conceitos-chave da computação reversível. Na Seção 2.1, apresentamos nosso algoritmo de síntese reversível. Na Seção 2.2, apresentamos os resultados experimentais que obtivemos. Na Seção 3.1, apresentamos regras de otimização visando um novo algoritmo para otimização pós-síntese de circuitos reversíveis. Na Seção. 3.2, apresentamos uma proposta de algoritmo de otimização pós-sintese. Finalmente, no Capítulo 4, apresentamos nossas considerações finais e propomos outras questões relacionadas.

Como forma de divulgação, o algoritmo de síntese reversível encontra-se documentado em artigo aceito para publicação no periódico Journal of Universal Computer Science, que constitui o apêndice da tese. Sobre o algoritmo de otimização pós-sintese, um resumo estendido foi selecionado para apresentação no Encontro de Teoria da Computação realizado no Congresso anual da Sociedade Brasileira de Computação.

1.1 Portas Reversíveis

Seguiremos as definições e a notação usadas nas duas teses de doutorado anteriores sobre o tema [13, 26]. Uma porta lógica que implementa uma função não-inversível (não bijetiva) é dita *irreversível*. Assim, por exemplo, todas as portas lógicas que possuam mais bits de entrada do que de saída são irreversíveis. Isso acontece, por exemplo, com as portas AND, OR e XOR padrão, entre outras, que transformam duas ou mais entradas em uma única saída.

Existem portas que implementam funções bijetivas $f(x_0, \ldots, x_{n-1})$ e são chamadas *portas reversíveis*, ou seja, tais portas computam a função f, dada a entrada $x_0, \ldots, x_{n-1}$. Para cada porta reversível g, existe uma porta g^{-1} que realiza a transformação inversa. Das portas clássicas convencionais de até 2 bits, apenas a porta NOT é reversível. Portas irreversíveis podem ser transformadas em reversíveis, com alguns ajustes e incluindo bits extras [2, 13].

As *portas Toffoli com Múltiplos Controles* (MCT) são algumas portas reversíveis importantes [12, 34]. Essas portas, também conhecidas como $\mathrm{C}^k\mathrm{NOT}(x_{e_0}, \ldots, x_{e_k})$, onde $0 \leq e_0, \ldots, e_k < n$, mantêm $k < n$ linhas relacionadas a $x_{e_0}, \ldots, x_{e_{k-1}}$ (as linhas de controle) inalteradas e inverte o bit na linha relacionada a x_{e_k} (a linha alvo) se e

somente se todas as k linhas de controle contiverem o valor 1.

Como exemplo, a Figura 1.1 mostra uma porta $\mathrm{C}^3\mathrm{NOT}(x_0, x_1, x_3, x_2)$ que mantém as linhas relacionadas a x_0, x_1 e x_3 inalteradas e inverte o bit alvo na linha relacionada a x_2 se e somente se todas as 3 linhas de controle contiverem o valor 1. Neste exemplo e no restante do texto, a linha inferior de uma representação de uma porta ou de um circuito denota o bit mais significativo. Em particular, para $k = 0$, 1 e 2 temos a porta NOT, porta CNOT, também conhecida como porta Feynman [8] e porta Toffoli [34], respectivamente. Essas três portas compõem a biblioteca NCT, que é um conjunto universal de portas para a computação clássica reversível [34], usadas com frequência na computação quântica [21]. Essas três portas reversíveis são mostradas na Figura 1.2.

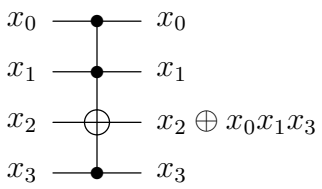

Figura 1.1: Porta $\mathrm{C}^3\mathrm{NOT}(x_0, x_1, x_3, x_2) = (x_0, x_1, x_3, x_2 \oplus x_0x_1x_3)$. A linha inferior denota o bit mais significativo.

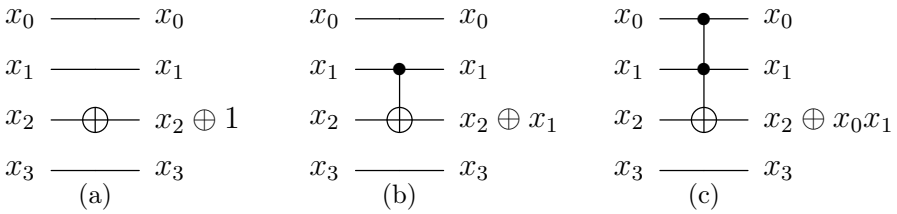

Figura 1.2: Representação das portas (a) $\mathrm{NOT}(x_2) = x_2 \oplus 1$, (b) $\mathrm{CNOT}(x_1, x_2) = (x_1, x_2 \oplus x_1)$ e (c) $\mathrm{Toffoli}(x_0, x_1, x_2) = (x_0, x_1, x_2 \oplus x_0x_1)$, que formam a biblioteca NCT.

As portas MCT, com controles positivos e negativos, chamadas aqui de *portas Toffoli generalizadas* (GT, do inglês *Generalized Toffoli*), permitem que controles positivos e negativos invertam o bit na linha alvo. Essas portas invertem o bit na linha alvo se e somente se cada linha de controle positivo (ou negativo) possuir o valor 1 (ou 0). Indicamos uma linha de controle negativo com $'$ após o rótulo do respectivo parâmetro. Os valores nas linhas restantes (sem controle) não afetam nenhuma saída. A biblioteca de portas utilizada neste trabalho é composta pelas portas Toffoli generalizadas. Se $k = n - 1$, temos as portas Toffoli generalizadas que são totalmente controladas, chamadas portas GT totalmente controladas. Por outro lado, se $k < n - 1$, temos as portas Toffoli generalizadas que são parcialmente controladas, chamadas portas GT parcialmente controladas. Veja a Figura 1.3

para um exemplo de uma porta GT totalmente controlada com controles positivos ou negativos nas três primeiras linhas e alvo na última linha, sendo denotada por $\mathrm{C}^3\mathrm{NOT}(x_0', x_1, x_2', x_3)$.

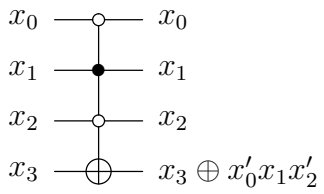

Figura 1.3: Porta GT totalmente controlada representando $\mathrm{C}^3\mathrm{NOT}(x_0', x_1, x_2', x_3) = (x_0', x_1, x_2', x_3 \oplus x_0' x_1 x_2')$.

1.2 Circuitos Reversíveis

No modelo combinacional, um circuito usando portas lógicas elementares AND, OR ou NOT pode ser usado para implementar uma função $f : \{0,1\}^n \to \{0,1\}^m$, onde n e m denotam o número de bits necessários para representar o tamanho de entrada e saída, respectivamente. Se f é uma função bijetiva, então ela pode ser representada por uma permutação com 2^n valores. Caso contrário, a função f pode ser transformada em uma nova função bijetiva, adicionando bits auxiliares. Por exemplo, a função $f = \{(0,7),(1,4),(2,1),(3,0),(4,3),(5,2),(6,6),(7,5)\}$ pode ser representada pela permutação $[7,4,1,0,3,2,6,5]$.

Um *circuito reversível* com n bits é definido como uma sequência de portas reversíveis, cada uma atuando em no máximo n linhas. Este tipo de circuito pode ser usado para implementar funções bijetivas. Mais especificamente, uma vez que cada porta lógica elementar AND, OR ou NOT pode ser simulada por um número constante de NOT, CNOT e portas Toffoli, os circuitos reversíveis formam um modelo computacional equivalente a circuitos combinacionais.

A concatenação de portas de um circuito é equivalente a realização da composição de permutações associadas a cada porta concatenada na mesma ordem. Nesse modelo de circuitos reversíveis, a distribuição das portas segue a convenção clássica de circuitos: a entrada é desenhada à esquerda e as portas do circuito são apresentadas em ordem de aplicação da esquerda para a direita [12].

Definição 1.1. A *distância de Hamming* $d_H(i,j)$ entre dois valores i e j na representação binária é o número de posições de bits que diferem em seus valores. Dois valores i e j são *adjacentes* se $d_H(i,j) = 1$.

Uma representação equivalente para portas GT totalmente controladas é $X(i,j)$, para indicar que troca os valores i e j, onde i e j devem ser adjacentes. Também

podemos entender a notação $X(i,j)$ como segue. Considerando $i < j$ e t a posição do bit em que eles diferem, sejam $b_{n-1}\cdots b_{t+1}\mathbf{0}b_{t-1}\cdots b_0$ e $b_{n-1}\cdots b_{t+1}\mathbf{1}b_{t-1}\cdots b_0$ as representações binárias de i e j, respectivamente, onde b_{n-1} é o bit mais significativo. Em um circuito de n bits, a porta $X(i,j)$ é equivalente a $\mathrm{C}^{n\text{-}1}\mathrm{NOT}(b_0,\ldots,b_{t-1}, b_{t+1},\ldots,b_{n-1},b_t)$. Observe que a linha alvo deve ser indicada como o último parâmetro em uma porta $\mathrm{C}^{n\text{-}1}$NOT, e a ordem dos outros parâmetros que indicam as linhas de controle, não é relevante. Ao longo do texto, usaremos os parâmetros de controle em ordem crescente dos índices.

Na Figura 1.3 apresentada acima, vimos um exemplo de uma porta GT totalmente controlada representando $\mathrm{C}^3\mathrm{NOT}(x'_0,x_1,x'_2,x_3) = (x'_0,x_1,x'_2,x_3\oplus x'_0x_1x'_2)$. Esta porta muda a sequência $x_3x_2x_1x_0 = 0010$ para 1010 (*em representação binária*) e vice versa. Uma representação equivalente para esta porta é $X(2,10)$. Veremos agora a Figura 1.4 para um exemplo de um circuito reversível que transforma a permutação identidade em uma pretendida permutação π. Para esclarecer nosso uso de representações equivalentes, incluímos a Tabela 1.1 correspondente com a sequência explícita de portas usadas. A escolha do exemplo é justificada na Seção 2.1.3, onde apresentamos nas Figuras 2.2 e 2.3 alguns circuitos equivalentes otimizados retornados por nosso algoritmo proposto.

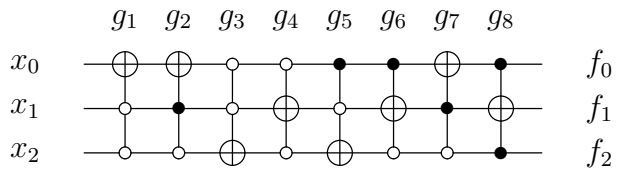

Figura 1.4: Exemplo de um circuito reversível que transforma a permutação identidade em $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ usando apenas portas GT totalmente controladas. A sequência de portas (lida da esquerda para a direita) é $g_1 = X(0,1)$, $g_2 = X(2,3)$, $g_3 = X(0,4)$, $g_4 = X(0,2)$, $g_5 = X(1,5)$, $g_6 = X(1,3)$, $g_7 = X(2,3)$ e $g_8 = X(5,7)$. Consideramos a representação dos números com x_0 sendo o bit menos significativo e x_2 sendo o bit mais significativo. Os bits de saída são denotados por $f_2f_1f_0$.

Definição 1.2. Sejam π e σ permutações com n elementos cada. A distância de *permutações* $d_H(\pi,\sigma)$ é a soma das distâncias de Hamming $d_H(\pi_i,\sigma_i)$, onde π_i e σ_i são os elementos na posição i de π e σ respectivamente, para $0 \le i < n$. Denotamos por $d_H(\pi)$ a distância de permutações entre π e a identidade ι.

Por exemplo, a distância de permutações $d_H(\pi)$ entre a permutação $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ e a identidade $\iota = [0\ 1\ 2\ 3\ 4\ 5\ 6\ 7]$ é a soma $3+2+2+2+3+3+0+1 = 16$. Um circuito que transforma a identidade em π é dado na Fig. 1.4.

Tabela 1.1: Sequência de portas $g_1 = X(0,1)$, $g_2 = X(2,3)$, $g_3 = X(0,4)$, $g_4 = X(0,2)$, $g_5 = X(1,5)$, $g_6 = X(1,3)$, $g_7 = X(2,3)$ e $g_8 = X(5,7)$ que transforma a permutação identidade em $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ usando apenas portas GT totalmente controladas (Veja Fig. 1.4). Em negrito, as mudanças nos bits introduzidas pelas portas correspondentes. O resultado da aplicação da porta na parte inferior de cada coluna é mostrado na próxima coluna durante a leitura da esquerda para a direita.

	$x_2x_1x_0$								$f_2f_1f_0$
	000	00**1**	001	001	001	**1**01	101	101	1**1**1 = 7
	001	00**0**	000	**1**00	100	100	100	100	100 = 4
	010	010	01**1**	011	011	011	0**0**1	001	001 = 1
	011	011	01**0**	010	0**0**0	000	000	000	000 = 0
	100	100	100	**0**00	0**1**0	010	010	01**1**	011 = 3
	101	101	101	101	101	**0**01	0**1**1	01**0**	010 = 2
	110	110	110	110	110	110	110	110	110 = 6
	111	111	111	111	111	111	111	111	1**0**1 = 5
Portas:	$g_1 \nearrow$	$g_2 \nearrow$	$g_3 \nearrow$	$g_4 \nearrow$	$g_5 \nearrow$	$g_6 \nearrow$	$g_7 \nearrow$	$g_8 \nearrow$	

Capítulo 2

Síntese de Circuitos Reversíveis

Uma vez que os algoritmos de síntese de circuitos irreversíveis não podem ser aplicados diretamente a circuitos reversíveis devido a diversas restrições, torna-se necessária a criação de algoritmos de síntese de circuitos reversíveis, ou seja, algoritmos de síntese que produzem circuitos formados por portas reversíveis para funções bijetivas dadas como entrada [9, 14, 18, 20, 27, 29]. Enunciamos então formalmente o principal problema estudado.

O *problema de síntese de circuito reversível* é definido como, dada uma função bijetiva $f : \{0,1\}^n \to \{0,1\}^n$, encontre um circuito reversível com n bits que calcula f. A entrada do problema é uma permutação π de S_{2^n} que representa a função bijetiva f e a saída é o circuito reversível que transforma a permutação identidade em π. Nosso objetivo é encontrar o menor circuito reversível possível usando portas Toffoli Generalizadas.

Nesse capítulo, nosso algoritmo de síntese reversível é apresentado, bem como os resultados experimentais obtidos. Para isso, introduzimos alguns conceitos e propriedades da representação das permutações por ciclos.

Seja $\pi = [\pi_0\ \pi_1\ \cdots\ \pi_{2^n-1}]$ uma permutação e seja $\pi = C_1C_2\cdots C_m$ sua representação por ciclos disjuntos, isto é, ciclos sem elementos comuns. Dizemos que uma permutação π é *unicíclica* se puder ser representada por um único ciclo. Cada ciclo C pode ser representado por $C = (c^1c^2\cdots c^{|C|})$, onde $c^{i+1} = \pi_{c^i}$ com $c^0 = c^{|C|}$. Um ciclo com comprimento 2 é chamado de *transposição*. Conforme Definição 1.1, no caso $c^i = \pi_j$, temos $d_H(c^{i-1}, c^i) = d_H(j, \pi_j)$. Denotamos por $d_H(\pi_j)$ a distância de Hamming entre π_j e j, e usaremos ambos os nomes $d_H(c^{i-1}, c^i)$ e $d_H(\pi_j)$ indistintamente.

Dado um ciclo C de comprimento $|C|$ em uma permutação π, definimos a soma das distâncias de Hamming entre elementos consecutivos do ciclo C como

$$S(C) = \sum_{i=1}^{|C|} d_H(c^{i-1}, c^i), \tag{2.1}$$

onde $c^0 = c^{|C|}$. Decorre imediatamente da definição que

$$d_H(\pi) \equiv d_H(\pi, \iota) = \sum_{\ell=1}^{m} S(C_\ell), \tag{2.2}$$

onde m é o número de ciclos de π.

Por exemplo, para a permutação π dada na Figura 1.4 a representação cíclica é (0 7 5 2 1 4 3)(6), e $d_H(\pi) = 16 + 0 = 16$.

O *número de portas* (GC, do inglês *gate count*) tem sido utilizado na literatura como medida para avaliar as abordagens de síntese de circuitos reversíveis [3, 11, 18–20, 29, 39, 40]. Para um circuito reversível arbitrário $\mathcal{C}$ que consiste numa sequência $g_1, g_2, \ldots, g_k$ de k portas reversíveis, a métrica número de portas é definida como $gc(\mathcal{C}) \equiv k$. A noção de *custo quântico* (QC, do inglês *quantum cost*) para medir o custo da implementação dos circuitos reversíveis também é muito utilizada na literatura [11, 17, 18, 38, 39]. Mais precisamente, o custo quântico é definido como o número de operações elementares quânticas necessárias para implementar uma porta. Para uma porta reversível arbitrária g que pode ser decomposta em l portas quânticas elementares, a métrica do seu custo quântico é definida como $qc(g) \equiv l$. O custo quântico para um circuito reversível $\mathcal{C}$ é definido como $qc(\mathcal{C}) = \Sigma_{g_i \in \mathcal{C}} qc(g_i)$ [11, 17, 26]. Neste trabalho, utilizaremos somente o número de portas como métrica para avaliar nossa abordagem de síntese de circuitos reversíveis.

2.1 Algoritmo

Nosso algoritmo é dividido em duas partes. A primeira parte, descrita na Subseção 2.1.1, consiste em buscar portas GT parcialmente controladas com o objetivo de encontrar a porta que diminui ao máximo a distância de permutações (entre a permutação atual e a identidade), aumentando progressivamente o número de controles. A permutação associada a cada porta é obtida em uma etapa de pré-processamento e armazenada em uma matriz. A segunda parte, descrita na Subseção 2.1.2, consiste em tentar aplicar apenas portas GT totalmente controladas. Finalmente, na Subseção 2.1.3, nosso algoritmo de síntese com aumento progressivo de controles em portas GT é descrito usando os algoritmos obtidos na primeira e segunda partes.

2.1.1 Primeira Parte: Portas GT parcialmente controladas

O objetivo é diminuir a distância de permutações entre a permutação atual e a identidade, usando o mínimo de controles possíveis para as portas Toffoli generalizadas no início do algoritmo. Assim, primeiro tentamos as portas NOT, enquanto for possível diminuir a distância de permutações, depois tentamos as portas CNOT,

e então, tentamos as portas com dois controles, e assim por diante, até chegarmos às portas GT totalmente controladas.

Em uma parte de pré-processamento, construímos uma matriz com todas as portas lógicas possíveis para a biblioteca Toffoli generalizada para circuitos de n bits. O número de portas GT possíveis é $n3^{n-1}$. De fato, para cada linha que o bit alvo ocupa entre as n linhas possíveis, as $n-1$ linhas restantes podem ser ocupadas de três maneiras diferentes: controle positivo, controle negativo ou ausente. A construção da matriz é feita na ordem inversa dos testes, ou seja, começa com as portas GT totalmente controladas, isto é, portas Toffoli com $n-1$ controles. A partir dessas portas, construímos portas Toffoli com $n-2$ controles, que por sua vez, são usadas para construir as portas Toffoli com $n-3$ controles e assim por diante.

As permutações associadas às portas GT totalmente controladas são da forma $(i\ j)$ na notação cíclica, que corresponde a uma transposição. Por sua vez, as permutações associadas às portas GT com $n-2$ controles são da forma $(i\ j)(k\ l)$, enquanto as permutações associadas às portas GT com $n-3$ controles são formadas por quatro transposições e assim por diante.

Apresentamos na Figura 2.1 um exemplo para 4 bits, onde a composição de duas portas GT totalmente controladas correspondentes a (4 12) e (5 13) é equivalente a uma porta GT com 2 controles, a saber, (4 12)(5 13).

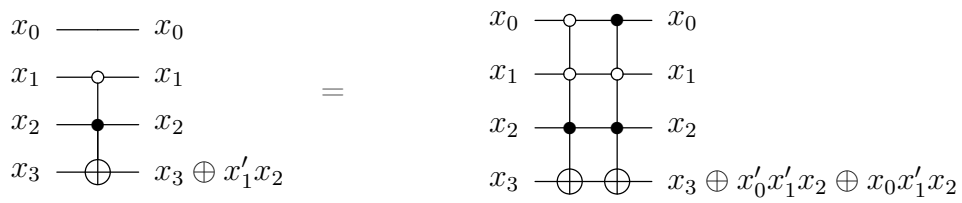

Figura 2.1: A porta $\mathrm{C^2NOT}(x_1', x_2, x_3)$ é equivalente à composição das portas $\mathrm{C^3NOT}(x_0', x_1', x_2, x_3)$ e $\mathrm{C^3NOT}(x_0, x_1', x_2, x_3)$.

Dizemos que uma transformação T sobre uma permutação π é um *d-move* se $d_H(\pi) - d_H(T(\pi)) = d$, ou equivalentemente, $d_H(\pi, T(\pi)) = d$.

Lema 2.1. *Se T é um d-move correspondente a uma porta* GT*, com n bits e c controles, onde $0 \leq c < n$, então $|d| \leq 2^{n-c}$ e $|d|$ é par.*

Demonstração. Seja π uma permutação. Para cada d-move T correspondente a uma porta GT, com n bits e c controles, onde $0 \leq c < n$, temos $d_H(\pi \oplus \sigma, 0) = 2^{n-c}$, onde $\sigma = T(\pi)$ e $\oplus$ é a operação XOR bit a bit. Portanto, exatamente 2^{n-c} bits (em elementos distintos) são trocados por T. Suponha que entre os 2^{n-c} bits trocados por T houvesse q bits que antes da troca diferiam dos bits correspondentes da permutação identidade ι. Ao efetuar a troca, tais q bits coincidirão com os

bits correspondentes de ι. Por outro lado, os $p = 2^{n-c} - q$ bits que inicialmente coincidiram com os bits correspondentes em ι agora serão diferentes. Portanto, d-move será tal que $d = q - (2^{n-c} - q) = 2(q - 2^{n-c-1})$. Portanto, d é par e $|d| = |q - p| = |2(q - 2^{n-c-1})| \leq 2^{n-c}$. □

Assim, em particular, se T é uma transformação correspondente a uma porta GT totalmente controlada, então T é um 2-move, ou um 0-move, ou um -2-move.

Por exemplo, seja $\pi = (0\ 7\ 6\ 1\ 2\ 11\ 13\ 12\ 3\ 15\ 14\ 8\ 10\ 5\ 9\ 4)$ uma permutação unicíclica. A porta GT parcialmente controlada $\mathrm{C^2NOT}(x_1, x_3', x_0)$, que corresponde a $(2\ 3)(6\ 7)$ aplicada a π, retorna $\sigma = (0\ 6\ 1\ 3\ 15\ 14\ 8\ 10\ 5\ 9\ 4)(2\ 11\ 13\ 12)(7)$ e $d_H(\pi) - d_H(\sigma) = 34 - 30 = 4$, então a porta $\mathrm{C^2NOT}(x_1, x_3', x_0)$ aplicada a π é um 4-move. A porta $\mathrm{C^2NOT}(x_1, x_3', x_0)$ aplicada a σ é um -4-move. A porta GT parcialmente controlada $\mathrm{C^2NOT}(x_1', x_3, x_0)$, que corresponde a $(8\ 9)(12\ 13)$ aplicada a π, retorna $\lambda = (0\ 7\ 6\ 1\ 2\ 11\ 12\ 3\ 15\ 14\ 9\ 4)(5\ 8\ 10)(13)$ e $d_H(\pi) - d_H(\lambda) = 34 - 36 = -2$, então a porta $\mathrm{C^2NOT}(x_1', x_3, x_0)$ aplicada a π é um -2-move. A porta $\mathrm{C^2NOT}(x_1', x_3, x_0)$ aplicada a λ é um 2-move.

Definição 2.1. Um d-move é um *movimento útil* se e somente se $d \geq 0$.

Portanto, na primeira parte de nosso algoritmo para síntese de circuitos reversíveis, usamos as portas GT parcialmente controladas. Inicialmente, aplicamos portas GT com poucos controles, começando com portas NOT. Primeiro, todas as portas NOT são testadas. A porta NOT que resultar no maior d-move será aplicada, onde d é par e $0 < d \leq 2^n$. Este procedimento é aplicado enquanto a distância de permutações entre a permutação atual e a identidade diminuir. Em seguida, todas as portas CNOT são testadas e a porta CNOT que resultar no maior d-move é aplicada, onde d é par e $0 < d \leq 2^{n-1}$. Novamente, tal procedimento será aplicado enquanto a distância de permutações entre a permutação atual e a identidade diminuir. O mesmo procedimento é aplicado para as portas $\mathrm{C^2NOT}$ e para todas as outras portas com mais controles, até que apenas as portas GT totalmente controladas sejam aplicadas na segunda parte do algoritmo. Nesse caso, o d-move aplicado é tal que $d = 0$ ou $d = 2$.

O Algoritmo 1 abaixo é a primeira parte do nosso algoritmo principal, sua biblioteca é composta apenas de portas GT com no máximo $n - 2$ controles. Denotamos por $L(c)$ a biblioteca de portas Toffoli com exatamente c controles.

2.1.2 Segunda Parte: Portas GT totalmente controladas

Os próximos lemas serão usados na segunda parte do nosso novo algoritmo, quando apenas portas GT totalmente controladas são usadas.

Algoritmo 1: Primeira Parte: Portas GT parcialmente controladas

Entrada: uma permutação π_0 representando uma função bijetiva.
Saída: uma permutação π tal que $d_H(\pi) \leq d_H(\pi_0)$ e um circuito $\mathcal{C}$ que transforma π_0 em π.

```
1  π ← π_0;
2  C ← ∅;
3  para c ← 0 até n − 2 faça
4  |  continue ← True;
5  |  enquanto continue faça
6  |  |  d_max ← 0;
7  |  |  para g ∈ L(c) faça
8  |  |  |  d ← d_H(π) − d_H(g(π));
   |  |  |  /* Ver Lema 2.1                                   */
9  |  |  |  se d > d_max então
10 |  |  |  |  d_max ← d;
11 |  |  |  |  g_max ← g;
12 |  |  |  fim
13 |  |  fim
14 |  |  se d_max = 0 então
15 |  |  |  continue ← False;
16 |  |  senão
17 |  |  |  atualize π e adicione g_max em C;
18 |  |  fim
19 |  fim
20 fim
21 retorna π e C;
```

Lema 2.2. *Seja σ a permutação resultante da aplicação da porta $X(i,j)$ a π, e seja $m(\sigma)$ o número de ciclos em sua representação cíclica. Se i e j estão no mesmo ciclo em π, então $m(\sigma) = m(\pi) + 1$. Caso contrário, $m(\sigma) = m(\pi) - 1$.*

Demonstração. No caso de $i = c^s$ e $j = c^t$ pertencerem ao mesmo ciclo C, com $s < t$, a porta $X(i,j)$ divide C em dois ciclos, um definido pela seqüência de elementos $c^s, \ldots, c^{t-1}$, e o outro definido pelos elementos restantes de C. Por outro lado, no caso de $i = c_1^s$ e $j = c_2^t$ pertencerem a ciclos distintos C_1 e C_2, a porta $X(i,j)$ junta C_1 e C_2 em um ciclo C definido pela sequência de modo que c_1^{s-1} precede c_2^t, c_2^{t-1} precede c_1^s e os outros elementos permanecem de acordo com a ordem de C_1 e C_2. □

Dados três elementos i, j e k de uma permutação, dizemos que k está em um *caminho mínimo* entre i e j se $d_H(i,j) = d_H(i,k) + d_H(k,j)$. Além disso, observe que se $d_H(i,j) = 1$, então para cada elemento k, ou i está em um caminho mínimo entre k e j, ou j está em um caminho mínimo entre i e k. Observe que, no caso dos valores i e j serem adjacentes, temos $d_H(i,j) = 1$ e para todos os outros elementos k, $|d_H(i,k) - d_H(j,k)| \leq 1$.

Lema 2.3. *Seja π uma permutação tal que π_i e π_j sejam valores adjacentes e considere as seguintes situações possíveis: (a) π_j está em um caminho mínimo entre i e π_i; (b) π_i está em um caminho mínimo entre j e π_j. Se ambas as condições (a) e (b) forem satisfeitas simultaneamente, então a porta $X(\pi_i, \pi_j)$ aplicada a π é um 2-move. Se apenas uma única condição (a) ou (b) for satisfeita, então a porta $X(\pi_i, \pi_j)$ aplicada a π é um 0-move. Se nem a condição (a) nem a condição (b) forem satisfeitas, então a porta $X(\pi_i, \pi_j)$ aplicada a π é um -2-move.*

Demonstração. Seja π' a permutação resultante da aplicação de $X(\pi_i, \pi_j)$ à permutação π. Os únicos elementos modificados pela aplicação desta porta são π_i e π_j, portanto $d_H(\pi, \pi') = d_H(i, \pi_j) + d_H(j, \pi_i) - d_H(i, \pi_i) - d_H(j, \pi_j)$.

Primeiro, consideremos o caso em que ambas as condições (a) e (b) são satisfeitas simultaneamente. Neste caso, $d_H(i, \pi_i) = d_H(i, \pi_j) + d_H(\pi_j, \pi_i)$ e $d_H(j, \pi_j) = d_H(j, \pi_i) + d_H(\pi_i, \pi_j)$. Portanto, $d_H(i, \pi_i) = d_H(i, \pi_j) + 1$ and $d_H(j, \pi_j) = d_H(j, \pi_i)$ $+1$. Assim, $d_H(\pi, \pi') = 2$, indicando que $X(\pi_i, \pi_j)$ é um 2-move.

Agora, consideremos o caso em que a condição (a) é satisfeita e a condição (b) não o é. Neste caso, $d_H(i, \pi_i) = d_H(i, \pi_j) + d_H(\pi_j, \pi_i)$ e π_j está no caminho mínimo entre j e π_i. Então, $d_H(j, \pi_i) = d_H(j, \pi_j) + d_H(\pi_j, \pi_i)$. Portanto, $d_H(i, \pi_j) = d_H(i, \pi_i) - 1$ e $d_H(j, \pi_i) = d_H(j, \pi_j) + 1$. Assim, $d_H(\pi, \pi') = 0$, indicando que $X(\pi_i, \pi_j)$ é um 0-move.

Omitimos a análise dos casos restantes porque são provados de forma análoga. □

Lema 2.4. *Se $\pi \neq \iota$ não é unicíclico, então existe um movimento útil correspondente a alguma porta* GT *totalmente controlada que une ciclos.*

Demonstração. A única permutação que não permite um 0-move nem um 2-move é a identidade. Consideremos uma permutação não unicíclica $\pi \neq \iota$ que não permite um 2-move. Isso implica que π tem mais de um ciclo e há pelo menos um ciclo com dois elementos distintos. Então, há um ciclo em π no qual existem dois elementos consecutivos c^i e c^{i+1} tal que existe um elemento c' em outro ciclo que pertence a um caminho mínimo entre c^i e c^{i+1}. □

Por exemplo, a porta $X(2,3)$ aplicada a $\pi = (0\ 3\ 1)(2)$ retorna $\lambda = (0\ 2\ 3\ 1)$.

Lema 2.5. *Seja $C = (c^1 c^2 \cdots c^{|C|})$ um ciclo com no máximo um par de elementos vizinhos que não são adjacentes. Então existe uma sequência de $|C| - 1$ portas* GT *totalmente controladas que transforma este ciclo em $|C|$ ciclos unitários.*

Demonstração. Sem perda de generalidade, considere um ciclo $C = (c^1 c^2 \cdots c^{|C|})$ tal que o único par não adjacente possível seja $(c^{|C|} c^1)$. Aplicando a porta GT totalmente controlada $X(c^1, c^2)$, este ciclo C é dividido em dois ciclos $C_1 = (c^1)$ e $C' = (c^2 \cdots c^{|C|})$. Depois de aplicar a sequência de portas $X(c^2, c^3)$, $X(c^3, c^4), \ldots, X(c^{|C|-1}, c^{|C|})$, obtemos os ciclos unitários $(c^1), (c^2), \ldots, (c^{|C|})$. □

Por exemplo, seja $\pi = (0\ 1\ 3\ 7\ 6)(2)(4)(5)$. Então, a sequência de portas $X(0,1)$, $X(1,3)$, $X(3,7)$ e $X(6,7)$ transforma π em ι.

Dado um ciclo C de comprimento $|C|$ em uma permutação π, definimos o quociente

$$P(C) = \frac{S(C)}{|C|}, \tag{2.3}$$

e também,

$$P(\pi) = \sum_{i=1}^{m} P(C_i). \tag{2.4}$$

Por exemplo, seja $\pi = C_1 C_2 = (0\ 3\ 1)(2)$. Temos que $S(C_1) = 4$ e $S(C_2) = 0$, portanto $P(C_1) = 1.33$, $P(C_2) = 0$ e $P(\pi) = 1.33$.

Lema 2.6. *Seja σ a permutação resultante da aplicação da porta $X(i,j)$ a π. Se $X(i,j)$ é um movimento útil que junta ciclos em π, então temos que $P(\sigma) < P(\pi)$.*

Demonstração. Seja $X(i,j)$ uma porta d-move que pode ser aplicada a π tal que $i \in C'$ e $j \in C'' \neq C'$, criando um novo ciclo C''' com a união dos dois ciclos. Seja $P(C') = S(C')/|C'|$, $P(C'') = S(C'')/|C''|$ e $P(C''') = S(C''')/|C'''|$. Como

$S(C''') = S(C') + S(C'') - d$ e $|C'''| = |C'| + |C''|$, temos que

$$\begin{aligned} P(C''') &= \frac{S(C') + S(C'') - d}{|C'| + |C''|} \\ &= \frac{P(C') \cdot |C'|}{|C'| + |C''|} + \frac{P(C'') \cdot |C''|}{|C'| + |C''|} - \frac{d}{|C'| + |C''|}. \end{aligned} \tag{2.5}$$

Considerando que $|C'| < |C'| + |C''|$ e $|C''| < |C'| + |C''|$, se $d = 0$ ou $d = 2$, então temos que $P(C''') < P(C') + P(C'')$. Como os únicos ciclos modificados em π foram C' e C'', temos que $P(\sigma) = P(\pi) + P(C''') - P(C') - P(C'')$. Portanto $P(\sigma) < P(\pi)$. □

Definição 2.2. Uma sequência de movimentos úteis é uma *sequência útil* se diminuir $d_H(\pi)$ ou $P(\pi)$.

Para finalizar a análise da convergência do Algoritmo 2, precisamos ver o caso em que a permutação é unicíclica e não há 2-move. Para este objetivo, usamos os próximos procedimentos e lemas.

Procedimento 1: Replace(i, j)

Entrada: $i = i_{n-1} \cdots i_0, j = j_{n-1} \cdots j_0 \in \pi$ $\{i_k$ é o bit k de $i\}$
Saída: $T(\pi) = X(i, j)\pi$.

```
para k ← 0 até n − 1 faça
    se i_k ≠ j_k então
        aplique a porta X(i, i'), onde i' é o elemento adjacente de i tal que
          i'_k = j_k;
        i ← i';
    fim
fim
```

Lema 2.7. *O procedimento 1 coloca j na posição de i em π, passando por todos os elementos de um caminho mínimo entre i e j.*

Demonstração. Em cada etapa do *loop* em que a condição do *se-então* é verdadeira, i' tem os mesmos bits de i, exceto o bit k, e $d_H(i', j) = d_H(i, j) - 1$, portanto i' está em um caminho mínimo entre i e j. Após aplicar a porta $X(i, i')$, i' permanece na posição de π ocupada por i. Após aplicar $d_H(i, j)$ portas, o elemento j ocupa o lugar do i original. O último i' é j. □

Lema 2.8. *Sejam i e j elementos de π, de modo que i seja o sucessor de j em um ciclo, então após Replace(i, j) existe um ciclo unitário (j).*

Demonstração. Se i é o sucessor de j em um ciclo, então i está na posição j ($\pi_j = i$). Portanto, Replace(i, j) coloca j na posição j. □

Procedimento 2: Sequência útil com 2-move

Entrada: Permutação atual π.
Saída: Uma sequência útil com 2-move.

```
1 j ← 2^n − 1;
2 repita
3 |   seja i sucessor de j no respectivo ciclo, Replace(i, j);
4 |   j ← j − 1;
5 até Replace(i, j) tem um 2-move;
```

Lema 2.9. *O Procedimento 2 produz uma sequência útil.*

Demonstração. Se j está em um ciclo unitário, o sucessor de j é j e, portanto, Replace(i, j) não tem d-move, caso contrário, todos os 'movimentos' são movimentos úteis. De fato, em cada porta de Replace(i, j), o presente i' está em um caminho mínimo entre j e π_j então, pelo Lema 2.3, este movimento é um movimento útil.

Este Procedimento converge porque, a cada iteração, os elementos maiores que j estão em ciclos unitários e não pertencem ao caminho mínimo entre i e j usado por Replace(i, j). □

O Algoritmo 2 começa a ser utilizado quando não for possível aplicar portas GT parcialmente controladas que diminuem a distância de permutações entre a permutação atual e a identidade.

Teorema 2.10. *Corretude do Algoritmo 2.*

Demonstração. Em cada etapa, o algoritmo aplica um 2-move ou um 0-move minimizando $P(\pi)$, veja Lemas 2.2 a 2.6 e Lema 2.9. Portanto, a cada etapa, a distância de permutações entre π e ι é mantida constante ou diminui. Quando a distância de permutações é mantida constante, a função $P(\pi)$ diminui até que a permutação seja unicíclica, veja Lemas 2.2, 2.4 e 2.6. Neste caso, o algoritmo aplica uma seqüência útil que aplica um 2-move, veja Lema 2.9. Assim, a permutação sempre converge para a permutação identidade ι. □

Dessa forma, finalizamos a análise da convergência do Algoritmo 2 e podemos concluir nosso método de síntese.

2.1.3 Algoritmo de síntese com aumento progressivo de controles em portas GT

Nosso método de síntese usa portas GT com $0, 1, \ldots, n-1$ controles, aumentando o número de controles progressivamente. O algoritmo de síntese com aumento progressivo de controles em portas GT (Algoritmo 3) sinteticamente usa os Algoritmos 1

Algoritmo 2: Segunda Parte: Portas GT totalmente controladas

Entrada: uma permutação π_0 representando uma função bijetiva.
Saída: o circuito $\mathcal{C}$ que transforma π_0 em ι.

```
1  π ← π_0;
2  C ← ∅;
3  enquanto d_H(π) > 0 faça
4      se existe um 2-move que junta ciclos então
5          escolha X(c^i, j) correspondente ao 2-move;
6          aplique a porta reversível correspondente;
7          atualize π e C;
8      senão se existe um 2-move que divide ciclos então
9          aplique a porta reversível correspondente;
10         atualize π e C;
11     senão se existe uma sequência de portas que divide o ciclo em ciclos
         unitários então
12         aplique a sequência de portas;
13         atualize π e C;
           /* Ver Lema 2.5                                          */
14     senão se existe um 0-move que junta ciclos então
15         escolha X(c^i, j) correspondente ao 0-move;
16         aplique a porta reversível correspondente;
17         atualize π e C;
           /* Diminui necessariamente o valor de P(π), ver Lema 2.6
              */
18     senão
           /* π é unicíclico, ver Lema 2.4                          */
19         aplique o Procedimento 2;
20         atualize π e C;
21     fim
22 fim
23 retorna C;
```

e 2 apresentados nas subseções anteriores. Veja abaixo o Algoritmo 3 para um esboço de nosso método de síntese usando portas com $0, 1, \ldots, n-1$ controles e o Teorema 2.11 que dá a sua corretude.

Teorema 2.11. *Corretude do Algoritmo 3.*

Demonstração. Nas primeiras etapas, para cada quantidade k de controles, $k \in \{0, 1, \ldots, n-2\}$, o algoritmo aplica d-moves ($d > 0$, o maior possível, de acordo com o Lema 2.1), de modo que a distância de permutações entre a permutação atual para a identidade diminui tanto quanto possível. Ao chegar em $k = n-1$ controles, basta usar o Algoritmo 2, que conforme o Teorema 2.10, obtém o circuito que transforma a permutação na identidade. □

A Figura 2.2 descreve o circuito que transforma a identidade em π =

Algoritmo 3: Algoritmo de síntese com aumento progressivo de controles em portas GT

Entrada: uma permutação π_0 representando uma função bijetiva.
Saída: o circuito $\mathcal{C}$ que transforma ι em π_0.

```
1 (π, C1) ← aplique o Algoritmo 1 a π0;
2 C2 ← aplique o Algoritmo 2 a π;
3 C ← reverso(C1C2);
4 retorna C;
```

[7 4 1 0 3 2 6 5], para o caso em que a entrada do Algoritmo 3 é $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, considerado no exemplo da Figura 1.4. Observe que a ordem das portas retornadas é invertida em relação à ordem em que as portas são obtidas.

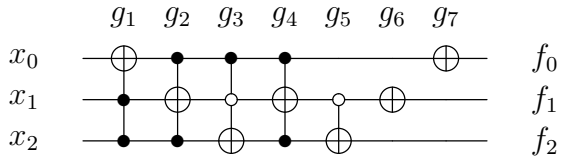

Figura 2.2: Exemplo de um circuito reversível retornado pelo Algoritmo 3, que transforma a permutação identidade em $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. Os bits de saída são denotados por $f_2 f_1 f_0$. A sequência de portas (lida da esquerda para a direita) é $g_1 = \text{Toffoli}(x_1, x_2, x_0)$, $g_2 = \text{Toffoli}(x_0, x_2, x_1)$, $g_3 = \text{Toffoli}(x_0, x_1', x_2)$, $g_4 = \text{Toffoli}(x_0, x_2, x_1)$, $g_5 = \text{CNOT}(x_1', x_2)$, $g_6 = \text{NOT}(x_1)$ e $g_7 = \text{NOT}(x_0)$, onde $g_7 g_6 g_5$ e $g_4 g_3 g_2 g_1$ são os circuitos retornados pelos Algoritmos 1 e 2, respectivamente.

O Algoritmo 3 produz o circuito selecionando portas Toffoli generalizadas, começando pelas portas que serão aplicadas por último, até encontrar as primeiras portas a serem aplicadas e que ficarão à esquerda no nosso modelo de circuito. Como a permutação está associada a uma função bijetiva e os circuitos reversíveis estão associados a funções bijetivas, podemos definir estratégias que usam o Algoritmo 3 para melhorar a contagem de portas.

Estratégia 1 (Síntese da Inversa)**.** *Aplique um algoritmo à permutação de entrada e à sua inversa. Em seguida, escolha o circuito com o menor número de portas.*

Estratégia 2 (Síntese Bidirecional)**.** *Aplique um algoritmo simultaneamente em ambas as direções. Escolha adicionar portas do lado da entrada ou do lado da saída do circuito em cada etapa da síntese.*

A Figura 2.3 mostra os circuitos que transformam a identidade em $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, obtidos com a Estratégia 1 e Estratégia 2, respectivamente, onde a entrada do Algoritmo 3 é a permutação $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. Como fizemos antes,

para esclarecer nosso uso de representações equivalentes, incluímos a Tabela 2.1 correspondente com a sequência explícita de portas usadas correspondendo ao circuito representado na Figura 2.3b.

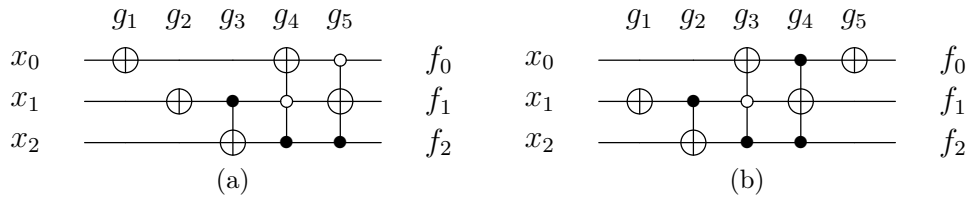

Figura 2.3: Exemplo de circuitos reversíveis obtidos do Algoritmo 3 com (a) Estratégia 1 e (b) Estratégia 2, respectivamente, que transformam a permutação identidade em π = [7 4 1 0 3 2 6 5]. Os bits de saída são denotados por $f_2 f_1 f_0$. (a) A sequência de portas é $g_1 = \text{NOT}(x_0)$, $g_2 = \text{NOT}(x_1)$, $g_3 = \text{CNOT}(x_1, x_2)$, $g_4 = \text{Toffoli}(x_1', x_2, x_0)$ e $g_5 = \text{Toffoli}(x_0', x_2, x_1)$. (b) A sequência de portas é $g_1 = \text{NOT}(x_1)$, $g_2 = \text{CNOT}(x_1, x_2)$, $g_3 = \text{Toffoli}(x_1', x_2, x_0)$, $g_4 = \text{Toffoli}(x_0, x_2, x_1)$ e $g_5 = \text{NOT}(x_0)$.

Tabela 2.1: Sequência de portas $g_1 = \text{NOT}(x_1)$, $g_2 = \text{CNOT}(x_1, x_2)$, $g_3 = \text{Toffoli}(x_1', x_2, x_0)$, $g_4 = \text{Toffoli}(x_0, x_2, x_1)$ e $g_5 = \text{NOT}(x_0)$ que transforma a permutação identidade em π = [7 4 1 0 3 2 6 5] usando o Algoritmo 3 e Estratégia 2, veja o circuito correspondente na Fig. 2.3b. Em negrito, as mudanças nos bits introduzidas pelas portas correspondentes. O resultado da aplicação da porta na parte inferior de cada coluna é mostrado na próxima coluna durante a leitura da esquerda para a direita.

	$x_2x_1x_0$					$f_2f_1f_0$
	000	0**1**0	**1**10	110	110	11**1** = 7
	001	0**1**1	**1**11	111	1**0**1	10**0** = 4
	010	0**0**0	000	000	000	00**1** = 1
	011	0**0**1	001	001	001	00**0** = 0
	100	1**1**0	**0**10	010	010	01**1** = 3
	101	1**1**1	**0**11	011	011	01**0** = 2
	110	1**0**0	100	10**1**	1**1**1	11**0** = 6
	111	1**0**1	101	10**0**	100	10**1** = 5
Portas:	g_1 ↗	g_2 ↗	g_3 ↗	g_4 ↗	g_5 ↗	

2.2 Resultados Experimentais

Em primeiro lugar, consideramos todas as funções bijetivas de 3 bits e comparamos nossos resultados com diferentes algoritmos de síntese de circuitos reversíveis. Os resultados estão na Tabela 2.2. A primeira coluna, GC, mostra o número de portas reversíveis que formam os circuitos que sintetizam as funções dadas. Para as demais colunas, os valores representam a quantidade de funções que são sintetizadas por

circuitos com o número GC de portas reversíveis. Assim, por exemplo, na coluna (a), uma função (identidade) não necessita de nenhuma porta reversível para ser sintetizada, vinte e sete funções necessitam de uma porta reversível para serem sintetizadas, e assim por diante. A linha MP relata o número médio de portas reversíveis necessárias para sintetizar uma função, considerando todas as funções bijetivas de 3 bits. Os principais resultados de algoritmos de síntese de circuitos reversíveis presentes na literatura são apresentados em duas colunas. A coluna **CGWZ** mostra os resultados do algoritmo de simplificação de síntese por CHENG *et al.* [3]. A coluna **ZLZPZ** mostra os resultados do algoritmo otimizado de síntese baseado em decomposição em ciclos por ZHU *et al.* [40]. A coluna **Exato** mostra os resultados de síntese exatos por WILLE *et al.* [37] para todas as funções bijetivas de 3 bits. Nesse caso, faz-se necessário observar que o algoritmo de síntese de circuitos reversíveis de WILLE *et al.* [37] não garante resultados ótimos para mais de 3 bits. Os resultados da aplicação do Algoritmo 3 a todas as funções bijetivas de 3 bits estão nas colunas **(a)**, **(b)** e **(c)** de acordo com diferentes estratégias da seguinte forma:

- Na coluna (a), aplicação do Algoritmo 3.
- Na coluna (b), aplicação do Algoritmo 3 melhorado pela Estratégia 1.
- Na coluna (c), aplicação do Algoritmo 3 melhorado pela Estratégia 1 mais Estratégia 2, seguido pela escolha do valor mínimo entre os dois resultados.

O Algoritmo 3 melhorado considerado na coluna (c) obteve o resultado de 5,23 portas em média, que é o melhor obtido por uma heurística, até onde sabemos. O resultado de 5,38 portas em média obtido pelo algoritmo de CHENG *et al.* [3] também considera a Estratégia 1 para o seu algoritmo de simplificação de síntese, além do uso de templates. Assim, em comparação, obtivemos uma melhora de 2,8%. Nosso Algoritmo 3 considerado na coluna (a) obteve o resultado de 5,82 portas em média, melhorando o resultado de 6,03 portas em média obtido pelo algoritmo de síntese de ZHU *et al.* [40], que também é um algoritmo unidirecional. Nesse caso, em comparação, obtivemos uma melhora de 3,7%. A ideia principal no algoritmo de ZHU *et al.* [40] é decompor os ciclos em permutações e usar o chamado "Head-Pointer-Adjust", que toma a maior distância de Hamming entre os elementos do ciclo. Além disso, os algoritmos de síntese de ZHU *et al.* [40] e de CHENG *et al.* [3] são específicos para 3 bits, enquanto nosso Algoritmo 3 funciona para um número arbitrário de bits. Por fim, vemos que o resultado de 5,23 portas em média obtido pelo nosso Algoritmo 3 está 14,2% acima em relação ao resultado de 4,58 portas em média obtido pela síntese exata de WILLE *et al.* [37].

Tabela 2.2: Número de funções bijetivas usando um número especificado de portas, considerando todas as funções bijetivas de 3 bits para diferentes algoritmos de síntese, conforme indicado pelas citações. Nossos resultados estão listados nas colunas (a), (b) e (c). Os principais resultados atuais usando a biblioteca GT estão nas colunas **CGWZ** [3] e **ZLZPZ** [40]. A síntese exata com circuitos com tamanho mínimo usando a biblioteca GT está na última coluna [37]. A linha MP relata o número médio de portas necessárias para sintetizar um circuito.

	Algoritmo 3			Literatura atual		Exato
GC	(a)	(b)	(c)	**CGWZ**	**ZLZPZ**	**WSPD**
0	1	1	1	1	1	1
1	27	27	27	27	15	27
2	309	369	369	369	129	369
3	1797	2505	2601	2633	753	2925
4	5376	7586	8114	7624	3100	13282
5	9758	12932	12994	11263	8409	20480
6	10529	9940	10066	10258	13405	3236
7	7046	4523	4652	5963	10506	0
8	3922	2166	1450	1716	3625	0
9	1206	213	24	372	369	0
10	319	54	18	93	8	0
11	30	4	4	1	0	0
MP	5,82	5,32	5,23	5,38	6,03	4,58

Nosso método é uma atualização do algoritmo de síntese baseado em ciclos (CBS) de RIBEIRO *et al.* [25], incluindo na biblioteca portas GT parcialmente controladas, e melhora o número médio de portas, como segue. O número médio de portas necessárias para sintetizar um circuito é reduzido de 6,74 com o Algoritmo CBS unidirecional para 5,82 com o Algoritmo 3 considerando a coluna (a), e é reduzido de 6,64 com Algoritmo CBS aplicando Estratégia 1 para 5,32 com Algoritmo 3 aplicando a Estratégia 1 considerando a coluna (b).

Em segundo lugar, realizamos uma série de experimentos na síntese reversível de funções benchmark. Os resultados estão na Tabela 2.3. A primeira e a segunda colunas mostram o nome de cada função benchmark e seu tamanho (número de variáveis) considerados na literatura, respectivamente. A terceira coluna mostra os resultados obtidos pela abordagem proposta, utilizando o Algoritmo 3 e as Estratégias 1 e 2. A quarta coluna mostra os melhores resultados para a contagem de portas do algoritmo de síntese por ZAKABLUKOV [39] que usa propriedades da Teoria de Grupos de Permutações para reduzir a complexidade da porta e combina algoritmos baseados em ciclos e espectros de Reed-Muller. As duas últimas colunas mostram os melhores resultados para a contagem de portas do método MMD sem/com correspondência de modelos por MASLOV *et al.* [19], e o método MMD juntamente com espectros de Reed-Muller por MASLOV *et al.* [18], respectivamente. Todas as

especificações para as funções benchmark e seus nomes foram retirados do *RevLib site*[1] [36] e do *Reversible Logic Synthesis Benchmarks Page*[2].

Tabela 2.3: Síntese de funções benchmark. Nossos resultados estão listados na coluna **Algoritmo 3**. Os principais resultados atuais estão nas colunas **Z-16** [39], **MMD-05** [19] e **MMD-07** [18].

benchmark	tamanho	**Algoritmo 3**	**Z-16**	**MMD-05**	**MMD-07**
3_17	3	6	4	6/6	6
4_49	4	14	-	16/16	12
4b15g_2	4	17	12	-	-
4b15g_4	4	19	12	-	-
4b15g_5	4	18	14	-	-
aj−e11	4	9	-	13*/−	-
ham3	3	6	-	6/5	5
hwb4	4	18	-	18/17	11
hwb5	5	43	-	57/55	24
hwb6	6	103	-	134/126	42
hwb7	7	282	603	302/289	236
hwb8	8	697	1594	688/−	614
hwb9	9	2633	3999	1625/−	1541
mod5adder	6	17	-	37/−	15
mod5mils	5	3	-	5/−	-
nth_prime4_inc	4	13	-	-	12**
nth_prime5_inc	5	38	-	-	25**
nth_prime6_inc	6	79	-	-	55**
nth_prime7_inc	7	231	427	-	-
nth_prime8_inc	8	627	977	-	-

* Resultado baseado no método MMD executado por [10] para gerar um resultado heurístico.
** Resultado disponível em https://reversiblebenchmarks.github.io/

Analisando os resultados obtidos na Tabela 2.3, vemos que nossa abordagem obteve em mais da metade dos casos, melhores resultados em relação aos obtidos por ZAKABLUKOV [39] e MASLOV *et al.* [19] para síntese de funções benchmark. Conseguimos obter 9 (de 17) circuitos reversíveis, que têm contagem de portas igual ou menor em relação às duas referências citadas. Comparando nossos resultados com os obtidos por ZAKABLUKOV [39], vemos que obtivemos melhores resultados em 5 das 9 comparações, sendo que nosso algoritmo de síntese teve melhor desempenho à medida que aumentamos o número de bits. Por outro lado, comparando com os resultados obtidos por MASLOV *et al.* [19], vemos que obtivemos melhores resultados em 7 das 12 (1 igual) comparações, sendo que, em geral nosso desempenho foi melhor para funções de até 7 bits. Porém, os resultados obtidos por MASLOV *et al.* [18] baseados no método MMD [19], usando templates e espectros de Reed-Muller

[1] disponível em http://www.revlib.org
[2] disponível em https://reversiblebenchmarks.github.io/

são os melhores resultados até agora.

Em terceiro lugar, disponibilizamos em https://github.com/Marquezino/dkmfr uma implementação do nosso principal resultado para a comunidade. Para ter uma noção de até que ponto as permutações de 8 bits podem ser tratadas, consulte na Figura 2.4 um gráfico onde representamos os tempos de execuções reais correspondentes obtidos. O gráfico linear obtido na escala logarítmica de base 10 concorda com o comportamento exponencial esperado à medida que aumentamos o número de bits.

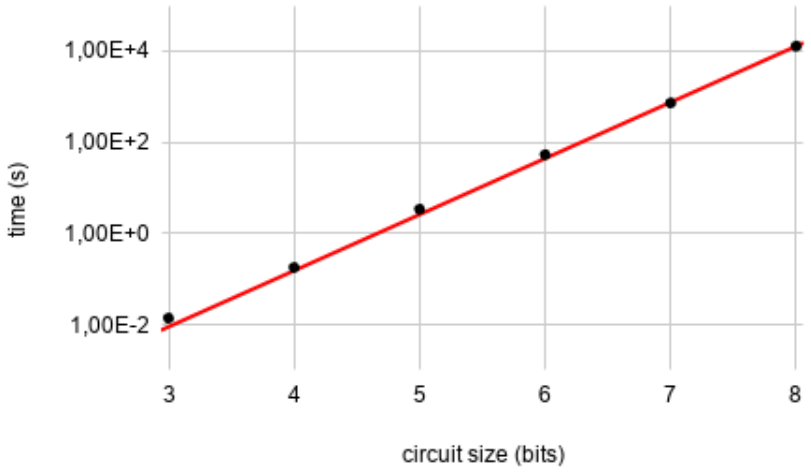

Figura 2.4: Tempos de execução reais obtidos (em segundos) do Algoritmo 3. O eixo do tempo é representado na escala logarítmica de base-10. Representamos em vermelho a linha de tendência com inclinação 6/5.

Capítulo 3

Otimização Pós-Síntese

Técnicas de otimização pós-síntese para circuitos reversíveis usando portas Toffoli com controles positivos (também conhecidas como portas Toffoli de múltiplos controles ou MCT) têm sido utilizadas em vários estudos [19, 20, 30]. Portas Toffoli generalizadas (portas GT), que também são conhecidas como portas MCT com controles mistos, foram introduzidas para a síntese de circuitos reversíveis por MASLOV e DUECK [16], e em DATTA *et al.* [5] foram usadas pela primeira vez para otimização pós-síntese de circuitos reversíveis. Nos últimos anos, tem havido um interesse crescente na otimização pós-síntese de circuitos reversíveis usando portas Toffoli com controles negativos via regras de substituição e correspondência de modelos [3–6, 23, 24, 31, 37, 38].

Neste capítulo, faremos revisão das regras usadas na otimização pós-síntese de circuitos reversíveis encontradas em [3, 4, 6, 24, 38]. Mas, também propomos uma nova organização para as regras com portas distância-2, tendo como objetivo principal um novo algoritmo para otimização pós-síntese de circuitos reversíveis usando portas GT. O número de portas (GC) é a métrica usada para avaliar nosso processo de otimização pós-síntese.

3.1 Regras de Otimização

Em geral, os circuitos retornados pelos algoritmos de síntese podem ser otimizados utilizando algumas propriedades das portas, formando regras de simplificação. Nesta seção, são enunciadas duas regras de eliminação, bem como regras de fusão, comutação, movimentação e substituição.

Regra 3.1 (Eliminação de Portas Idênticas)**.** *Duas portas* GT *consecutivas g e h, com os mesmos controles e alvos se anulam [3, 6, 20, 24].*

A Regra 3.1 também é conhecida como propriedade de autorreversibilidade [19].

Regra 3.2 (Eliminação de Portas NOT)**.** *Seja $\mathcal{C}$ um circuito de portas* GT*. Se em determinada linha i do circuito $\mathcal{C}$ existir um par de portas NOT e entre elas não houver portas com alvo nesta linha, estas podem ser eliminadas do circuito invertendo as conexões de controle na linha i de todas as portas intermediárias [4, 5].*

A Figura 3.1 ilustra a aplicação da Regra 3.2.

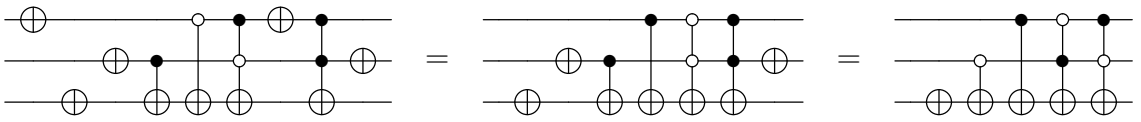

Figura 3.1: Regra de eliminação de portas NOT. Neste caso, o pares de portas NOT, na linha superior e na linha do meio, são eliminados.

Definição 3.1 (Portas Adjacentes)**.** Duas portas GT g e h são ditas *adjacentes* se tiverem os alvos na mesma linha e diferirem em somente uma linha em relação às conexões de controle (positivo, negativo ou ausente) [4].

Regra 3.3 (Fusão para Portas Adjacentes)**.** *Duas portas* GT *adjacentes consecutivas g e h podem ser fundidas em uma porta no circuito reversível. A porta resultante terá a conexão de controle (positivo, negativo ou ausente) que não aparece na linha em que as duas portas adjacentes g e h diferem [3–6, 24].*

A Figura 3.2 ilustra as possibilidades de aplicações para a Regra 3.3.

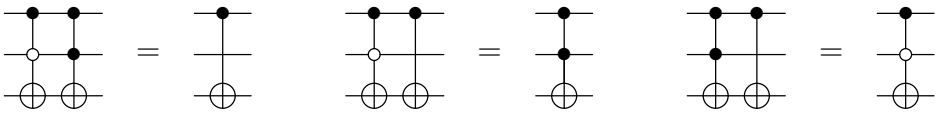

Figura 3.2: Regra de fusão para portas adjacentes.

Regra 3.4 (Comutação)**.** *Duas portas* GT *consecutivas g e h podem ser comutadas se uma das seguintes condições ocorrer: (i) a linha alvo de uma porta não é uma linha de controle da outra porta; (ii) as duas portas têm uma conexão de controle reverso em uma linha, e em todas as outras linhas nas quais ambas as portas têm conexões de controle, essas conexões são idênticas [3, 6, 24, 38].*

Existem duas possibilidades para o caso (i) da Regra 3.4. Na primeira, os alvos estão em linhas distintas e na segunda, os alvos estão na mesma linha. Os casos (i) e (ii) são ilustrados nas Figuras 3.3 e 3.4, respectivamente.

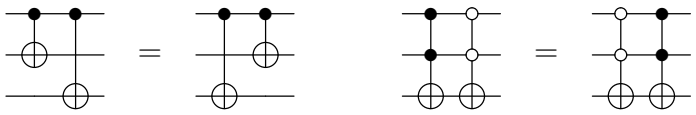

Figura 3.3: Regra de comutação: Caso (i), alvos em linhas distintas ou alvos na mesma linha.

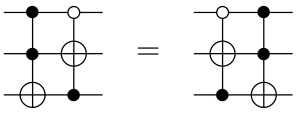

Figura 3.4: Regra de comutação: Caso (ii).

Regra 3.5 (Movimentação)**.** *Dadas duas portas* GT *g e h que satisfaçam as duas condições a seguir: (i) o alvo da porta g está em uma linha onde a porta h tem uma conexão de controle, e (ii) todas as linhas na porta g com uma conexão de controle também tem uma conexão de controle na porta h, e em todas as linhas nas quais ambas têm conexões de controle, estas são as mesmas. Neste caso g e h podem ser comutadas invertendo a conexão de controle da porta h posicionada na mesma linha que o alvo da porta g [3].*

A Figura 3.5 ilustra a Regra 3.5.

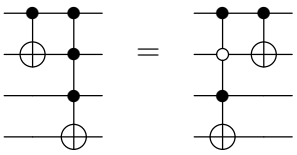

Figura 3.5: Regra de movimentação: comuta e inverte controle.

Definição 3.2 (Portas Distância-2)**.** Duas portas GT g e h são ditas *distância*-2 se tiverem os alvos na mesma linha e diferirem em duas linhas em relação às conexões de controle (positivo, negativo ou ausente) [4].

Regra 3.6 (Substituição para Portas Distância-2)**.** *Um par de portas distância-2 g e h pode ser substituído por outro par de portas distância-2 de acordo com a Tabela 3.1, que mostra as possibilidades de substituição, onde os casos anteriormente conhecidos [4] são referenciados.*

A Tabela 3.1 descreve os seis conjuntos diferentes possíveis com três pares equivalentes de portas. Em cada conjunto de portas podemos escolher o par de portas mais adequado para o circuito reversível, ou seja, um par de portas que leva a uma nova

aplicação de regras, possivelmente proporcionando uma diminuição na contagem de portas.

Tabela 3.1: Cenários para portas distância-2

Caso	Linha	Par 1		Par 2		Par 3	
		g_1	g_2	g_1	g_2	g_1	g_2
a[4]	i	1	0	1	-	0	-
	j	0	1	-	1	-	0
b[4]	i	1	0	1	-	0	-
	j	0	-	1	-	1	0
c[4]	i	1	-	0	-	1	0
	j	0	1	0	-	-	1
d[4]	i	1	-	0	-	1	0
	j	1	0	1	-	-	0
e	i	1	0	1	-	0	-
	j	1	-	0	-	0	1
f	i	1	-	1	0	0	-
	j	-	0	1	0	-	1

'1': controle positivo, '0': controle negativo,
'-': ausente

Lema 3.1. *Existem dezoito tipos de pares de portas distância-2, que podem ser divididos em seis conjuntos de três elementos cada. Os três pares de portas descritos em cada linha da Tabela 3.1 são equivalentes.*

Demonstração. Dadas quaisquer duas portas distância-2 g e h, a Regra 3.4 nos permite escrever que os pares ordenados gh e hg são equivalentes. Assim, após inspeção, é fácil ver que existem dezoito tipos de pares de portas distância-2. Portanto, usando as Regras 3.1, 3.3 e 3.4 mostra-se que tais pares de portas de cada linha da Tabela 3.1 podem ser divididos em seis conjuntos contendo três elementos equivalentes cada, ou seja, os três pares de portas têm o mesmo efeito no circuito reversível. □

As Figuras 3.6 a 3.11 ilustram os casos apresentados na Tabela 3.1, incluindo alguns circuitos intermediários. Note que usando as Regras 3.1, 3.3 e 3.4, verifica-se que os pares de portas dentro de cada conjunto (casos a a f) são equivalentes.

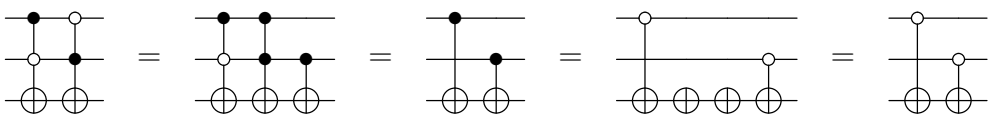

Figura 3.6: Portas distância-2 equivalentes. Caso (a) da Tabela 3.1.

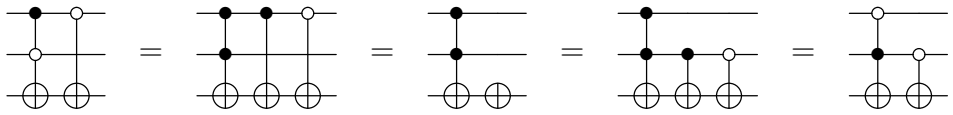

Figura 3.7: Portas distância-2 equivalentes. Caso (b) da Tabela 3.1.

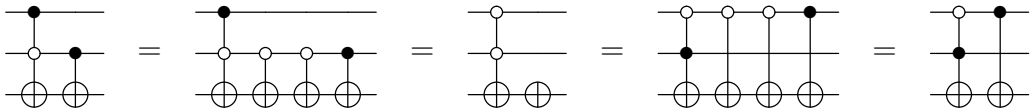

Figura 3.8: Portas distância-2 equivalentes. Caso (c) da Tabela 3.1.

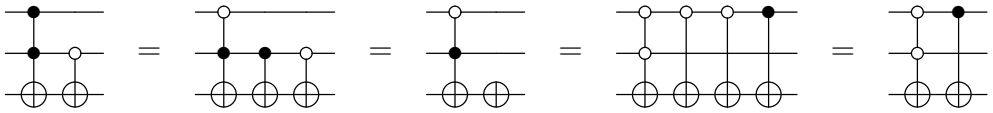

Figura 3.9: Portas distância-2 equivalentes. Caso (d) da Tabela 3.1.

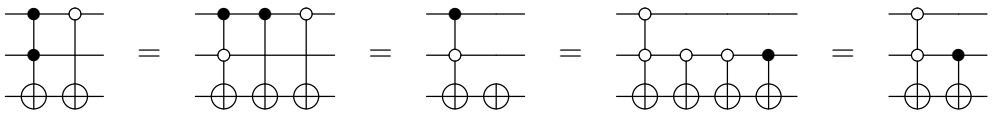

Figura 3.10: Portas distância-2 equivalentes. Caso (e) da Tabela 3.1.

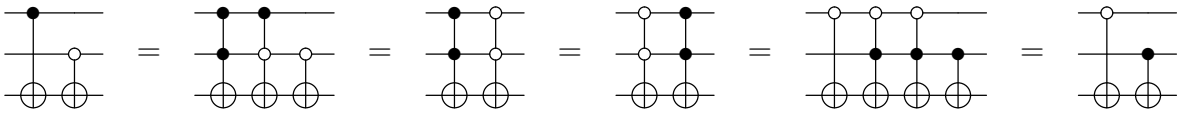

Figura 3.11: Portas distância-2 equivalentes. Caso (f) da Tabela 3.1.

3.2 Proposta de Algoritmo

A ordem de aplicação das regras apresentadas pode fazer diferença na quantidade de portas do circuito retornado pelo algoritmo de síntese de circuitos reversíveis. Assim, apresentamos no Algoritmo 4 a seguir uma proposta de algoritmo para Otimização Pós-Síntese. Para descrever este algoritmo que explora pela primeira vez de forma conjunta as seis regras descritas na seção anterior, precisamos do conceito de *segmento*. Em um dado circuito reversível de portas Toffoli generalizadas $\mathcal{C} = \{g_1, g_2, \ldots, g_p\}$, um segmento é definido como um conjunto de portas consecutivas $S = \{g_{k_1}, g_{k_1+1}, \ldots, g_{k_2}\}$, onde $S \subseteq \mathcal{C}, 1 \leq k_1 \leq k_2 \leq p$, tal que qualquer par de portas em S pode ser trocado sem afetar a funcionalidade do circuito.

O Algoritmo 4 descrito como proposta para Otimização Pós-Síntese pode ser dividido em duas fases: a primeira, com aplicações de regras dentro de cada segmento; a segunda, com aplicações de regras envolvendo segmentos consecutivos. O primeiro passo do algoritmo é identificar os segmentos do circuito dado como entrada. Assim, na primeira fase do algoritmo, em cada segmento, o algoritmo tenta aplicar primeiro a regra 3.2 (eliminação de portas NOT), pois tal regra pode levar à grande diminuição na contagem de portas do circuito. Em seguida, ele analisa os pares de portas contidos no segmento. Se forem encontradas duas portas iguais, utiliza, se necessário, a regra 3.4 (comutação) até que tais portas sejam consecutivas e então, aplica a regra 3.1 (eliminação de portas idênticas). A partir daí, identifica pares de portas adjacentes, e aplica o mesmo procedimento descrito acima, até que a regra 3.3 (fusão) possa ser aplicada.

Na segunda fase, o algoritmo analisa pares de portas em segmentos consecutivos e tenta aplicar a regra 3.5 (movimentação) e a regra 3.6 (substituição para portas distância-2), de modo a permitir a aplicação da regra 3.1 (eliminação de portas idênticas) ou a regra 3.3 (fusão). Para isso, a regra 3.4 (comutação) é utilizada novamente, para que as portas sejam movidas para as extremidades desses segmentos. Repare que a regra de comutação, assim como as regras de movimentação e de substituição para portas distância-2, são muito importantes, pois preparam o circuito para aplicações de outras regras que podem levar à diminuição do número de portas desse circuito.

Algoritmo 4: Otimização Pós-Síntese

Entrada: Circuito reversível $\mathcal{C}$.

Saída: Circuito reversível equivalente $\mathcal{C}'$ de $\mathcal{C}$.

```
1  Identificar os segmentos S_1, S_2, ...;
2  para cada segmento S faça
3      Tentar aplicar a regra 3.2 (eliminação de portas NOT);
4      para cada par de portas g, h de S faça
5          se g = h então
6              enquanto g e h não são consecutivas faça
7                  se possível aplicar regra 3.4 então
8                      aplicar regra 3.4 (comutação)
9                  senão
10                     break;
11                 fim
12             fim
13             se g e h são consecutivas então
14                 aplicar regra 3.1 (eliminação de portas idênticas);
15             fim
16         fim
17         se g e h são adjacentes então
18             enquanto g e h não são consecutivas faça
19                 se possível aplicar regra 3.4 então
20                     aplicar regra 3.4 (comutação)
21                 senão
22                     break;
23                 fim
24             fim
25             se g e h são consecutivas então
26                 aplicar regra 3.3 (fusão);
27             fim
28         fim
29     fim
30 fim
31 para cada par de portas g, h tal que g ∈ S_i e h ∈ S_{i+1} faça
32     se existe um par g', h' equivalente a g, h (de acordo com as regras 3.5
         e 3.6) tal que g' é idêntica ou adjacente a uma porta em S_{i+1} ou h' é
         idêntica ou adjacente a uma porta em S_i então
33         Aplicar regra 3.4 (comutação) para mover g para o final de S_i e h
             para o início de S_{i+1};
34         Aplicar regras 3.5 (movimentação) e 3.6 (substituição para portas
             distância-2);
35         Aplicar regra 3.1 (eliminação de portas idênticas) ou regra 3.3
             (fusão);
36     fim
37 fim
```

Capítulo 4

Conclusão

A presente tese de doutorado dá continuidade a duas teses anteriores sobre circuitos reversíveis, ambas desenvolvidas no Programa de Engenharia de Sistemas e Computação da COPPE. A tese de doutorado de Luis Antonio Brasil Kowada de 2006 intitulada "Construção de Algoritmos Reversíveis e Quânticos" é pioneira no tema no Brasil [13]. A tese de doutorado de André da Cunha Ribeiro de 2013 intitulada "Sobre grafos de Cayley, permutações e circuitos reversíveis" apresenta o algoritmo de síntese baseado em ciclos (CBS) [26] que a presente tese atualiza e melhora.

Neste trabalho, apresentamos um novo algoritmo de síntese de circuitos reversíveis usando portas Toffoli generalizadas (portas GT). Nosso Algoritmo 3 é uma atualização do algoritmo CBS [25], incluindo as portas GT parcialmente controladas na biblioteca. A nova abordagem aumenta progressivamente o número de controles nas portas Toffoli. Além disso, nosso Algoritmo 3 explora propriedades das representações de ciclo de permutações como parte do processo e usa estratégias bidirecionais. Os circuitos obtidos por nosso novo algoritmo de síntese em geral usam menos portas do que os circuitos obtidos pelo algoritmo CBS. Nosso Algoritmo 3 funciona para funções bijetivas arbitrárias de n bits, enquanto a síntese exata encontrada na literatura funciona apenas para os casos de $n = 3$ e $n = 4$ bits [33, 37]. Considerando que o custo quântico [1] de cada porta GT é proporcional ao número de bits de controle, o custo quântico dos circuitos gerados pelo Algoritmo 3 é, em média, menor que o custo quântico dos circuitos gerados pelo algoritmo CBS. No presente trabalho, consideramos como métrica de complexidade apenas o número de portas, um modelo de custo frequentemente utilizado na literatura e considerado adequado quando se permanece dentro do arcabouço reversível tradicional [27].

As principais contribuições incluem a melhor contagem média de portas até agora, para todas as funções bijetivas de 3 bits para a biblioteca composta pelas portas GT. Além disso, apresentamos resultados experimentais para síntese de funções de benchmark reversíveis, que inclui vinte circuitos reversíveis constituídos por portas da biblioteca GT, que são competitivos quando comparados com os

melhores resultados para contagem de portas de heurísticas de síntese de circuitos reversíveis da literatura.

Além disso, consideramos regras de otimização visando um novo algoritmo para otimização pós-síntese de circuitos reversíveis compostos por portas Toffoli generalizadas. O Algoritmo 4 proposto é o primeiro a explorar de forma simultânea as seis regras descritas.

Como pesquisas futuras, possíveis direções incluem o uso de modelos para substituir por outros mais curtos certas sequências de portas nos circuitos retornados pelo Algoritmo 3. Além disso, poderíamos usar algumas outras abordagens, como a teoria de grupos de permutações [32, 35, 39] e espectros de Reed-Muller [18, 22], a fim de encontrar melhores sequências de portas. Também poderíamos aplicar técnicas semelhantes para a síntese de circuitos quânticos [7], levando em consideração que passar para o nível de circuitos quânticos requer um conjunto de portas mais refinado do que o considerado aqui e otimizações que se alinham com as necessidades de correção de erros. Finalmente, podemos também aperfeiçoar o Algoritmo 4 proposto para otimização pós-síntese e concluir os experimentos, bem como, procurar novas regras de otimização.

Referências Bibliográficas

[1] BARENCO, A., BENNETT, C. H., CLEVE, R., et al., 1995, "Elementary gates for quantum computation", *Phys. Rev. A*, v. 52, n. 5 (Nov), pp. 3457–3467.

[2] BENNETT, C. H., 1973, "Logical reversibility of computation", *IBM Journal of Research and Development*, v. 17, n. 6 (Nov), pp. 525–532.

[3] CHENG, X., GUAN, Z., WANG, W., et al., 2012, "A simplification algorithm for reversible logic network of positive/negative control gates". In: *9th International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 2442–2446, May.

[4] DATTA, K., SENGUPTA, I., RAHAMAN, H., 2015, "A post-synthesis optimization technique for reversible circuits exploiting negative control lines", *IEEE Transactions on Computers*, v. 64, n. 4 (Apr), pp. 1208–1214.

[5] DATTA, K., RATHI, G., WILLE, R., et al., 2013, "Exploiting negative control lines in the optimization of reversible circuits". In: Dueck, G. W., Miller, D. M. (Eds.), *Reversible Computation*, pp. 209–220, Berlin, Heidelberg. Springer.

[6] DE ALMEIDA, A. A. A., DUECK, G. W., DA SILVA, A. C. R., 2018, "Reversible circuit optimization based on Tabu Search". In: *IEEE 48th International Symposium on Multiple-Valued Logic*, pp. 103–108.

[7] DE ALMEIDA, A. A. A., DUECK, G. W., DA SILVA, A. C. R., 2019, "Efficient realization of Toffoli and NCV circuits for IBM QX architectures". In: Thomsen, M. K., Soeken, M. (Eds.), *Reversible Computation*, pp. 131–145, Cham. Springer International Publishing.

[8] FEYNMAN, R., 1985, "Quantum mechanical computers", *Optics News*, v. 11, n. 2 (Fev), pp. 11–20.

[9] GOLUBITSKY, O., FALCONER, S. M., MASLOV, D., 2010, "Synthesis of the optimal 4-bit reversible circuits". In: *Design Automation Conference*, pp. 653–656, July.

[10] GROSSE, D., WILLE, R., DUECK, G. W., et al., 2009, "Exact multiple-control Toffoli network synthesis with SAT techniques", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 28, n. 5 (May), pp. 703–715.

[11] GUPTA, P., AGRAWAL, A., JHA, N., 2006, "An algorithm for synthesis of reversible logic circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 25, n. 11 (Nov), pp. 2317–2330.

[12] IWAMA, K., KAMBAYASHI, Y., YAMASHITA, S., 2002, "Transformation rules for designing CNOT-based quantum circuits". In: *Proceedings of the 39th Annual Design Automation Conference*, pp. 419–424, New York, NY, USA. ACM.

[13] KOWADA, L. A. B., 2006, *Construção de Algoritmos Reversíveis e Quânticos*. Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

[14] KOWADA, L. A. B., PORTUGAL, R., DE FIGUEIREDO, C. M. H., 2006, "Reversible Karatsuba's algorithm", *Journal of Universal Computer Science*, v. 12, n. 5 (June), pp. 499–511.

[15] LANDAUER, R., 1961, "Irreversibility and heat generation in the computing process", *IBM Journal of Research and Development*, v. 5, n. 3 (July), pp. 183–191.

[16] MASLOV, D., DUECK, G. W., 2006, "Level compaction in quantum circuits". In: *IEEE International Conference on Evolutionary Computation*, pp. 2405–2409.

[17] MASLOV, D., DUECK, G. W., 2004, "Reversible cascades with minimal garbage", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 23, n. 11 (Nov), pp. 1497–1509.

[18] MASLOV, D., DUECK, G. W., MILLER, D. M., 2007, "Techniques for the synthesis of reversible Toffoli networks", *ACM Trans. Des. Autom. Electron. Syst.*, v. 12, n. 4 (Sep), pp. 42–es.

[19] MASLOV, D., DUECK, G. W., MILLER, D. M., 2005, "Toffoli network synthesis with templates", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 24, n. 6 (June), pp. 807–817.

[20] MILLER, D. M., MASLOV, D., DUECK, G. W., 2003, "A transformation based algorithm for reversible logic synthesis". In: *Proceedings of the 40th Annual Design Automation Conference*, pp. 318–323, New York, NY, USA. ACM.

[21] NIELSEN, M. A., CHUANG, I. L., 2000, *Quantum Computation and Quantum Information*. New York, NY, USA, Cambridge University Press.

[22] PORWIK, P., 2002, "Efficient calculation of the Reed-Muller form by means of the Walsh transform", *International Journal of Applied Mathematics and Computer Science*, v. 12, n. 4 (May), pp. 571–579.

[23] RAHMAN, M. M., SOEKEN, M., DUECK, G. W., 2015, "Dynamic template matching with mixed-polarity Toffoli gates". In: *IEEE 45nd International Symposium on Multiple-Valued Logic*, pp. 72–77, May.

[24] RAHMAN, M. Z., RICE, J. E., 2014, "Templates for positive and negative control Toffoli networks". In: Yamashita, S., Minato, S.-I. (Eds.), *Reversible Computation*, pp. 125–136, Cham. Springer International Publishing.

[25] RIBEIRO, A. C., KOWADA, L. A. B., MARQUEZINO, F. L., et al., 2015, "A new reversible circuit synthesis algorithm based on cycle representations of permutations", *Electronic Notes in Discrete Mathematics*, v. 50 (Dec), pp. 187–192.

[26] RIBEIRO, A. C., 2013, *Sobre Grafos de Cayley, Permutações e Circuitos Reversíveis*. Tese de Doutorado, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, RJ, Brasil.

[27] SAEEDI, M., MARKOV, I. L., 2013, "Synthesis and optimization of reversible circuits - a survey", *ACM Comput. Surv.*, v. 45, n. 2 (Mar), pp. 21:1–21:34.

[28] SAEEDI, M., ZAMANI, M. S., SEDIGHI, M., et al., 2010, "Reversible circuit synthesis using a cycle-based approach", *J. Emerg. Technol. Comput. Syst.*, v. 6, n. 4 (Dec), pp. 13:1–13:26.

[29] SHENDE, V. V., PRASAD, A. K., MARKOV, I. L., et al., 2003, "Synthesis of reversible logic circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 22, n. 6 (June), pp. 710–722.

[30] SOEKEN, M., WILLE, R., DUECK, G. W., et al., 2010, "Window optimization of reversible and quantum circuits". In: *13th IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, pp. 341–345.

[31] SOEKEN, M., THOMSEN, M. K., 2013, "White dots do matter: Rewriting reversible logic circuits". In: Dueck, Gerhard W.and Miller, D. M. (Ed.), *Reversible Computation*, pp. 196–208, Berlin, Heidelberg. Springer.

[32] STORME, L., VOS, A. D., JACOBS, G., 1999, "Group theoretical aspects of reversible logic gates", *Journal of Universal Computer Science*, v. 5, n. 5 (Jan), pp. 307–321.

[33] SZYPROWSKI, M., KERNTOPF, P., 2012, "A study of optimal 4-bit reversible circuit synthesis from mixed-polarity Toffoli gates". In: *12th IEEE International Conference on Nanotechnology*, pp. 1–6, Aug.

[34] TOFFOLI, T., 1980, "Reversible computing". In: de Bakker, J., van Leeuwen, J. (Eds.), *Automata, Languages and Programming*, p. 632, New York. Springer. MIT Technical Memo No. MIT/LCS/TM-151, 1980 (unpublished).

[35] VOS, A. D., RAA, B., STORME, L., 2002, "Generating the group of reversible logic gates", *Journal of Physics A: Mathematical and General*, v. 35, n. 33 (Aug), pp. 7063–7078.

[36] WILLE, R., GROSSE, D., TEUBER, L., et al., 2008, "RevLib: An online resource for reversible functions and reversible circuits". In: *IEEE 38th International Symposium on Multiple-Valued Logic*, pp. 220–225, May.

[37] WILLE, R., SOEKEN, M., PRZIGODA, N., et al., 2012, "Exact synthesis of Toffoli gate circuits with negative control lines". In: *IEEE 42nd International Symposium on Multiple-Valued Logic*, pp. 69–74, May.

[38] WILLE, R., SOEKEN, M., OTTERSTEDT, C., et al., 2013, "Improving the mapping of reversible circuits to quantum circuits using multiple target lines". In: *18th Asia and South Pacific Design Automation Conference*, pp. 145–150.

[39] ZAKABLUKOV, D. V., 2016, "Application of permutation group theory in reversible logic synthesis". In: Devitt, S., Lanese, I. (Eds.), *Reversible Computation*, pp. 223–238, Cham. Springer International Publishing.

[40] ZHU, W., LI, Z., ZHANG, G., et al., 2018, "A reversible logical circuit synthesis algorithm based on decomposition of cycle representations of permutations", *International Journal of Theoretical Physics*, v. 57, n. 8 (Aug), pp. 2466–2474.

Apêndice A

Anexo: Manuscrito "A reversible circuit synthesis algorithm with progressive increase of controls in generalized Toffoli gates"

Journal of Universal Computer Science, vol. 27, no. 6 (2021), 544-563
submitted: 5/5/2020, accepted: 14/5/2021, appeared: 28/6/2021 CC BY-ND 4.0

A reversible circuit synthesis algorithm with progressive increase of controls in generalized Toffoli gates

Edinelço Dalcumune
(Universidade Federal dos Vales do Jequitinhonha e Mucuri, Brazil
Universidade Federal do Rio de Janeiro, Brazil
https://orcid.org/0000-0002-3999-6664, edcomune@ufvjm.edu.br)

Luis Antonio Brasil Kowada
(Universidade Federal Fluminense, Brazil
https://orcid.org/0000-0002-7975-0060, luis@ic.uff.br)

André da Cunha Ribeiro
(Instituto Federal Goiano, Brazil
https://orcid.org/0000-0003-0641-854X, andre.cunha@ifgoiano.edu.br)

Celina Miraglia Herrera de Figueiredo
(Universidade Federal do Rio de Janeiro, Brazil
https://orcid.org/0000-0002-6393-0876, celina@cos.ufrj.br)

Franklin de Lima Marquezino
(Universidade Federal do Rio de Janeiro, Brazil
https://orcid.org/0000-0001-9712-1930, franklin@cos.ufrj.br)

Abstract: We present a new algorithm for synthesis of reversible circuits for arbitrary n-bit bijective functions. This algorithm uses generalized Toffoli gates, which include positive and negative controls. Our algorithm is divided into two parts. First, we use partially controlled generalized Toffoli gates, progressively increasing the number of controls. Second, exploring the properties of the representation of permutations in disjoint cycles, we apply generalized Toffoli gates with controls on all lines except for the target line. Therefore, new in the method is the fact that the obtained circuits use first low cost gates and consider increasing costs towards the end of the synthesis. In addition, we employ two bidirectional synthesis strategies to improve the gate count, which is the metric used to compare the results obtained by our algorithm with the results presented in the literature. Accordingly, our experimental results consider all 3-bit bijective functions and twenty widely used benchmark functions. The results obtained by our synthesis algorithm are competitive when compared with the best results known in the literature, considering as a complexity metric just the number of gates, as done by alternative best heuristics found in the literature. For example, for all 3-bit bijective functions using generalized Toffoli gates library, we obtained the best so far average count of 5.23.

Keywords: design of algorithms, reversible computing, circuit synthesis, cycle representations of permutations
Categories: B.2, B.6, F.1, F.2
DOI: 10.3897/jucs.69617

1 Introduction

In the last decades, the synthesis of reversible circuits has received considerable attention due to the possibility of applications in several areas of science, such as signal processing, cryptography, quantum computing, low-power design, computer graphics, optical computing, DNA computing, bioinformatics, nano and photonic circuits [Saeedi and Markov, 2013]. One of the main motivations for reversible computing is that quantum computing has as one of its foundations the reversibility of all gates, that is, quantum computing circuit models are reversible. This fact stems from the evolution postulate of quantum mechanics, where a unitary operator describes the time-evolution of the state of a closed quantum system [Nielsen and Chuang, 2000]. Another main motivation is the important physical consequences for reversible computing. Landauer [Landauer, 1961] proved that using irreversible logic gates necessarily leads to power dissipation regardless of the underlying technology. The reason is that each erased bit leads to at least $kT \ln 2$ energy dissipation, where k is Boltzmann's constant and T is the absolute temperature of the circuit. On the other hand, Bennett [Bennett, 1973] showed how to use reversible logic gates to reduce or even eliminate power dissipation in a circuit. Indeed, in a reversible circuit—classical or quantum—we can retrieve all information from the circuit input using what was obtained at the circuit output. That is, in this process no input information is erased. Moreover, Bennett also showed that zero power dissipation in circuits is only possible if their calculation is reversible.

An important problem in reversible computing that has been intensively studied for the last decades is the synthesis of reversible circuits. The problem can be stated as: Given a bijective function f, find the best possible reversible circuit that implements f [Shende et al., 2003, Miller et al., 2003, Kowada et al., 2006, Maslov et al., 2007, Golubitsky et al., 2010, Saeedi and Markov, 2013].

In this work, we present a new algorithm for synthesis of reversible circuits using multiple-control Toffoli gates [Toffoli, 1980, Iwama et al., 2002] with any number of positive or negative controls, also known as generalized Toffoli gates [Maslov and Dueck, 2004, Zakablukov, 2016]. Our method is an upgrade of the cycle-based synthesis (CBS) algorithm [Ribeiro et al., 2015]. Besides using only totally controlled gates—gates with controls on all lines except for the target line—we also include partially controlled Toffoli gates [Iwama et al., 2002]. An important contribution from our method is the fact that the obtained circuits use low cost gates first, and consider increasing costs towards the end of the synthesis. For 3 bits, the circuits achieved by our new synthesis algorithm use less gates than the circuits achieved by the CBS algorithm, and by recent algorithms in Zhu *et al.* [Zhu et al., 2018] and Cheng *et al.* [Cheng et al., 2012] that rely on partially controlled Toffoli gates with negative controls. Previous works [Wille et al., 2012, Datta et al., 2013, Rahman and Rice, 2014] have shown that gate libraries including negative controlled Toffoli gates may be more efficient for circuit synthesis. Our proposed algorithm and previous algorithms found in [Maslov and Dueck, 2004, Maslov et al., 2005] and [Ribeiro et al., 2015] rely on the Hamming distance between the permutation that needs to be synthesized and the identity, and try to minimize by making the moves (gate assignments) that do not increase the Hamming distance. We start with gates with few controls, and then increase the number of controls until we finally have totally controlled gates, in which case we can apply CBS algorithm [Ribeiro et al., 2015], which explores properties of the cycle representation of permutations. An alternative cycle representation has been used in [Saeedi et al., 2010], where the authors consider the permutation as a product of small not necessarily disjoint cycles and explores the properties of certain kinds of products, such as products of transpositions, for instance.

Synthesis can be performed optimally or heuristically, we refer to the survey on synthesis and optimization of reversible circuits [Saeedi and Markov, 2013]. We compare our heuristic results for the synthesis of all 3-bit bijective functions with the main current results using the Generalized Toffoli library obtained by Zhu *et al.* [Zhu et al., 2018] and Cheng *et al.* [Cheng et al., 2012]. Zhu *et al.* cycle-decomposition-based synthesis algorithm is specific to 3-bit bijective functions, and its main idea consists in representing the permutation in disjoint cycles and then using the so-called Head-Pointer-Adjust, which, over all cycles, takes the greatest Hamming distance between two consecutive elements. On the other hand, Cheng *et al.* synthesis algorithm also specific to 3-bit bijective functions uses templates and bidirectional method. It is important to mention that, differently from the algorithms of Zhu *et al.* and Cheng *et al.*, our algorithm works for arbitrary n-bit bijective functions. Besides that, our algorithm achieves circuits with smaller number of gates in average, a cost model adequate when staying within the traditional reversible framework.

There are optimal algorithms restricted to 3 bits [Wille et al., 2012] and to 4 bits [Szyprowski and Kerntopf, 2012]. These methods mainly formulate the synthesis problem as a sequence of instances of standard decision problems, such as Boolean satisfiability. We remark that only a small number of qubits can be handled by these methods.

Furthermore, we perform a series of experiments on benchmark bijective functions. We compare the results obtained by our synthesis algorithm with the best results in the literature, due to Zakablukov [Zakablukov, 2016] and Maslov *et al.* [Maslov et al., 2005, Maslov et al., 2007], for the gate count problem.

The present work is organized as follows. In Sec. 2, we briefly introduce the key concepts of reversible computing. In Sec. 3, we present our synthesis algorithm. In Sec. 4, we present the experimental results that we obtained. Finally, in Sec. 5, we present our final remarks and propose further related questions.

2 Preliminary notions

In the combinational model, a circuit using elementary logical gates AND, OR or NOT can be used to implement a function $f : \{0,1\}^n \to \{0,1\}^m$, where n and m denote the numbers of bits necessary to represent the input and output size, respectively. If f is a bijective function, then it can be represented by a permutation with 2^n values. Otherwise, the function f can be transformed into a new bijective function, by adding ancilla bits. For instance, the function $f = \{(0,7),(1,4),(2,1),(3,0),(4,3),(5,2),(6,6),(7,5)\}$ can be represented by the permutation $[7,4,1,0,3,2,6,5]$.

There are gates that implement bijective functions $f(x_0,\ldots,x_{n-1})$ called *reversible gates*, that is, such gates compute the function f, given the input $x_0,\ldots,x_{n-1}$. The Multiple-Control Toffoli (MCT) gates are some important reversible gates [Toffoli, 1980, Iwama et al., 2002]. These gates, also known as C^{k}NOT$(x_{e_0},\ldots,x_{e_k})$, where $0 \leq e_0,\ldots,e_k < n$, keep the $k<n$ lines related to $x_{e_0},\ldots,x_{e_{k-1}}$ (the control lines) unchanged and flip the bit in the line related to x_{e_k} (the target line) if and only if all control lines carry the 1 value. In particular, for $k = 0, 1$ and 2 the gates are named NOT, CNOT and Toffoli, respectively. These reversible gates together with C^{3}NOT are depicted in Fig. 1.

An n-bit *reversible circuit* is defined as a sequence of reversible gates each acting on at most n lines. This type of circuits can be used to implement bijective functions. More specifically, since each elementary logical gate AND, OR or NOT can be simulated

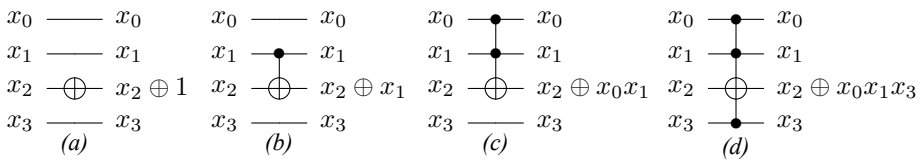

Figure 1: Representation of (a) $\text{NOT}(x_2) = x_2 \oplus 1$*,*
(b) $\text{CNOT}(x_1, x_2) = (x_1, x_2 \oplus x_1)$*, (c) Toffoli*$(x_0, x_1, x_2) = (x_0, x_1, x_2 \oplus x_0x_1)$ *and*
(d) $\text{C}^3\text{NOT}(x_0, x_1, x_3, x_2) = (x_0, x_1, x_3, x_2 \oplus x_0x_1x_3)$.

by a constant number of NOT, CNOT, and Toffoli gates, reversible circuits form a computational model equivalent to combinational circuits.

The *problem of reversible circuit synthesis* is defined as, given a bijective function $f : \{0,1\}^n \to \{0,1\}^n$, find an n-bit reversible circuit that computes f. The input of the problem is a permutation π of S_{2^n} and the output is the reversible circuit that transforms the identity permutation into π. We aim to find the optimal reversible circuit.

The C^kNOT gates, with positive and negative controls, called Generalized Toffoli (GT) gates, allow positive and negative controls to flip the bit in the target line. These gates flip the bit in the target line if and only if each positive (or negative) control line carries the 1 (or 0) value. We indicate a negative control line with $'$ after the label of the respective parameter. The values on the remaining lines (without control) do not affect any output. If $k = n-1$, we have the generalized Toffoli gates that are totally controlled, called Totally GT gates. On the other hand, if $k < n-1$, we have the generalized Toffoli gates that are partially controlled, called Partially GT gates.

Definition 1 *The Hamming distance $d_H(i,j)$ between two values i and j in binary representation is the number of bit positions that differ. Two values i and j are adjacent if $d_H(i,j) = 1$.*

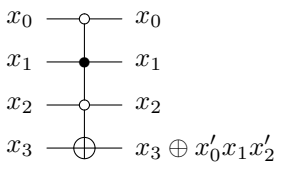

Figure 2: Totally GT *gate representing*
$\text{C}^3\text{NOT}(x_0', x_1, x_2', x_3) = (x_0', x_1, x_2', x_3 \oplus x_0'x_1x_2')$*. This gate changes the sequence $x_3x_2x_1x_0 = 0010$ into 1010 (in binary representation) and vice versa. The bottom line denotes the most significant bit. An equivalent representation is* $X(2, 10)$.

An equivalent representation for the Totally GT gate is $X(i,j)$ to indicate that it swaps values i and j, where i and j should be adjacent. We may also understand the $X(i,j)$ notation as follows. Let us consider $i < j$ and t the bit position that they differ. Let $b_{n-1} \cdots b_{t+1}\mathbf{0}b_{t-1} \cdots b_0$ and $b_{n-1} \cdots b_{t+1}\mathbf{1}b_{t-1} \cdots b_0$ be the binary representations of i and j, respectively, where b_{n-1} is the most significant bit. In an n-bit circuit, the

$X(i,j)$ gate is equivalent to C^{n-1}NOT$(b_0,\ldots,b_{t-1},b_{t+1},\ldots,b_{n-1},b_t)$. Note that the target line must be indicated as the last parameter in a C^{n-1}NOT gate, and the order of the other parameters, which indicate the control lines, is not relevant. Throughout the text we shall use the control parameters in crescent order of the indices. See Fig. 2 for an example of a Totally GT gate, and see Fig. 3 for an example of a reversible circuit that transforms the identity permutation into π. To clarify our use of equivalent representations, we include a corresponding Table 1 with the explicit sequence of used gates. The choice of the example is justified in Sec. 3.3, where we present in Figs. 5 and 6 some optimized equivalent circuits returned by our proposed algorithm.

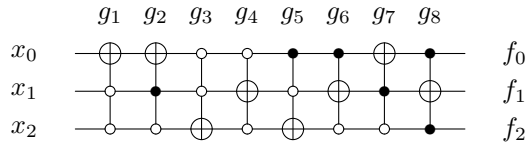

Figure 3: Example of a reversible circuit that transforms the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ *using only Totally* GT *gates. The sequence of gates (reading from left to right) is* $g_1 = X(0,1)$*,* $g_2 = X(2,3)$*,* $g_3 = X(0,4)$*,* $g_4 = X(0,2)$*,* $g_5 = X(1,5)$*,* $g_6 = X(1,3)$*,* $g_7 = X(2,3)$ *and* $g_8 = X(5,7)$*. We consider the representation of the numbers with* x_0 *being the least significant bit and* x_2 *being the most significant bit. The output bits are denoted by* $f_2 f_1 f_0$*.*

	$x_2x_1x_0$								$f_2f_1f_0$
	000	00**1**	001	001	001	**1**01	101	101	1**1**1 = 7
	001	00**0**	000	**1**00	100	100	100	100	100 = 4
	010	010	01**1**	011	011	011	0**0**1	001	001 = 1
	011	011	01**0**	010	0**0**0	000	000	000	000 = 0
	100	100	100	**0**00	0**1**0	010	010	01**1**	011 = 3
	101	101	101	101	101	**0**01	0**1**1	01**0**	010 = 2
	110	110	110	110	110	110	110	110	110 = 6
	111	111	111	111	111	111	111	111	1**0**1 = 5
Gates:	g_1 ↗	g_2 ↗	g_3 ↗	g_4 ↗	g_5 ↗	g_6 ↗	g_7 ↗	g_8 ↗	

Table 1: Sequence of gates $g_1 = X(0,1)$*,* $g_2 = X(2,3)$*,* $g_3 = X(0,4)$*,* $g_4 = X(0,2)$*,* $g_5 = X(1,5)$*,* $g_6 = X(1,3)$*,* $g_7 = X(2,3)$ *and* $g_8 = X(5,7)$ *that transforms the identity permutation into* $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ *using only Totally* GT *gates (See Fig. 3). In bold, the bit changes introduced by the corresponding gates. The result of application of the gate on the bottom of each column is shown in the next column while reading from left to right.*

Definition 2 *Let π and σ be permutations with n elements each. The Hamming distance $d_H(\pi, \sigma)$ is the sum of $d_H(\pi_i, \sigma_i)$ where π_i and σ_i are the elements in position i of π and σ respectively, for $0 \leq i < n$. We denote by $d_H(\pi)$ the Hamming distance between π and the identity ι.*

For example, the Hamming distance $d_H(\pi)$ between permutation $\pi = [7\,4\,1\,0\,3\,2\,6\,5]$ and the identity is the sum $3+2+2+2+3+3+0+1 = 16$. A circuit that transforms the identity into π is given in Fig. 3.

Our algorithm uses the cycle representation of a permutation. Let $\pi = [\pi_0\ \pi_1\ \cdots\ \pi_{2^n-1}]$ be a permutation and let $\pi = C_1C_2\cdots C_m$ be its corresponding cycle representation. Each cycle C can be represented by $C = (c^1c^2\cdots c^{|C|})$, where $c^{i+1} = \pi_{c^i}$ with $c^0 = c^{|C|}$. Therefore, in case $c^i = \pi_j$, we have $d_H(c^{i-1}, c^i) = d_H(j, \pi_j)$. We denote by $d_H(\pi_j)$ the Hamming distance between π_j and j, and we shall use both names $d_H(c^{i-1}, c^i)$ and $d_H(\pi_j)$ interchangeably. A permutation π is unicyclic if it can be represented by a single cycle.

Given a cycle C of length $|C|$ in a permutation π, we define

$$S(C) = \sum_{i=1}^{|C|} d_H(c^{i-1}, c^i), \tag{1}$$

with $c^0 = c^{|C|}$. It follows immediately from the definition that

$$d_H(\pi) \equiv d_H(\pi, \iota) = \sum_{\ell=1}^{m} S(C_\ell), \tag{2}$$

where m is the number of cycles of π.

For example, for the permutation π given by Fig. 3 the cycle representation is $(0\ 7\ 5\ 2\ 1\ 4\ 3)(6)$, and $d_H(\pi) = 16 + 0 = 16$.

3 Algorithm

Our algorithm is divided in two parts. The first part which is described in Subsection 3.1 consists in searching Partially GT gates with the goal of finding the gate that gives us the greatest decrease in Hamming distance, increasing progressively the number of controls. The permutation associated with each gate is obtained in a preprocessing step and stored in an array. The second part which is described in Subsection 3.2 consists only in trying to apply Totally GT gates. Finally in Subsection 3.3, our Synthesis algorithm with progressive increase of controls in GT gates is described using the algorithms obtained in the First and Second parts.

3.1 First Part: Partially GT gates

The goal is to decrease the Hamming distance between the current permutation and the identity, using as few controls as possible for the generalized Toffoli gates at the beginning of the algorithm. Hence, we first try the NOT gates, while it is possible to decrease the Hamming distance, then we try the CNOT gates, then we try the gates with two controls, and so on, until we get to the Totally GT gates.

In a preprocessing part, we build an array with all possible logical gates for the Generalized Toffoli library for n-bit circuits. The number of possible GT gates is $n3^{n-1}$. Indeed, for each line that the target bit occupies among the n possible lines, the $n-1$ remaining lines can be occupied in three different ways: positive control, negative control or uncontrolled. The construction of the array begins with the Totally GT gates, i.e., gates with $n-1$ controls. From these gates, we build Toffoli gates with $n-2$ controls, which in turn are used to build the Toffoli gates with $n-3$ controls, and so on.

The permutations associated with the Totally GT gates are of the form $(i\ j)$ in the cyclic notation, which corresponds to one transposition. In turn, the permutations associated with the GT gates with $n-2$ controls are of the form $(i\ j)(k\ l)$, while the permutations associated with the GT gates with $n-3$ controls are formed by four transpositions, and so on.

We present in Fig. 4 an example for 4 bits, where the composition of two Totally GT gates corresponding to $(4\ 12)$ and $(5\ 13)$ is equivalent to one GT gate $(4\ 12)(5\ 13)$ with 2 controls.

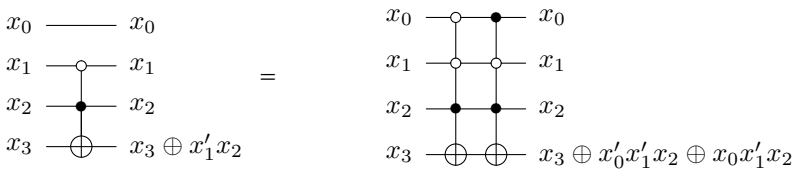

Figure 4: The C^{2}NOT(x_1', x_2, x_3) *gate is equivalent to the composition of gates* C^{3}NOT(x_0', x_1', x_2, x_3) *and* C^{3}NOT(x_0, x_1', x_2, x_3).

We say a transformation T over a permutation π is a *d-move* if $d_H(\pi) - d_H(T(\pi)) = d$, or equivalently, $d_H(\pi, T(\pi)) = d$.

Lemma 3 *If T is a d-move corresponding to a* GT *gate, with n bits and c controls, where $0 \leq c < n$, then $|d| \leq 2^{n-c}$ and $|d|$ is even.*

Proof. Let π be a permutation. For each d-move T corresponding to a GT gate, with n bits and c controls, where $0 \leq c < n$, we have $d_H(\pi \oplus \sigma, 0) = 2^{n-c}$, where $\sigma = T(\pi)$ and $\oplus$ is the bitwise XOR operation. Therefore, exactly 2^{n-c} bits (in distinct elements) are swapped by T. Suppose that among the 2^{n-c} bits swapped for T there were q bits that before the swap differed from the corresponding bits of the ι identity permutation. When effecting the swap, such q bits will coincide with the corresponding bits of the ι. On the other hand, the $p = 2^{n-c} - q$ bits that initially coincided with the corresponding bits in ι will now be different. So, d-move will be such that $d = q - (2^{n-c} - q) = 2(q - 2^{n-c-1})$. Therefore, d is even and $|d| = |q - p| = |2(q - 2^{n-c-1})| \leq 2^{n-c}$. □

Thus, in particular, if T is a transformation corresponding to a Totally GT gate, then T is either a 2-move, or a 0-move, or a -2-move.

For example, let $\pi = (0\,7\,6\,1\,2\,11\,13\,12\,3\,15\,14\,8\,10\,5\,9\,4)$ be a unicyclic permutation. The Partially GT gate C^{2}NOT(x_1, x_3', x_0), which corresponds to $(2\ 3)(6\ 7)$ applied to π, returns $\sigma = (0\,6\,1\,3\,15\,14\,8\,10\,5\,9\,4)(2\,11\,13\,12)(7)$ and $d_H(\pi) - d_H(\sigma) = 34 - 30 = 4$, so gate C^{2}NOT$(x_1, x_3', x_0)$ applied to π is a 4-move. Gate C^{2}NOT(x_1, x_3', x_0) applied to σ is a -4-move. The Partially GT gate C^{2}NOT(x_1', x_3, x_0), which corresponds to

$(8\ 9)(12\ 13)$ applied to π, returns $\lambda = (0\ 7\ 6\ 1\ 2\ 11\ 12\ 3\ 15\ 14\ 9\ 4)(5\ 8\ 10)(13)$ and $d_H(\pi) - d_H(\lambda) = 34 - 36 = -2$, so gate C^{2}NOT$(x_1', x_3, x_0)$ is a -2-move. Gate C^{2}NOT(x_1', x_3, x_0) applied to λ is a 2-move.

Definition 4 *A d-move is a useful move if and only if $d \geq 0$.*

So, in the first part of our new algorithm for synthesis of reversible circuits we use Partially GT gates. Initially, we apply GT gates with few controls, starting with NOT gates. First, all NOT gates are tested. The NOT gate that results in the larger d-move will be applied, where d is even and $0 < d \leq 2^n$. This procedure will be applied while the Hamming distance between the current permutation for identity decreases. Then, all CNOT gates are tested, and the CNOT gate that results in the largest d-move is applied, where d is even and $0 < d \leq 2^{n-1}$. Again, such a procedure will be applied while the Hamming distance between the current permutation for identity decreases. The same procedure is applied for the gates C^{2}NOT and for all other gates with more controls, until only Totally GT gates are applied in the second part of the algorithm. In this case, the applied d-move is such that $d = 0$ or $d = 2$.

Algorithm 1 below is the first part of our main algorithm, its library is composed only of GT gates with at most $n - 2$ controls. The $L(c)$ is the library of Toffoli gates with exactly c controls.

3.2 Second Part: Totally GT gates

The next lemmas will be used in the second part of our new algorithm, when only Totally GT gates will be used.

Lemma 5 *Let σ be the permutation resulting from the application of the $X(i, j)$ gate to π, and let $m(\sigma)$ be the number of cycles in its cycle representation. If i and j are on the same cycle in π, then $m(\sigma) = m(\pi) + 1$. Otherwise, $m(\sigma) = m(\pi) - 1$.*

Proof. In case $i = c^s$ and $j = c^t$ belong to the same cycle C, with $s < t$, the gate $X(i, j)$ splits C into two cycles, one defined by the sequence of elements $c^s, \ldots, c^{t-1}$, and the other defined by the remaining elements of C. On the other hand, in case $i = c_1^s$ and $j = c_2^t$ belong to distinct cycles C_1 and C_2, the gate $X(i, j)$ joins C_1 and C_2 into one cycle C defined by the sequence so that c_1^{s-1} precedes c_2^t, c_2^{t-1} precedes c_1^s and the other elements remain according to the order of C_1 and C_2. □

Given three elements i, j and k of a permutation, we say k is in a *minimum path* between i and j if $d_H(i, j) = d_H(i, k) + d_H(k, j)$. Moreover, notice that if $d_H(s, t) = 1$, then for every element u, either s is in a minimum path between u and t, or t is in a minimum path between s and u. Notice that, in case values i and j are adjacent, we have $d_H(i, j) = 1$ and for every other element u, $|d_H(i, u) - d_H(j, u)| \leq 1$.

Lemma 6 *Let π be a permutation such that π_i and π_j are adjacent values and consider the following possible situations: (a) π_j is in a minimum path between i and π_i; (b) π_i is in a minimum path between j and π_j. If both conditions (a) and (b) are simultaneously satisfied, then gate $X(\pi_i, \pi_j)$ applied to π is a 2-move. If only a single condition (a) or (b) is satisfied, then gate $X(\pi_i, \pi_j)$ applied to π is a 0-move. If neither condition (a) nor condition (b) are satisfied, then gate $X(\pi_i, \pi_j)$ applied to π is a -2-move.*

Algorithm 1: First Part: Partially GT gates

Input: a permutation π_0 representing a bijective function.
Output: a permutation π such that $d_H(\pi) \leq d_H(\pi_0)$ and a circuit G that transforms π_0 into π.

```
1  π ← π_0;
2  G ← ∅;
3  for c ← 0 to n − 2 do
4      continue ← True;
5      while continue do
6          d_max ← 0;
7          for g ∈ L(c) do
8              d ← d_H(π) − d_H(g(π));
               /* See Lemma 3                                   */
9              if d > d_max then
10                 d_max ← d;
11                 g_max ← g;
12             end
13         end
14         if d_max = 0 then
15             continue ← False;
16         else
17             update π and add g_max in G;
18         end
19     end
20 end
21 return π and G;
```

Proof. Let π' be the permutation resulting from the application of $X(\pi_i, \pi_j)$ to the permutation π. The only elements modified by the application of this gate are π_i and π_j, therefore $d_H(\pi, \pi') = d_H(i, \pi_j) + d_H(j, \pi_i) - d_H(i, \pi_i) - d_H(j, \pi_j)$.

First, let us consider the case in which both conditions (a) and (b) are simultaneously satisfied. In this case, $d_H(i, \pi_i) = d_H(i, \pi_j) + d_H(\pi_j, \pi_i)$ and $d_H(j, \pi_j) = d_H(j, \pi_i) + d_H(\pi_i, \pi_j)$. Therefore, $d_H(i, \pi_i) = d_H(i, \pi_j) + 1$ and $d_H(j, \pi_j) = d_H(j, \pi_i) + 1$. Thus, $d_H(\pi, \pi') = 2$, indicating that $X(\pi_i, \pi_j)$ is a 2-move.

Now, let us consider the case in which condition (a) is satisfied and condition (b) is not. In this case, $d_H(i, \pi_i) = d_H(i, \pi_j) + d_H(\pi_j, \pi_i)$ and π_j is in the minimum path between j and π_i. Then, $d_H(j, \pi_i) = d_H(j, \pi_j) + d_H(\pi_j, \pi_i)$. Therefore, $d_H(i, \pi_j) = d_H(i, \pi_i) - 1$ and $d_H(j, \pi_i) = d_H(j, \pi_j) + 1$. Thus, $d_H(\pi, \pi') = 0$, indicating that $X(\pi_i, \pi_j)$ is a 0-move.

The analyses of the remaining cases are analogous. □

Lemma 7 *If $\pi \neq \iota$ is not unicyclic, then there is a useful move corresponding to some Totally* GT *gate which joins cycles.*

Proof. The only permutation that does not allow a 0-move nor a 2-move is the identity.

Let us consider a not unicyclic permutation $\pi \neq \iota$ which does not allow a 2-move. This implies that π has more than one cycle and there is at least one cycle with two distinct elements. Then, there is a cycle in π in which there are two consecutive elements c^i and c^{i+1} such that there is an element c' in another cycle that belongs to the minimum path between c^i and c^{i+1}. □

For example, the gate $X(2,3)$ applied to $\pi = (0\ 3\ 1)(2)$ returns $\lambda = (0\ 2\ 3\ 1)$.

Lemma 8 *Let $C = (c^1 c^2 \cdots c^{|C|})$ be a cycle with at most one pair of neighbor elements that are not adjacent. Then there is a sequence of $|C| - 1$ Totally* GT *gates which transforms this cycle into $|C|$ unitary cycles.*

Proof. Without loss of generality, consider a cycle $C = (c^1 c^2 \cdots c^{|C|})$ such that the only possible not adjacent pair is $(c^{|C|} c^1)$. Applying Totally GT gate $X(c^1, c^2)$, this cycle C is split in two cycles $C_1 = (c^1)$ and $C' = (c^2 \cdots c^{|C|})$. After applying the sequence of gates $X(c^2, c^3)$, $X(c^3, c^4), \ldots, X(c^{|C|-1}, c^{|C|})$, we obtain the unitary cycles $(c^1), (c^2), \ldots, (c^{|C|})$. □

For example, let $\pi = (0\ 1\ 3\ 7\ 6)(2)(4)(5)$. Then, the sequence of gates $X(0,1)$, $X(1,3)$, $X(3,7)$ and $X(6,7)$ transforms π into ι.

Given a cycle C of length $|C|$ in a permutation π, we define

$$P(C) = \frac{S(C)}{|C|}, \tag{3}$$

and

$$P(\pi) = \sum_{i=1}^{m} P(C_i). \tag{4}$$

For example, let $\pi = C_1 C_2 = (0\ 3\ 1)(2)$. We have that $S(C_1) = 4$ and $S(C_2) = 0$, therefore $P(C_1) = 1.33$, $P(C_2) = 0$ and $P(\pi) = 1.33$.

Lemma 9 *Let σ be the permutation resulting from the application of gate $X(i,j)$ to π. If $X(i,j)$ is a useful move that joins cycles in π, then we have that $P(\sigma) < P(\pi)$.*

Proof. Let $X(i,j)$ be a d-move gate that can be applied to π such that $i \in C'$ and $j \in C'' \neq C'$, creating a new cycle C''' with the union of the two cycles. Let $P(C') = S(C')/|C'|$ and $P(C'') = S(C'')/|C''|$ and $P(C''') = S(C''')/|C'''|$. Since $S(C''') = S(C') + S(C'') - d$ and $|C'''| = |C'| + |C''|$, we have that

$$\begin{aligned} P(C''') &= \frac{S(C') + S(C'') - d}{|C'| + |C''|} \\ &= \frac{P(C') \cdot |C'|}{|C'| + |C''|} + \frac{P(C'') \cdot |C''|}{|C'| + |C''|} - \frac{d}{|C'| + |C''|}. \end{aligned} \tag{5}$$

Considering that $|C'| < |C'| + |C''|$ and $|C''| < |C'| + |C''|$, if $d = 0$ or $d = 2$, then we have that $P(C''') < P(C') + P(C'')$. Since the only cycles modified in π were C' and C'', we have that $P(\sigma) = P(\pi) + P(C''') - P(C') - P(C'')$. Therefore $P(\sigma) < P(\pi)$. □

Definition 10 *A sequence of useful moves is a useful sequence if it decreases $d_H(\pi)$ or $P(\pi)$.*

In order to finish the analysis of the algorithm 4 convergence, we need to see the case in which the permutation is unicyclic and there is no 2-move. For this goal, we use the next procedures and lemmas.

Procedure 2: Replace(i, j)

Input: $i = i_{n-1} \cdots i_0, j = j_{n-1} \cdots j_0 \in \pi$ $\{i_k$ is the bit k of $i\}$
Output: $T(\pi) = X(i, j)\pi$.

```
for k ← 0 to n − 1 do
    if i_k ≠ j_k then
        apply the gate X(i, i'), where i' is the adjacent element of i s.t. i'_k = j_k;
        i ← i';
    end
end
```

Lemma 11 *Procedure 2 puts j in the position of i in π, going through all the elements of a minimum path between i and j.*

Proof. In each step of the loop, in which the condition of 'if' is true, i' has the same bits of i except bit k, and $d_H(i', j) = d_H(i, j) - 1$, therefore i' is in a minimum path between i and j. After applying the gate $X(i, i')$, i' stays in position of π occupied by i. After applying $d_H(i, j)$ gates, the element j occupies the place of original i. The last i' is j. □

Lemma 12 *Let i and j be elements of π, such that i is successor of j in a cycle, then after Replace(i, j) there is a unitary cycle (j).*

Proof. If i is successor of j in a cycle then i is in position j ($\pi_j = i$). Therefore, Replace(i, j) puts j in position j. □

Procedure 3: Useful sequence with 2-move

Input: Current permutation π.
Output: A useful sequence with 2-move.

```
j ← 2^n − 1;
repeat
    let i be successor of j in the respective cycle, Replace(i, j);
    j ← j − 1;
until Replace(i, j) has a 2-move;
```

Lemma 13 *Procedure 2 outputs a useful sequence.*

Proof. If j is in a unitary cycle then successor of j is j and so Replace(i, j) has no move, else all moves are useful moves. Indeed, in each gate of Replace(i, j), the present i' is in a minimum path between j and π_j so, by Lemma 6, this move is a useful move.

This Procedure converges because, in each iteration, the elements greater than j are in unitary cycles and they do not belong to the minimum path between i and j used by Replace(i, j). □

Algorithm 4 is used when it is not possible to apply partially controlled GT gates that decrease the Hamming distance of the permutation.

Algorithm 4: Second Part: Totally GT gates

Input: a permutation π_0 representing a bijective function.
Output: the circuit G that transforms π_0 into ι.

```
1  π ← π_0;
2  G ← ∅;
3  while d_H(π) > 0 do
4      if there is a 2-move that joins cycles then
5          choose X(c^i, j) corresponding to the 2-move;
6          apply the corresponding reversible gate;
7          update π and G;
8      else if there is a 2-move that splits cycles then
9          apply the corresponding reversible gate;
10         update π and G;
11     else if there is a sequence of gates that splits cycle in unitary cycles then
12         apply the sequence of gates;
13         update π and G;
           /* See Lemma 8                                          */
14     else if there is a 0-move that joins cycles then
15         choose X(c^i, j) corresponding to the 0-move;
16         apply the corresponding reversible gate;
17         update π and G;
           /* Necessarily decreases the value of P(π), see
              Lemma 9                                              */
18     else
           /* π is unicyclic, see Lemma 7                          */
19         apply Procedure 2;
20         update π and G;
21     end
22 end
23 return G;
```

Theorem 14. *Correctness of Algorithm 4.*

Proof. At each step, the algorithm either applies a 2-move or a 0-move minimizing $P(\pi)$, see Lemmas 5 to 9 and Lemma 13. Therefore, at each step the Hamming distance between π and ι is kept constant or decreases. When the Hamming distance is kept constant, the function $P(\pi)$ decreases until the permutation is unicyclic, see Lemmas 5, 7 and 9. In this case, the algorithm applies a useful sequence that applies a 2-move, see Lemma 13. Thus, the permutation always converges to the identity permutation ι. □

3.3 Algorithm with progressive increase of controls in GT gates

Our synthesis method uses gates with $0, 1, \ldots, n-1$ controls, increasing the number of controls progressively. The synthesis algorithm with progressive increase of controls in GT gates (Algorithm 5) summarized uses Algorithms 1 and 4. See Algorithm 5 for a sketch of our synthesis method using gates with $0, 1, \ldots, n-1$ controls.

Algorithm 5: Synthesis algorithm with progressive increase of controls in GT gates

Input: a permutation π_0 representing a bijective function.
Output: the circuit G that transforms ι into π_0.
1 $(\pi, G_1) \leftarrow$ apply Algorithm 1 to π_0;
2 $G_2 \leftarrow$ apply Algorithm 4 to π;
3 $G \leftarrow reverse(G_1 G_2)$;
4 return G;

Theorem 15. *Correctness of Algorithm 5.*

Proof. In the first steps, for each quantity k of controls, $k \in \{0, 1, \ldots, n-2\}$, the algorithm applies d-moves ($d > 0$, largest possible, according to Lemma 3), such that the Hamming distance between the current permutation for identity decreases as much as possible. When arriving at $k = n-1$ controls, just use Theorem 14. □

Figure 5 depicts the circuit that transforms identity into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, for the case where the input of Algorithm 5 is $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, considered in the example in Figure 3. Note that the order of the returned gates is reversed in relation to the order in which the gates are obtained.

Algorithm 5 produces the circuit by selecting generalized Toffoli gates that manipulate the output side of the circuit. Since the permutation is reversible, we can define strategies that use Algorithm 5 to improve the gate count.

Strategy 1 (Synthesis of the Inverse) *Apply an algorithm to the input permutation and its inverse. Then, choose the circuit with the least number of gates.*

Strategy 2 (Bidirectional Synthesis) *Apply an algorithm simultaneously in both directions. Choose to add input-side or output-side gates of the circuit at each step of the synthesis.*

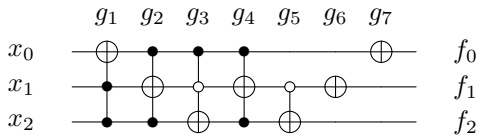

Figure 5: Example of a reversible circuit returned by Algorithm 5, that transforms the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. *The output bits are denoted by* $f_2 f_1 f_0$. *The sequence of gates (reading from left to right) is* $g_1 = \textit{Toffoli}(x_1, x_2, x_0)$, $g_2 = \textit{Toffoli}(x_0, x_2, x_1)$, $g_3 = \textit{Toffoli}(x_0, x_1', x_2)$, $g_4 = \textit{Toffoli}(x_0, x_2, x_1)$, $g_5 = \text{CNOT}(x_1', x_2)$, $g_6 = \text{NOT}(x_1)$ *and* $g_7 = \text{NOT}(x_0)$, *where* $g_7 g_6 g_5$ *and* $g_4 g_3 g_2 g_1$ *are the circuits returned by Algorithms 1 and 4 respectively.*

Figure 6 depicts the circuits that transform the identity into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, obtained with Strategy 1 and Strategy 2, respectively, where the input of Algorithm 5 is $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. As we did before, to clarify our use of equivalent representations, we include a corresponding Table 2 with the explicit sequence of used gates corresponding to the circuit depicted in Fig. 6b.

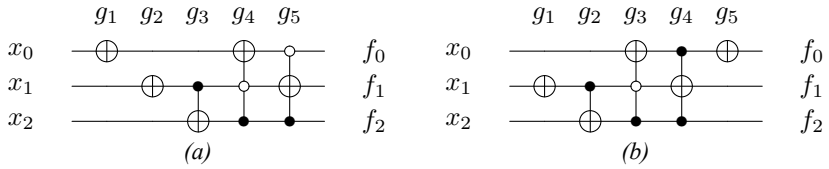

Figure 6: Example of reversible circuits, obtained from Algorithm 5 with (a) Strategy 1 and (b) Strategy 2, respectively, that transform the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. *The output bits are denoted by* $f_2 f_1 f_0$. *(a) The sequence of gates is* $g_1 = \text{NOT}(x_0)$, $g_2 = \text{NOT}(x_1)$, $g_3 = \text{CNOT}(x_1, x_2)$, $g_4 = \textit{Toffoli}(x_1', x_2, x_0)$ *and* $g_5 = \textit{Toffoli}(x_0', x_2, x_1)$. *(b) The sequence of gates is* $g_1 = \text{NOT}(x_1)$, $g_2 = \text{CNOT}(x_1, x_2)$, $g_3 = \textit{Toffoli}(x_1', x_2, x_0)$, $g_4 = \textit{Toffoli}(x_0, x_2, x_1)$ *and* $g_5 = \text{NOT}(x_0)$.

4 Experimental Results

Firstly, we have considered all 3-bit bijective functions and compared our results with different synthesis algorithms. The results are in Table 3. Column **Size** shows the number of gates of the circuit. The main current results appear in two columns. Column **CGWZ** shows the results of synthesis simplification algorithm by Cheng *et al.* [Cheng et al., 2012]. Column **ZLZPZ** shows the results of cycle-decomposition-based synthesis optimized algorithm by Zhu *et al.* [Zhu et al., 2018]. The results of applying Algorithm 5 to all 3-bit bijective functions are in columns **(a)**, **(b)** and **(c)** according to different strategies as follows:

	$x_2x_1x_0$					$f_2f_1f_0$
	000	0**1**0	**1**10	110	110	11**1** = 7
	001	0**1**1	**1**11	111	1**0**1	10**0** = 4
	010	0**0**0	000	000	000	00**1** = 1
	011	0**0**1	001	001	001	00**0** = 0
	100	1**1**0	**0**10	010	010	01**1** = 3
	101	1**1**1	**0**11	011	011	01**0** = 2
	110	1**0**0	100	10**1**	1**1**1	11**0** = 6
	111	1**0**1	101	10**0**	100	10**1** = 5
Gates:	g_1 ↗	g_2 ↗	g_3 ↗	g_4 ↗	g_5 ↗	

Table 2: Sequence of gates $g_1 = \text{NOT}(x_1)$, $g_2 = \text{CNOT}(x_1, x_2)$, $g_3 = \textit{Toffoli}(x_1', x_2, x_0)$, $g_4 = \textit{Toffoli}(x_0, x_2, x_1)$ *and* $g_5 = \text{NOT}(x_0)$ *that transforms the identity permutation into* $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ *using the Algorithm 5 and Strategy 2, see the corresponding circuit in Fig. 6b. In bold, the bit changes introduced by the corresponding gates. The result of application of the gate on the bottom of each column is shown in the next column while reading from left to right.*

– In column (a), application of Algorithm 5.

– In column (b), application of Algorithm 5 improved by Strategy 1.

– In column (c), application of Algorithm 5 improved by Strategy 1 plus Strategy 2, followed by choosing the minimum value between the two results.

The improved Algorithm 5 considered in column (c) obtained the best number 5.23 of gates in average. The result 5.38 obtained by the algorithm of Cheng *et al.* [Cheng et al., 2012] also considers Strategy 1 of their simplification algorithm, in addition to the use of templates. Our Algorithm 5 considered in column (a) obtained 5.82 gates in average, improving the result of Zhu *et al.* [Zhu et al., 2018], which is also a unidirectional algorithm. Moreover, Zhu *et al.* algorithm is specific to 3 bits while our Algorithm 5 works for an arbitrary number of bits. The main idea in Zhu *et al.* algorithm is to decompose cycles in permutations and use the so-called Head-Pointer-Adjust, which takes the greatest Hamming distance between elements of the cycle.

Our method is an upgrade of the cycle-based synthesis (CBS) algorithm of Ribeiro *et al.* [Ribeiro et al., 2015], including Partially GT gates at the library, and improves the average number of gates, as follows. The average number of gates necessary to synthesize a circuit is reduced from 6.74 with unidirectional CBS algorithm to 5.82 with Algorithm 5 considering column (a), and is reduced from 6.64 with CBS algorithm applying Strategy 1 to 5.32 with Algorithm 5 applying Strategy 1 considering column (b).

Secondly, we have performed a series of experiments on reversible benchmark function synthesis. The results are in Table 4. The first and the second columns show the name of each **benchmark function** and its **size** (number of variables) considered in the literature, respectively. The third column shows the results obtained by the proposed approach, using the Algorithm 5 and strategies 1 and 2. The fourth column shows the best results for the gate count of synthesis algorithm by Zakablukov [Zakablukov, 2016] which uses properties of Permutation Group Theory to reduce gate complexity and combines

	Algorithm 5			Current literature	
Size	(a)	(b)	(c)	CGWZ	ZLZPZ
0	1	1	1	1	1
1	27	27	27	27	15
2	309	369	369	369	129
3	1797	2505	2601	2633	753
4	5376	7586	8114	7624	3100
5	9758	12932	12994	11263	8409
6	10529	9940	10066	10258	13405
7	7046	4523	4652	5963	10506
8	3922	2166	1450	1716	3625
9	1206	213	24	372	369
10	319	54	18	93	8
11	30	4	4	1	0
AG	5.82	5.32	5.23	5.38	6.03

Table 3: Number of bijective functions using a specified number of gates, considering all 3*-bit bijective functions for different synthesis algorithms as indicated by citations. Our results are listed in columns (a), (b) and (c). The main current results using* GT *library are in columns* **CGWZ** *[Cheng et al., 2012] and* **ZLZPZ** *[Zhu et al., 2018]. Row AG reports the average number of gates necessary to synthesize a circuit.*

cycle-based and Reed-Muller-spectra-based algorithms. The last two columns show the best results for the gate count of MMD method without/with template matching by Maslov *et al.* [Maslov et al., 2005], and the MMD method with the Reed-Muller spectra by Maslov *et al.* [Maslov et al., 2007], respectively. All specifications for benchmark function and their names were taken from the *RevLib site*[1] [Wille et al., 2008] and from the *Reversible Logic Synthesis Benchmarks Page*[2].

Analyzing the results obtained in Table 4, we see that our approach obtained in more than half of cases, best results in relation to those obtained by Zakablukov [Zakablukov, 2016] and Maslov *et al.* [Maslov et al., 2005] for benchmark function synthesis. We were able to get 9 (out of 17) reversible circuits, which have less or equal gate count in relation to the two cited references. Comparing our results with those obtained by Zakablukov, we see that we obtained better results in 5 out of 9 comparisons, being that, our synthesis algorithm performed better as we increased the number of bits. On the other hand, comparing with the results obtained by Maslov *et al.*, we see that we obtained better results in 7 out of 12 comparisons (1 equal), being that, in general our performance was better for functions up to 7 bits. However, the results obtained by Maslov *et al.* [Maslov et al., 2007] based on MMD method [Maslov et al., 2005] using templates with Reed-Muller spectra are the best results so far.

Thirdly, we have made available at https://github.com/Marquezino/dkmfr an implementation of our main result to the community. In order to get a sense to what extent 8-bit permutations can be handled, please see in Figure 7 a graph where we have depicted the corresponding actual obtained running times. The obtained linear graph in the base-10

[1] available at http://www.revlib.org
[2] available at http://webhome.cs.uvic.ca/∼dmaslov/

benchmark function	size	**Algorithm 5**	Z-16	MMD-05	MMD-07
3_17	3	6	4	6/6	6
4_49	4	14	-	16/16	12
4b15g_2	4	17	12	-	-
4b15g_4	4	19	12	-	-
4b15g_5	4	18	14	-	-
aj−e11	4	9	-	13*/−	-
ham3	3	6	-	6/5	5
hwb4	4	18	-	18/17	11
hwb5	5	43	-	57/55	24
hwb6	6	103	-	134/126	42
hwb7	7	282	603	302/289	236
hwb8	8	697	1594	688/−	614
hwb9	9	2633	3999	1625/−	1541
mod5adder	6	17	-	37/−	15
mod5mils	5	3	-	5/−	-
nth_prime4_inc	4	13	-	-	12**
nth_prime5_inc	5	38	-	-	25**
nth_prime6_inc	6	79	-	-	55**
nth_prime7_inc	7	231	427	-	-
nth_prime8_inc	8	627	977	-	-

* Result based on MMD method executed by [Große et al., 2009] to generate a heuristic result.
** Result available at http://webhome.cs.uvic.ca/∼dmaslov/

Table 4: Benchmark Function Synthesis. Our results are listed in column **Algorithm 5**. *The main current results are in columns* **Z-16** *[Zakablukov, 2016],* **MMD-05** *[Maslov et al., 2005] and* **MMD-07** *[Maslov et al., 2007].*

logarithmic scale agrees with the expected exponential behavior as we increase the number of bits.

5 Concluding remarks

In this work, we presented a new algorithm of reversible circuit synthesis using generalized Toffoli gates (GT gates). Our Algorithm 5 is an upgrade of CBS algorithm [Ribeiro et al., 2015], including Partially GT gates at the library. The new approach progressively increases the number of Toffoli gate controls. Besides that, our Algorithm 5 explores properties of the cycle representations of permutations as part of the process and uses bidirectional strategies. The circuits achieved by our new synthesis algorithm in general use less gates than the circuits achieved by the CBS algorithm. Our Algorithm 5 works for arbitrary n-bit bijective functions, whereas the exact synthesis found in the literature work only for the cases of $n = 3$ and $n = 4$ bits [Wille et al., 2012, Szyprowski and Kerntopf, 2012]. Considering that the quantum cost [Barenco et al., 1995] of each GT gate is proportional to the number of control bits, the quantum cost of circuits generated by Algorithm 5 is, in average, less than the quantum cost of circuits generated by CBS

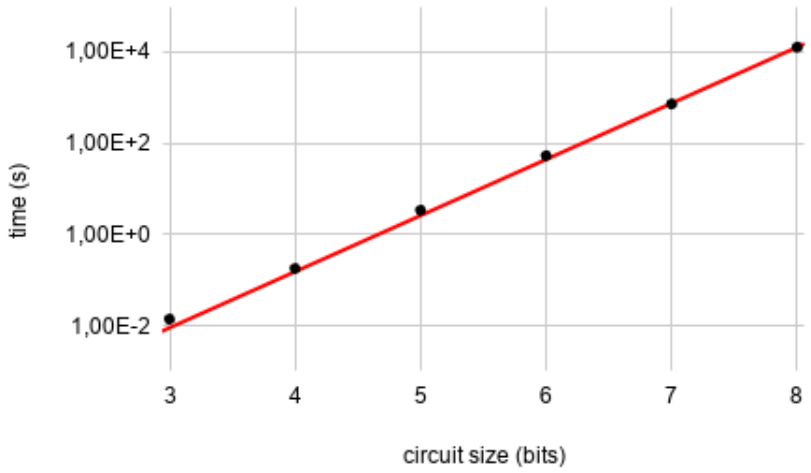

*Figure 7: Actual obtained running times (in seconds) of Algorithm 5. The time-axis is represented in the base-*10 *logarithmic scale. We depict in red the trend line with slope 6/5.*

algorithm. In the present work, we consider as a complexity metric just the number of gates, a cost model frequently used in the literature and considered adequate when staying within the traditional reversible framework [Saeedi and Markov, 2013].

The main contributions include the best so far average gate count for all 3-bit bijective functions for the GT library. Besides that, we present experimental results for reversible benchmark function synthesis, which include twenty reversible circuits consisting of gates from GT library, that are competitive when compared with the best results for the gate count of reversible synthesis heuristics in the literature.

As future research, possible directions include using templates to replace by shorter ones certain sequences of gates in the circuits returned by Algorithm 5. Moreover, we could use some other approaches such as permutation group theory and Reed-Muller spectra [Maslov et al., 2007, Zakablukov, 2016] in order to find better gate sequences. We could also apply similar techniques to the synthesis of quantum circuits [de Almeida et al., 2019], taking into account that moving to the level of quantum circuits requires a more fine-grained gate set than considered here and optimizations that align with error-correction needs.

Acknowledgements

The authors wish to thank the editor for handling our submission during the pandemic, and the two anonymous referees for the diligent reading and several useful suggestions. This work was partially supported by CAPES, CNPq and FAPERJ.

References

[Barenco et al., 1995] Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J. A., and Weinfurter, H. (1995). Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467.

[Bennett, 1973] Bennett, C. H. (1973). Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532.

[Cheng et al., 2012] Cheng, X., Guan, Z., Wang, W., and Zhu, L. (2012). A simplification algorithm for reversible logic network of positive/negative control gates. In *9th International Conference on Fuzzy Systems and Knowledge Discovery*, pages 2442–2446.

[Datta et al., 2013] Datta, K., Rathi, G., Wille, R., Sengupta, I., Rahaman, H., and Drechsler, R. (2013). Exploiting negative control lines in the optimization of reversible circuits. In Dueck, G. W. and Miller, D. M., editors, *Reversible Computation*, pages 209–220, Berlin, Heidelberg. Springer.

[de Almeida et al., 2019] de Almeida, A. A. A., Dueck, G. W., and da Silva, A. C. R. (2019). Efficient realization of Toffoli and NCV circuits for IBM QX architectures. In *Reversible Computation - 11th International Conference, RC 2019, Lausanne, Switzerland, June 24-25, 2019, Proceedings*, pages 131–145.

[Golubitsky et al., 2010] Golubitsky, O., Falconer, S. M., and Maslov, D. (2010). Synthesis of the optimal 4-bit reversible circuits. In *Design Automation Conference*, pages 653–656.

[Große et al., 2009] Große, D., Wille, R., Dueck, G. W., and Drechsler, R. (2009). Exact multiple-control Toffoli network synthesis with SAT techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5):703–715.

[Iwama et al., 2002] Iwama, K., Kambayashi, Y., and Yamashita, S. (2002). Transformation rules for designing CNOT-based quantum circuits. In *Proceedings of the 39th Annual Design Automation Conference*, pages 419–424, New York, NY, USA. ACM.

[Kowada et al., 2006] Kowada, L. A. B., Portugal, R., and de Figueiredo, C. M. H. (2006). Reversible Karatsuba's algorithm. *Journal of Universal Computer Science*, 12(5):499–511.

[Landauer, 1961] Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191.

[Maslov and Dueck, 2004] Maslov, D. and Dueck, G. W. (2004). Reversible cascades with minimal garbage. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(11):1497–1509.

[Maslov et al., 2005] Maslov, D., Dueck, G. W., and Miller, D. M. (2005). Toffoli network synthesis with templates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):807–817.

[Maslov et al., 2007] Maslov, D., Dueck, G. W., and Miller, D. M. (2007). Techniques for the synthesis of reversible Toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4).

[Miller et al., 2003] Miller, D. M., Maslov, D., and Dueck, G. W. (2003). A transformation based algorithm for reversible logic synthesis. In *Proceedings of the 40th Annual Design Automation Conference*, pages 318–323, New York, NY, USA. ACM.

[Nielsen and Chuang, 2000] Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press, New York, NY, USA.

[Rahman and Rice, 2014] Rahman, M. Z. and Rice, J. E. (2014). Templates for positive and negative control Toffoli networks. In Yamashita, S. and Minato, S.-I., editors, *Reversible Computation*, pages 125–136, Cham. Springer International Publishing.

[Ribeiro et al., 2015] Ribeiro, A. C., Kowada, L. A. B., Marquezino, F. L., and Figueiredo, C. M. H. (2015). A new reversible circuit synthesis algorithm based on cycle representations of

permutations. *Electronic Notes in Discrete Mathematics*, 50:187–192. LAGOS'15–VIII Latin-American Algorithms, Graphs and Optimization Symposium.

[Saeedi and Markov, 2013] Saeedi, M. and Markov, I. L. (2013). Synthesis and optimization of reversible circuits - a survey. *ACM Comput. Surv.*, 45(2):21:1–21:34.

[Saeedi et al., 2010] Saeedi, M., Zamani, M. S., Sedighi, M., and Sasanian, Z. (2010). Synthesis of reversible circuit using cycle-based approach. *J. Emerg. Technol. Comput. Syst*, 6(4):13.1–13.26.

[Shende et al., 2003] Shende, V. V., Prasad, A. K., Markov, I. L., and Hayes, J. P. (2003). Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722.

[Szyprowski and Kerntopf, 2012] Szyprowski, M. and Kerntopf, P. (2012). A study of optimal 4-bit reversible circuit synthesis from mixed-polarity Toffoli gates. In *12th IEEE International Conference on Nanotechnology (IEEE-NANO)*, pages 1–6.

[Toffoli, 1980] Toffoli, T. (1980). Reversible computing. In de Bakker, J. and van Leeuwen, J., editors, *Automata, Languages and Programming*, page 632, New York. Springer. MIT Technical Memo No. MIT/LCS/TM-151, 1980 (unpublished).

[Wille et al., 2008] Wille, R., Große, D., Teuber, L., Dueck, G. W., and Drechsler, R. (2008). Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic*, pages 220–225.

[Wille et al., 2012] Wille, R., Soeken, M., Przigoda, N., and Drechsler, R. (2012). Exact synthesis of Toffoli gate circuits with negative control lines. In *IEEE 42nd International Symposium on Multiple-Valued Logic*, pages 69–74.

[Zakablukov, 2016] Zakablukov, D. V. (2016). Application of permutation group theory in reversible logic synthesis. In Devitt, S. and Lanese, I., editors, *Reversible Computation*, pages 223–238, Cham. Springer International Publishing.

[Zhu et al., 2018] Zhu, W., Li, Z., Zhang, G., Pan, S., and Zhang, W. (2018). A reversible logical circuit synthesis algorithm based on decomposition of cycle representations of permutations. *International Journal of Theoretical Physics*, 57(8):2466–2474.