CONSTRUÇÃO DE ALGORITMOS REVERSÍVEIS E QUÂNTICOS

Luis Antonio Brasil Kowada

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO 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.

Aprovada por:

Profª. Celina Miraglia Herrera de Figueiredo, D.Sc.

Prof. Renato Portugal, D.Sc.

Prof. Carlile Campos Lavor, D.Sc.

Prof. Luiz Mariano Paes de Carvalho Filho, Docteur

Prof. Valmir Carneiro Barbosa, Ph.D.

Prof. Raul José Donangelo, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
MARÇO DE 2006

KOWADA, LUIS ANTONIO BRASIL

Construção de Algoritmos Reversíveis e Quânticos [Rio de Janeiro] 2006

XIII, 136 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2006)

Tese – Universidade Federal do Rio de Janeiro, COPPE

1 - Algoritmos Reversíveis e Quânticos

2 - Computação Quântica

3 - Otimização Combinatória Discreta

4 - Implementação de Operadores Aritméticos

I. COPPE/UFRJ II. Título (série)

A Deus, meu grande Oráculo.

Agradecimentos

Agradeço a Deus e a todas as pessoas que, direta ou indiretamente, me ajudaram nesta caminhada.

Em particular, a meus pais e irmãos, por tudo o que sempre fizeram por mim.

À professora Celina, por ter aceitado o desafio de me orientar em uma área tão distante da sua linha de pesquisa, e por dado todo o apoio necessário para a realização desta tese.

Ao professor Renato Portugal, que se deslocou tantas vezes de Petrópolis, orientando, junto com a Celina, todos os passos deste trabalho.

Ao professor Carlile Lavor, que também acompanhou de perto este trabalho e deu um inestimável apoio, principalmente na parte que se refere ao algoritmo de otimização combinatória.

Ao professor Raul Donangelo, que se mostrou sempre afável e disposto a ajudar e ao professor Valmir Barbosa que deu valiosas contribuições para este trabalho, no exame de qualificação.

Ao professor Luiz Mariano, por ter aceitado participar da banca de defesa desta tese.

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

Agradeço ao CNPq, pelo suporte financeiro.

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

CONSTRUÇÃO DE ALGORITMOS REVERSÍVEIS E QUÂNTICOS

Luis Antonio Brasil Kowada

Março/2006

Orientadores: Celina Miraglia Herrera de Figueiredo
Renato Portugal (LNCC)

Programa: Engenharia de Sistemas e Computação

Este trabalho consiste na proposta de algoritmos aritméticos reversíveis e algoritmos quânticos para problemas de otimização combinatória. Ambos os tipos podem ser utilizados na construção de outros circuitos e algoritmos.

Mostramos como implementar circuitos reversíveis para operações aritméticas, tais como multiplicação e divisão de inteiros não-negativos, que podem ser utilizados para operações quânticas com ambos operandos em superposição. Em particular, são propostas versões reversíveis do algoritmo de multiplicação Karatsuba [18], com procedimento de limpeza de lixo recursivo.

Propomos um algoritmo quântico, denominado *Medida-Linear*, para encontrar um valor ótimo em uma lista não-ordenada. Para uma lista com N elementos, são feitas $8,27\sqrt{N}$ chamadas do oráculo, a fim de obter pelo menos 50% de chance de sucesso. O algoritmo quântico mais eficiente que existe atualmente, DH96 [12], realiza a mesma tarefa com $22,5\sqrt{N}$ chamadas do oráculo. Além do mais, o nosso algoritmo Medida-Linear necessita de $\Theta(n)$ medições em contraposição às $\Theta(n^2)$ medições do DH96, onde n é a quantidade de bits necessária para representar o tamanho da lista. Propomos também um operador quântico que permite adaptar o DH96 [12] e o algoritmo Medida-Linear para resolver problemas de otimização combinatória.

Foram realizadas simulações comparando os algoritmos acima descritos para encontrar o menor elemento de uma lista ou retornado por uma função. A comparação foi feita principalmente sobre o número de consultas à lista e a quantidade de medições realizadas. Em todas as simulações, o algoritmo Medida-Linear teve desempenho superior ao DH96, confirmando também a análise teórica de que o algoritmo Medida-Linear necessita de uma quantidade linear de medições dos resultados parciais enquanto que o algoritmo DH96 necessita de uma quantidade quadrática destas medições.

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

DESIGN OF REVERSIBLE AND QUANTUM ALGORITHMS

Luis Antonio Brasil Kowada

March/2006

Advisors: Celina Miraglia Herrera de Figueiredo
Renato Portugal (LNCC)

Department: Systems Engineering and Computer Science

In this work, we propose reversible arithmetic algorithms and quantum algorithms for combinatorial optimization problems. Both types of algorithms can be used in the design of other circuits and algorithms.

We show some reversible circuits for non-negative integer operations with both operands in state superposition. We propose reversible circuits for integer multiplication and division. Besides the naive circuit for integer multiplication, we develop some versions for Karatsuba algorithm [18].

We propose a quantum algorithm, denoted Medida-linear, that makes $8.27\sqrt{N}$ oracle calls for finding the smallest (or largest) value with 50% success in an unsorted list with N elements. The DH96 algorithm [12] is nowadays the most efficient quantum algorithm for solving this problem. It needs $22.5\sqrt{N}$ oracle calls. Another difference is the number of measurements: Medida-linear makes $\Theta(n)$ measurements and DH96 makes $\Theta(n^2)$, where n is the number of bits needed to represent the list size.

We simulated the above algorithms to find the smallest element of a list. A comparison was made considering the number of oracle calls in the list and the number of measurements. In all simulations, the performance of Medida-Linear was better than DH96. The simulations confirmed the linear complexity of measurements of Medida-Linear and the quadratic complexity of DH96.

Sumário

Lista de Algoritmos

Lista de Figuras

Capítulo 1

Introdução

Segundo a lei de Moore, hipótese formulada na década de 60 por Gordon Moore, fundador da empresa Intel, a cada 18 meses a capacidade de processamento dobra e o tamanho necessário para a representação de um bit se reduz pela metade. Em 1950, por exemplo, eram necessários 10^{19} átomos para representar um único bit de informação [35]. Até agora, esta estimativa tem se comprovado na prática. Se continuar neste ritmo, em 2045, cada bit será representado por um único átomo. Nesta escala, os efeitos quânticos não podem ser desprezados e, portanto, propriedades da computação convencional, como a possibilidade de cópia de dados, não são válidas. Já atualmente, os efeitos quânticos não são desprezíveis [34, cap. 1].

Em paralelo a este desenvolvimento tecnológico, o modelo de Computação Quântica tem se mostrado mais forte do que os modelos clássicos, pois, resolve todos problemas computáveis, com a mesma ordem de complexidade de tempo que um computador clássico, ou menor. Por exemplo, em relação ao problema da fatoração de um número inteiro, até o momento, não se conhece algoritmo clássico para resolvê-lo em tempo polinomial, mas existe algoritmo quântico polinomial para isso [40]. Detalhes deste algoritmo podem ser vistos em [25, 36], entre outros. No capítulo 2, são mostrados os modelos de computação clássico e quântico, dando um destaque ao modelo

reversível que pode ser tanto clássico quanto quântico.

Apesar de haver um grande interesse da comunidade científica a respeito da Computação Quântica [5], têm havido poucos progressos no desenvolvimento de novos algoritmos [41, 42].

Qualquer problema que possa ser resolvido em um computador clássico, pode ser resolvido em um computador quântico com a mesma ordem de complexidade de tempo, conforme será comentado no capítulo 2. Mas a conversão de algoritmos clássicos para quânticos não é muito simples, como pode ser observado na prática pela existência de poucos algoritmos quânticos. Além do mais, como será visto adiante, a conversão direta nem sempre produz o melhor algoritmo.

Neste trabalho, propomos alguns algoritmos tanto reversíveis clássicos como quânticos. No capítulo 3, apresentamos alguns algoritmos para operações aritméticas com inteiros não negativos, e propomos implementações reversíveis dos algoritmos clássicos para multiplicação e divisão. Há algumas propostas para um problema similar, multiplicação modular de números inteiros, que possuem $\Theta(n^2)$ portas lógicas [46], mas que não permitem a superposição de estados quânticos em ambos operandos, conforme será explicado posteriormente.

A multiplicação é uma operação fundamental em diversos algoritmos, por isso a necessidade de algoritmos eficientes. A complexidade de tempo do algoritmo comumente usado nos computadores convencionais para esta operação é $\Theta(n^2)$, onde n é tamanho de cada operando. Há um algoritmo, chamado Karatsuba [18], que possui complexidade de tempo $\Theta(n^{\log_2 3})$, sendo mais eficiente que o algoritmo convencional para multiplicar grandes números. Este algoritmo é utilizado em sistemas de computação algébrica [15, cap. 8]. Apresentamos uma implementação reversível do algoritmo de multiplicação Karatsuba que pode ser utilizada em um computador quântico ou em um computador clássico reversível. O algoritmo Karatsuba reversível, que

estamos propondo, foi submetido para publicação no *Journal of Universal Computer Science* [23].

Além das operações aritméticas, outro problema tratado é o da busca do menor elemento dentro de um conjunto (capítulo 4). Dada uma lista não-ordenada com N elementos, existe um algoritmo quântico, DH96 [12], que retorna o menor valor desta lista, com pelo menos 50% de chance de sucesso, após $22,5\sqrt{N}$ consultas à lista e $\Theta(\log^2 N)$ medições intermediárias. Neste trabalho, propomos um novo oráculo, chamado Oráculo Desigualdade, que permite adaptar este algoritmo para que possa ser utilizado também para resolver problemas de otimização combinatória. A complexidade tanto de tempo quanto de espaço deste oráculo está relacionada com a complexidade da função certificadora de cada problema específico, ou seja, da função que retorna a solução relacionada a uma determinada entrada. Alterando também um parâmetro do algoritmo DH96 original, a complexidade de tempo da versão de otimização combinatória é $11,48\sqrt{N}$ execuções do Oráculo Desigualdade, para obter uma solução correta em pelo menos metade das execuções. Desenvolvemos um outro algoritmo, chamado de Medida-Linear, para encontrar o menor elemento de uma lista ou resolver problemas de otimização combinatória, com a mesma chance de sucesso do que o algoritmo DH96, após $8,27\sqrt{N}$ iterações do oráculo, fazendo apenas $O(\log_2 N)$ medições.

No capítulo 5, são apresentadas simulações comparando os algoritmos Medida-Linear na busca do menor elemento em listas geradas a partir de pequenas instâncias de dois problemas NP-difíceis. Também foram feitas comparações destes algoritmos para funções injetoras, que é o pior caso de ambos algoritmos; com instâncias de até 100 q-bits. Foram comparados o número de chamadas do oráculo e o número de medições realizadas até chegar ao menor elemento. Para ter outro referencial na comparação, também é mostrado o resultado para a simulação do algoritmo BBHT98, quando o valor mínimo é conhecido de antemão.

Ao longo do documento, sempre que se omitir a base do logaritmo, ela representa a base 2.

Capítulo 2

Modelos de Computação

2.1 Modelos de Computação Clássica

2.1.1 Máquina de Turing

Alan Turing, em 1936, formulou um modelo de computação [45] que ficou conhecido como máquina de Turing. O modelo consiste de: uma fita linear de comprimento infinito, uma peça que se posiciona em uma determinada posição da fita podendo ler e escrever nesta posição (cabeçote de leitura/gravação), podendo se deslocar para a esquerda ou para a direita, um conjunto de instruções para a movimentação do cabeçote, e um elemento que lê as instruções e comanda o cabeçote. Os possíveis valores em cada posição da fita são *zero* ou *um*.

Turing mostrou que há uma máquina, de acordo com o modelo acima descrito, que é universal, ou seja, é capaz de simular o comportamento de qualquer outra máquina de Turing.

Alonzo Church [8], simultaneamente e independentemente de Turing, desenvolveu uma hipótese de que qualquer procedimento algorítmico pode ser simulado em uma Máquina de Turing Universal. Esta tese ficou conhecida como Tese de Church-Turing. Posteriormente, foi demonstrado que muitos

modelos são equivalentes à maquina de Turing em relação à capacidade de processamento, e até o momento, não foi encontrado nenhum contra-exemplo da tese na forma que acabamos de descrevê-la. No capítulo 4, será comentado sobre uma versão mais forte desta tese.

2.1.2 Circuitos Combinacionais

Um modelo computacional equivalente à Máquina de Turing é formado pelos circuitos combinacionais. Um circuito lógico consiste de portas lógicas interconectadas (a saída de uma porta pode ser ligada à entrada de outra(s)). Cada porta lógica é uma função $f : \{0,1\}^n \rightarrow \{0,1\}^m$ de algum número fixo n de bits de entrada para algum número fixo m de bits de saída.

Um circuito lógico acíclico no qual cada instância de porta lógica é utilizada no máximo uma vez é chamado de circuito combinacional. Os circuitos combinacionais clássicos são determinísticos, ou seja, para uma determinada entrada (seqüência de bits) o resultado é sempre o mesmo. Considerando valores armazenados como fazendo parte da entrada, qualquer cálculo computacional pode ser realizado através de circuitos combinacionais clássicos. Uma ressalva a se fazer é que existem os circuitos combinacionais clássicos probabilísticos. Nestes, a execução depende de uma ou mais variáveis aleatórias, cujos valores variam aleatoriamente de acordo com alguma distribuição de probabilidade. Estes circuitos são determinísticos, se consideramos estas variáveis aleatórias como parte da entrada.

As principais portas lógicas conhecidas são: NOT, AND, OR e XOR, com as respectivas representações: $\neg, \wedge, \vee$ e $\oplus$. Todas elas possuem apenas um único bit de saída. A porta NOT, que também pode ser representada por um traço em cima da variável, possui apenas um bit de entrada e o inverte. As outras portas possuem dois bits de entrada. A porta AND retorna 1 se, e somente se, ambas entradas forem 1. A porta OR retorna 0, se e somente

se, ambas entradas forem 0. A porta XOR retorna 0, se e somente, se ambas entradas forem iguais. As portas AND e OR podem ser generalizadas para mais de duas entradas. O resultado da porta AND generalizada é igual a 1 se todas as entradas forem iguais a 1 e será 0 caso contrário. E a porta OR generalizada retorna 0 se todos os bits que compõe sua entrada forem iguais a 0, caso contrário, o resultado será 1.

Há ainda duas portas especiais: FANOUT, que possui uma única entrada e duas saídas que são cópias independentes da entrada, e CROSSOVER, que possui duas entradas e duas saídas cujos valores são os dois bits de entrada na ordem inversa. Nos circuitos irreversíveis, o FANOUT é representado por uma bifurcação na linha entre portas consecutivas. O CROSSOVER é representado apenas pelo cruzamento das linhas.

Há uma enorme quantidade de possíveis portas lógicas para um número fixo de entradas e saídas. Considerando que cada função com m bits de saídas pode ser substituída por m funções com um bit de saída, há 2^{m2^n} portas lógicas diferentes com n bits de entrada e m bits de saída. Todas estas portas lógicas podem ser construídas utilizando apenas portas AND e NOT, além das cópias. Isto também pode ser feito com outros conjuntos finitos de portas, como por exemplo, OR e XOR. Estes conjuntos de portas, que servem para implementar qualquer função, são chamados de conjuntos universais.

Uma forma de descrever a função que se deseja computar é através da tabela-verdade, na qual se enumeram todas as possíveis entradas e, para cada uma delas, é indicado o valor da saída. Existe um procedimento geral que indica como construir um circuito, a partir da tabela-verdade, usando portas AND, OR e NOT. Este procedimento consiste em montar um circuito para cada bit de saída. A entrada deste circuito é a entrada da função. Para cada linha da tabela-verdade cuja saída seja 1, coloca-se uma porta AND. Esta porta é precedida pela porta NOT em todas suas entradas cujo valor seja 0.

As saídas de todas estas portas AND formam a entrada de uma porta OR que retorna o resultado para aquele bit e saída da função. Para n entradas, a tabela-verdade possui 2^n linhas. Considerando que cada linha com uma das saídas com o valor 1 necessita de n portas AND e até n portas NOT, o procedimento padrão para construir o circuito a partir desta tabela pode gerar um circuito exponencial, necessitando de até $n2^n$ portas AND e NOT e uma porta OR generalizada com até 2^n entradas que, desmembrada em portas de duas entradas, é formada por 2^n portas OR.

Para mostrar como a construção, a partir da tabela-verdade, pode ser realizada para um exemplo concreto, vejamos o caso do somador completo (*full adder*). O somador completo é um circuito cuja finalidade é somar três bits. Este circuito possui duas saídas de acordo com as combinações dadas na tabela 2.1.

entradas			saídas	
a	b	c_e	s	c_s
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 2.1: Tabela-verdade para o somador completo. Os bits a e b se referem aos dígitos binários das entradas, c_e é o bit passado pela soma dos bits anteriores, c_s é o bit que será passado para frente e s é o dígito binário atual do resultado da soma.

Na figura 2.1, são mostrados o circuito gerado automaticamente a partir da tabela-verdade e um circuito otimizado que executa a mesma função.

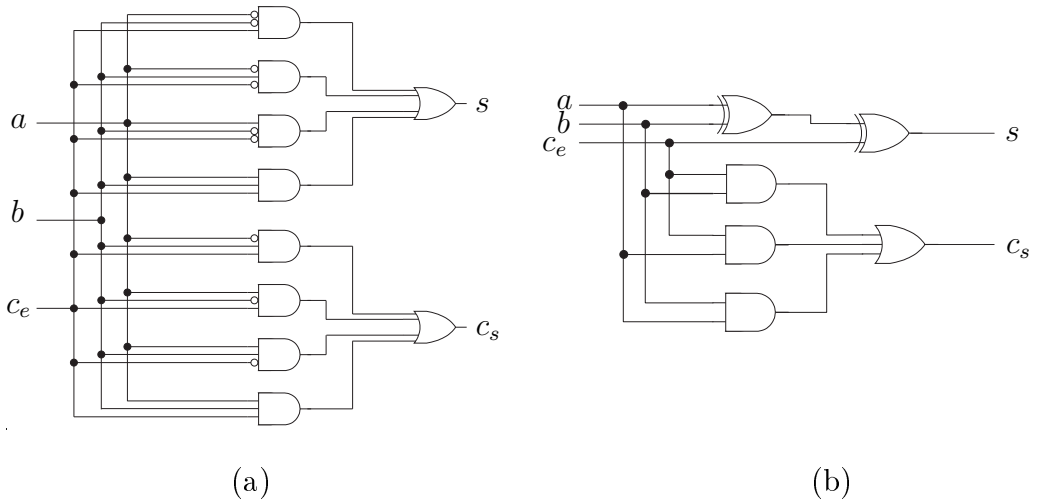

Figura 2.1: (a) Somador Completo construído a partir da tabela-verdade e (b) Somador Completo mais otimizado. Mesmo substituindo as portas XOR por AND's, OR's e NOT's, o circuito (b) possui menos portas no total.

2.2 Circuitos Reversíveis

Como foi dito anteriormente, cada circuito pode ser associado a uma função $f : \{0,1\}^n \rightarrow \{0,1\}^m$, onde n e m são as quantidades de bits necessários para representar a entrada e a saída, respectivamente. Se $n = m$ e a função f é bijetora, ela possui inversa, e conseqüentemente, existe um circuito que, para cada valor de saída y de f, produz o valor x tal que $f(x) = y$. Neste caso, dizemos que o circuito é reversível. Um circuito ou uma porta lógica que implementa uma função não inversível (não bijetora) é dito irreversível. Assim, todas as portas lógicas que possuam mais bits de entrada do que de saída são irreversíveis. Isto acontece, por exemplo, com as portas AND, OR e XOR, entre outras, que transformam duas ou mais entradas em uma única saída.

Na teoria, os circuitos reversíveis podem ser implementados através de um sistema físico adiabático, ou seja, um sistema conservativo, não há perda de calor. Na prática, há gasto de energia, mas esta é consideravelmente menor do que nos circuitos convencionais, nos quais, como mostrou Rolf Landauer [24], apagar irreversivelmente um bit gasta no mínimo $kT \ln 2$, onde k é a constante de Boltzmann e T é a temperatura. Por isso, os computadores reversíveis podem ser economicamente mais interessantes do que um computador convencional equivalente pela economia de energia.

Mas a principal importância dos circuitos reversíveis é o fato de serem base para os circuitos quânticos, que serão explicados mais adiante.

2.2.1 Portas Reversíveis

Das portas clássicas convencionais, desconsiderando a porta trivial (função identidade), apenas a porta NOT é reversível. Neste modelo, ela é representada por $\boxed{\text{X}}$.

Com alguns ajustes, portas irreversíveis podem ser transformadas em reversíveis [3, 26]. Por exemplo, se tomarmos a porta XOR, incluindo na saída, o valor de um dos bits de entrada, tem-se:

$$\begin{array}{rccc} f: & \{0,1\}^2 & \rightarrow & \{0,1\}^2 \\ & (x_1, x_2) & \mapsto & (x_1, x_1 \oplus x_2). \end{array} \tag{2.1}$$

Teoricamente, poderíamos montar uma porta XOR reversível, utilizando uma porta XOR padrão e copiando um dos bits. Mas para o circuito ser realmente reversível, ele deve utilizar apenas portas reversíveis e como acabamos de dizer, o XOR padrão não é reversível. Além do mais, também não se pode copiar o bit usando o FANOUT, pois esta porta também não é reversível.

Para diferenciar o modelo que permite o uso de portas irreversíveis do modelo que não o permite, denominamos o primeiro como circuitos convencionais e o segundo como circuitos reversíveis.

Toda porta lógica convencional pode ser implementada reversivelmente usando uma quantidade constante de portas reversíveis e alguns bits extras com valores pré-definidos.

A versão reversível da porta XOR, definida pela relação 2.1, chama-se CNOT (*CONTROLED NOT*). Esta operação pode ser vista como sendo uma aplicação do operador NOT no segundo bit, caso o valor do primeiro seja 1. O primeiro bit permanece inalterado, enquanto o segundo bit passa a ter o resultado da operação XOR. O primeiro bit é chamado de bit de controle e o segundo, bit alvo (*target*). A representação gráfica desta porta é dada na figura 2.2.

A porta CNOT pode simular a porta FANOUT. Dado um bit a, se utilizarmos este bit como bit de controle da porta CNOT e um bit com valor 0 como alvo, ou seja, entrada: $(a, 0)$, a saída é (a, a).

Para implementar a porta AND de forma reversível, é necessário mais um bit, pois a função original não é balanceada, ou seja, não gera igual quantidade de bits 0 e 1. Isto pode ser feito utilizando a porta Toffoli, cujo funcionamento é dado pela relação

$$\begin{array}{rccl} f: & \{0,1\}^3 & \to & \{0,1\}^3 \\ & (x_1, x_2, x_3) & \mapsto & (x_1, x_2, (x_1 \wedge x_2) \oplus x_3), \end{array} \tag{2.2}$$

fixando o último bit em 0. A representação usual das portas CNOT e Toffoli é dada na figura 2.2.

Como no caso dos circuitos convencionais, no modelo reversível, há conjuntos universais de portas lógicas, que permitem a construção de quaisquer outras portas ou circuitos, conforme o lema a seguir.

Proposição 1. *A porta Toffoli é universal para os circuitos reversíveis clássicos.*

Demonstração. Considerando que a porta Toffoli de 3 bits pode simular a porta AND, conforme a relação (2.2), e é mais geral do que a porta CNOT,

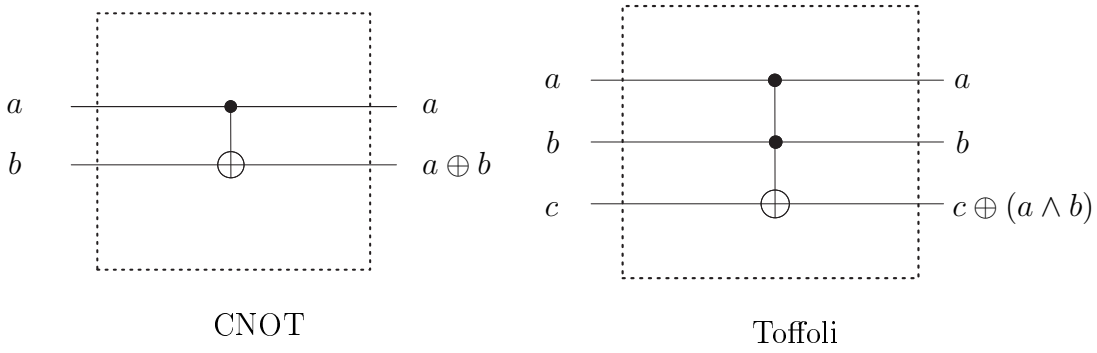

Figura 2.2: Representação usual das portas CNOT e Toffoli. Os bits superiores são de controle e o bit inferior é o alvo.

que simula as portas NOT e FANOUT, podemos afirmar que a porta Toffoli de 3 bits é universal, pelo fato de que as portas AND, NOT e FANOUT formam um conjunto universal no modelo convencional. Por outro lado, como a porta Toffoli pode ser simulada por um número constante de portas AND, NOT e FANOUT, podemos afirmar que ambos os modelos são equivalentes em termos de capacidade computacional.

Outra forma de mostrar que a porta Toffoli é universal para o modelo reversível é através da teoria de grupo de permutações [37]. Considere uma porta reversível genérica de n bits. Há 2^n possíveis entradas que estão no conjunto $\{0, ..., 2^{n-1}\}$ na notação decimal. A saída (imagem da função reversível correspondente) é uma permutação deste conjunto. Qualquer porta reversível de n bits pode ser representada por uma permutação em S_n, onde S_n é o grupo Simétrico de permutações de n elementos. A recíproca também é verdadeira, ou seja, qualquer permutação em S_n corresponde a uma porta reversível. A primeira parte da demonstração é a construção de uma porta Toffoli generalizada utilizando a porta Toffoli usual de 3 bits, introduzindo um espaço adicional. A construção da porta Toffoli de 4 bits a partir da porta de 3 bits pode ser feita conforme figura 2.3. Além dos três bits de controle a, b e c e do bit alvo d, nesta construção, foi utilizado um bit auxiliar com o valor inicial zero que armazena o valor intermediário $a \wedge b$, devido à primeira

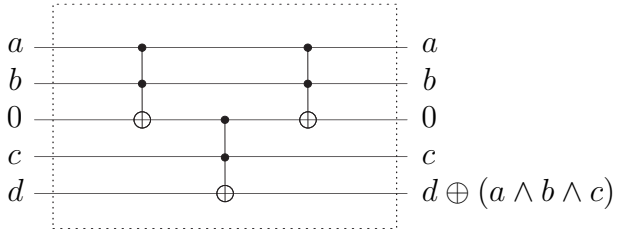

Figura 2.3: Construção da porta Toffoli de 4 bits (3 de controle e um alvo).

porta Toffoli de 3 bits. A segunda porta faz com que o último bit tenha o resultado $d \oplus (a \wedge b \wedge c)$. A terceira porta é apenas para limpar o conteúdo do bit auxiliar.

Da mesma forma como foi construída a porta Toffoli de 4 bits a partir da porta de 3 bits, é possível construir uma porta Toffoli de n bits a partir de portas de $n-1$ bits. A construção desta porta é uma generalização da descrição da figura 2.3, conforme pode ser observado na figura 2.4.

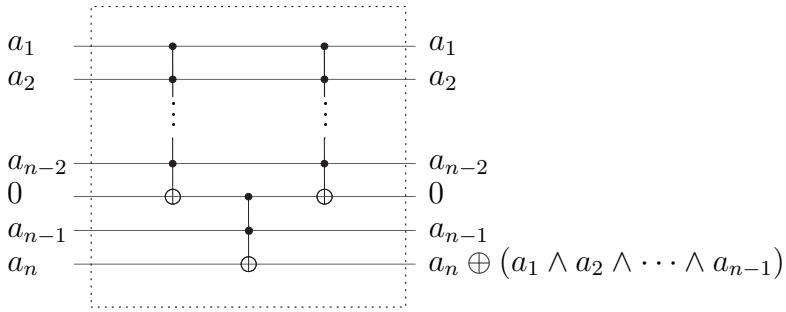

Figura 2.4: Construção da porta Toffoli generalizada de n bits a partir da Toffoli de $n-1$ bits.

Cada Toffoli generalizada de n bits pode ser montada com 2 portas Toffoli de $n-1$ bits e mais uma porta Toffoli de 3 bits. A quantidade de portas Toffoli de 3 bits utilizadas neste circuito é definida pela seguinte relação de recorrência:

$$T(n) = \begin{cases} 1, & \text{se } n = 3 \\ 2T(n-1) + 1, & \text{se } n > 3. \end{cases}$$

Isto significa que a quantidade de portas Toffoli utilizadas é exponencial:

$T(n) = 2^{n-2} - 1$. Este procedimento necessita de apenas um bit extra, pois em cada vez que há necessidade de utilizar o bit, pode-se utilizar o bit extra utilizado anteriormente, pois ele está com o valor zero.

Felizmente, a construção da porta Toffoli generalizada pode ser feita com complexidade linear, conforme é mostrado na figura 2.5. Pela figura, pode-

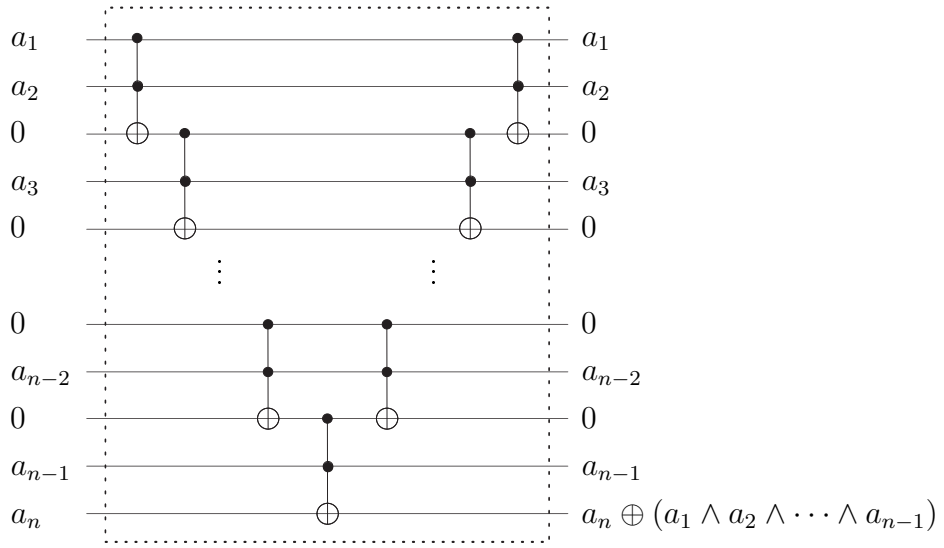

Figura 2.5: Construção da porta Toffoli generalizada com complexidade linear.

mos observar que este procedimento necessita de $2(n-3)+1$ portas Toffoli de 3 bits, ou seja $2n-5$ portas. O espaço também é linear: são necessários $n-3$ bits extras para montar uma porta Toffoli generalizada de n bits.

Usando portas X, que podem ser simuladas por portas Toffoli, é possível construir uma porta Toffoli generalizada cujos bits de controle são ativados por 0 em vez de 1, conforme mostrado na figura 2.6, lembrando que a porta X é a versão reversível da porta NOT. De posse desta nova porta, é possível transpor duas cadeias de n bits que diferem em apenas um bit. Isto pode ser feito colocando portas X antes e depois da porta Toffoli generalizada e usando como bit alvo, o bit que difere em ambas cadeias. Por exemplo, suponha que estejamos trabalhando com 4 bits, e queremos transpor as cadeias 0110 e 0111, ou seja, 6 e 7 em notação decimal. Isto significa que a saída do circuito

Figura 2.6: Porta Toffoli controlada negativamente, ou seja, por 0's em vez de 1's. Também neste caso, os bits de controle mantém seus valores originais.

desejado é

$$f(x) = \begin{cases} x, & \text{se } x \notin \{6,7\} \\ 6, & \text{se } x = 7 \\ 7, & \text{se } x = 6. \end{cases}$$

Este circuito consiste em uma porta Toffoli de 4 bits com o segundo e terceiro bits precedidos e sucedidos por uma porta X controlando o quarto bit, o que equivale a dizer que é uma porta Toffoli controlada positivamente pelo primeiro bit e negativamente pelos segundo e terceiro bits.

A transposição de duas cadeias de n bits, que diferem em s bits pode ser feita através de s transposições que permutam cadeias que diferem em apenas um bit. Toda permutação é um produto de transposições, logo toda função inversível sobre n bits é um produto de portas Toffoli de 3 bits e portas X. □

Se as portas AND, NOT e FANOUT podem ser simuladas pela porta Toffoli no modelo reversível, com certeza a porta OR também pode. Seja $c = a \vee b$ o resultado da operação OR entre a e b. Pelas propriedades de lógica booleana tem-se que $c = \overline{\overline{c}}$, onde $\bar{c}$ significa NOT c. Isto significa que $c = \overline{\overline{a \vee b}}$. Pela lei de De Morgan, $c = \overline{\overline{a} \wedge \overline{b}}$. Esta expressão é equivalente a $c = 1 \oplus (\overline{a} \wedge \overline{b})$. Isto implica em que a porta irreversível OR pode ser implementada no modelo reversível, utilizando também a porta Toffoli, colocando uma porta X em cada bit de controle, antes e depois da Toffoli, e fixando o bit alvo inicialmente em 1.

Na figura 2.7, é mostrado o circuito reversível equivalente ao circuito da figura 2.1, através da substituição direta das portas AND e OR por Toffolis e das portas XOR e Fanout por CNOTs.

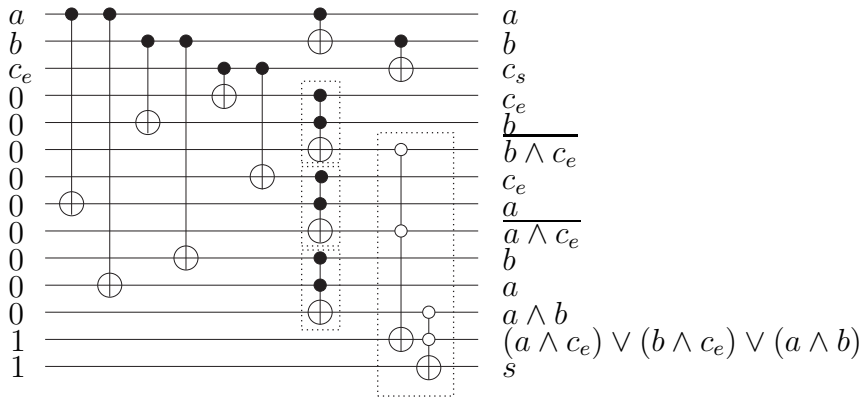

Figura 2.7: Somador Completo Reversível construído a partir do somador completo otimizado convencional da figura 2.1 (b). Os bits com as saídas são o terceiro (c_s) e o último (s).

A versão do somador completo reversível, descrita na figura 2.7, possui 11 bits extras além dos bits que armazenam as entradas e as saídas. A conversão direta do circuito convencional para o circuito reversível não garante bons circuitos, mesmo que seja a partir do melhor circuito convencional. Isto acontece por pelo menos dois motivos: no modelo convencional, as portas FANOUT são ignoradas e no modelo reversível não, além disso, no primeiro modelo, os bits podem ser descartados sem nenhum custo extra, enquanto que no segundo modelo, esta limpeza envolve um custo adicional. No caso da construção de um somador completo reversível, vemos isso claramente. Na figura 2.8 é mostrado um circuito equivalente ao da figura 2.7, porém, mais otimizado. Na tabela 2.2.1 é mostrado o valor de cada bit após a aplicação de cada porta.

Ao se projetar um circuito reversível, deve-se sempre levar em conta o que se deseja otimizar: quantidade de portas usadas ou quantidade de bits extras. Sempre é possível construir um circuito, com no máximo 1 bit extra além

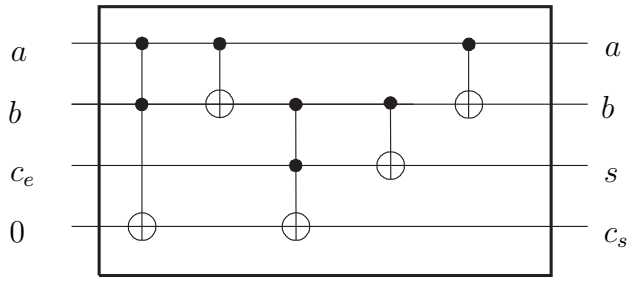

Figura 2.8: Somador Completo Reversível Otimizado

bits			
a	b	c_e	0
a	b	c_e	$a \wedge b$
a	$a \oplus b$	c_e	$a \wedge b$
a	$a \oplus b$	c_e	$a \wedge b \oplus (c_e \wedge (a \oplus b))$
a	$a \oplus b$	$s \equiv a \oplus b \oplus c_e$	$c_s \equiv (a \wedge b) \oplus (a \wedge c_e) \oplus (b \wedge c_e)$
a	b	$s \equiv a \oplus b \oplus c_e$	$c_s \equiv (a \wedge b) \oplus (a \wedge c_e) \oplus (b \wedge c_e)$

Tabela 2.2: Evolução dos bits do somador descrito na figura 2.8

dos bits de entrada e saída, utilizando o procedimento descrito na seção 6.1.3 de [37]. Mas este procedimento pode levar a um aumento considerável de portas adicionais, podendo ser necessárias $\Theta(2^n)$ portas [39]. Por todas estas considerações, a construção de circuitos reversíveis eficientes é uma arte.

2.2.2 Análise de complexidade

A análise do uso de recursos (complexidade de tempo e espaço) para os circuitos reversíveis pode ser feita de forma similar ao modelo convencional. A complexidade de tempo pode ser dada pelo número de passos necessários para a execução completa do circuito. Por simplicidade, considera-se que cada porta elementar é executada em um único passo e não há intervalo de tempo entre o fim da execução de uma porta e o início da seguinte. Portas

aplicadas a bits distintos podem ser executadas em paralelo. Desta forma, a complexidade de tempo pode ser mensurada pela maior seqüência de portas que envolvem um mesmo bit [37, cap. 6]. Esta medida é chamada de profundidade (*depth*), enquanto a quantidade total de portas é chamada de tamanho do circuito (*size*).

A complexidade de espaço é mais simples de se avaliar do que no modelo convencional, pois a quantidade de bits permanece constante durante todo o procedimento, visto que o número de bits de saída em cada porta reversível é igual ao número de bits de entrada nesta porta.

Suponha que tenhamos um circuito convencional utilizando apenas portas AND, OR e NOT, lembrando que todo circuito convencional pode ser construído desta forma. Como foi visto antes, a porta NOT é reversível e as portas AND e OR podem ser substituídas por Toffolis. Mas cada substituição desta necessita de um bit extra, que inicialmente possui um valor fixo e depois guarda o resultado da operação (no caso AND e OR). Como cada porta convencional pode ser substituída por um número constante de portas reversíveis, a ordem de complexidade de portas do circuito reversível será a mesma do circuito convencional equivalente, multiplicado por uma constante. Mas em relação ao espaço, como cada substituição de portas pode necessitar de um bit extra, a complexidade de espaço pode chegar a ser da mesma ordem de grandeza da complexidade de tempo, caso estes bits extras não sejam reaproveitados.

2.2.3 Limpeza do lixo

No modelo reversível, devido ao fato de não ser possível descartar o valor de um bit, não é tão simples, como no modelo convencional, atribuir um novo valor para um bit desconsiderando o valor anterior.

Charles Bennett em 1973 [3], propôs um procedimento genérico reversível

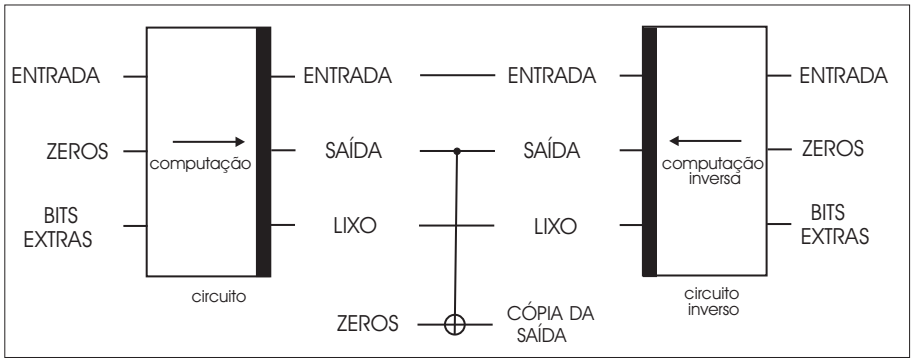

Figura 2.9: Procedimento geral para limpar o valor dos bits extras

para limpar o valor dos bits extras de cada circuito (ou porta) reversível. Trata-se de executar o circuito na ordem direta, copiar os bits com o resultado da função para novos bits e executar novamente o circuito, mas na ordem inversa, conforme pode ser visto na figura 2.9.

Caso o procedimento seja aplicado apenas no circuito como um todo, para limpar todos os bits extras, este método possui complexidade de tempo $\Theta(T)$ e espaço $\Theta(T+S)$, onde $\Theta(T)$ e $\Theta(S)$ são as complexidades de tempo e de espaço respectivamente do circuito convencional. Então, um circuito convencional que tivesse complexidade de tempo exponencial e de espaço polinomial, no caso reversível teria a mesma complexidade de tempo, mas seu espaço seria exponencial.

Bennett, em 1989 [4], propôs então o uso deste método em etapas intermediárias do circuito, e assim, conseguiria uma complexidade de tempo $O(T^{1+\epsilon})$ e de espaço $O(S \log T)$, dado um $\epsilon > 0$. Levine e Sherman [27] mostraram, no ano seguinte, que há um fator escondido na ordem de complexidade de espaço, da forma $\epsilon 2^{1/\epsilon}$, que diverge quando ϵ se aproxima de 0. Usando equações de recorrência, eles provaram que o novo método do Bennett possui complexidade de tempo $\Theta(T^{1+\epsilon}/S^{\epsilon})$ e de espaço $\Theta(S(1+\log T/S))$.

O segundo método do Bennett, que ele chama de jogo dos seixos, funciona da seguinte forma: divide-se o circuito em m blocos de comprimento S. Se

forem executados n blocos consecutivos do algoritmo, então o espaço usado é $O(nS)$. Se em seguida forem copiados os bits, com o resultado parcial, e executados os mesmos passos para trás, os bits extras estarão zerados para serem usados nos nS passos seguintes do circuito. Repetindo este procedimento de executar alguns passos para frente, copiar o resultado parcial e executar alguns passos para trás, é possível ir mantendo o espaço extra em um determinado tamanho. O maior problema é a necessidade de guardar o valor dos bits extras e do resultado da saída do bloco anterior para executar na ordem direta o bloco seguinte do circuito, e guardar os valores de saída de cada bloco para executá-lo de forma reversa. Alguns blocos de referência devem ser mantidos para proceder os passos reversos para eliminação dos valores indesejáveis. Depois de executar outro conjunto de blocos, para remover o lixo do último bloco de referência, é necessário reexecutar o circuito desde o início até aquele ponto. Li, Tromp e Vitányi, em [28], definem o jogo dos seixos da seguinte forma: seja G uma lista linear de nós rotulados de 1 a T_G. Há um único jogador, que possui n peças iguais que podem ser colocadas ou retiradas dos nós em G de acordo com a seguinte regra: caso o nó i esteja ocupado com uma peça, se o nó seguinte está livre, então pode-se colocar uma peça neste nó, caso contrário pode-se tirar a peça deste nó. Ou seja, pode-se tirar ou colocar peças, apenas se o nó anterior estiver com outra peça. O objetivo é colocar uma peça no último nó, com o menor número de movimentos. Para começar o jogo, assume-se que sempre se pode colocar uma peça no primeiro nó. A relação do jogo com a limpeza de lixo é feita da seguinte forma: dado um procedimento irreversível com T passos, estes passos são divididos em segmentos iguais de T_G passos. Para todos os efeitos, podem ser considerados apenas os passos irreversíveis. A porta NOT, por exemplo, não produz lixo, assim como outras portas, em que a saída é armazenada no lugar de alguma entrada. Cada peça é uma unidade de memória. Como temos n peças, podem ser gastas até n unidades de memória.

A peça no i-ésimo nó guarda o resultado do i-ésimo passo deste trecho do programa. Colocar uma peça em um determinado nó significa executar, na ordem direta, este passo. Retirar uma peça significa executar este passo no sentido reverso. Tanto para executar um passo no sentido direto como no reverso, é necessário ter o resultado do passo anterior, por isso a exigência de haver uma peça na casa anterior.

2.3 Circuitos Quânticos

2.3.1 Bits Quânticos

Em um computador clássico (circuito clássico), em cada instante de tempo, um bit ou está no estado 0 ou no estado 1. Já em um circuito quântico, o estado de cada bit é uma superposição dos estados 0 (denotado por $|0\rangle$) e 1 (denotado por $|1\rangle$). Do ponto de vista matemático, um bit quântico, também chamado de q-bit, é um vetor normalizado (módulo unitário) de um espaço vetorial bidimensional sobre os complexos (espaço de Hilbert), no qual os estados os estados $|0\rangle$ e $|1\rangle$ formam uma base ortonormal, chamada de base computacional. Logo, o q-bit pode ser escrito como uma combinação linear destes estados: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, onde $\alpha, \beta \in \mathbb{C}$. Se $\alpha, \beta \neq 0$, então diz-se que o estado $|\psi\rangle$ é uma superposição dos estados clássicos 0 e 1.

O par ordenado (α, β) define o estado de um q-bit, normalmente representado na forma matricial:

$$|\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \alpha + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \beta$$

Se α e β forem reais, o q-bit pode ser representado em um círculo trigonométrico, conforme a figura 2.10. No caso geral, o q-bit pode ser representado em uma esfera de raio unitário, chamada esfera de Bloch [34, seção 1.2].

A medição do estado de um q-bit equivale a uma projeção do mesmo

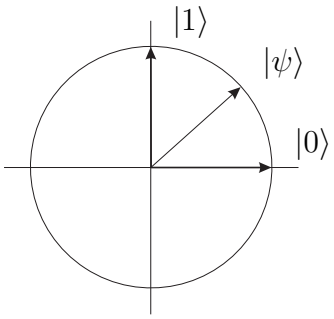

Figura 2.10: Representação do q-bit com coeficientes reais

em um dos elementos da base. Por isso, apesar de o q-bit poder estar numa superposição de estados clássicos, ao ser observado, ele passa a ter um estado clássico. $|\alpha|^2$ e $|\beta|^2$ são as probabilidades de serem medidos os valores 0 e 1 respectivamente. Por isso, de certa forma, pode se dizer que a entrada e a saída de um computador quântico são clássicas.

Múltiplos q-bits

O estado formado pelo conjunto de vários q-bits não é definido apenas pela concatenação dos estados individuais de cada um deles, como acontece em um sistema clássico, mas sim pelo produto tensorial ou produto de Kronecker destes estados (cuja notação é $\otimes$). A definição e propriedades do produto tensorial podem ser encontradas em alguns livros de Álgebra Linear ou outros textos que explicam as operações em espaços vetoriais como [34]. A dimensão do espaço vetorial que contém um estado formado por n q-bits é 2^n.

Os estados $|0\rangle \otimes |0\rangle$, $|0\rangle \otimes |1\rangle$,$|1\rangle \otimes |0\rangle$ e $|1\rangle \otimes |1\rangle$ podem ser escritos como $|00\rangle$, $|01\rangle$, $|10\rangle$ e $|11\rangle$ ou também na forma decimal: $|0\rangle$, $|1\rangle$, $|2\rangle$ e $|3\rangle$. Algumas vezes será indicado explicitamente a quantidade de q-bits que compõe um estado, isto será feito através de um índice após a representação do estado. Por exemplo, o estado $|1\rangle \otimes |1\rangle \otimes |0\rangle = |6\rangle_3$.

A seguir, é mostrado o estado do sistema formado por dois q-bits $|\psi_1\rangle =$

$\begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix}$ e $|\psi_2\rangle = \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix}$:

$$|\phi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = \begin{bmatrix} \alpha_1 \\ \beta_1 \end{bmatrix} \otimes \begin{bmatrix} \alpha_2 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} \alpha_1\alpha_2 \\ \alpha_1\beta_2 \\ \beta_1\alpha_2 \\ \beta_1\beta_2 \end{bmatrix}$$

$$= \alpha_1\alpha_2|0\rangle + \alpha_1\beta_2|1\rangle + \beta_1\alpha_2|2\rangle + \beta_1\beta_2|3\rangle.$$

Pelo resultado acima, pode-se verificar que o estado do sistema formado por dois q-bits é uma combinação linear dos quatro arranjos clássicos possíveis de dois valores binários.

Estes estados formam uma base ortonormal de um espaço vetorial quadridimensional.

O estado de qualquer sistema de dois q-bits pode ser escrito então como: $|\psi\rangle = a_0|0\rangle + a_1|1\rangle + a_2|2\rangle + a_3|3\rangle$, onde $a_0 = \alpha_1\alpha_2, a_1 = \alpha_1\beta_2, a_2 = \beta_1\alpha_2$ e $a_3 = \beta_1\beta_2$.

O produto tensorial preserva a norma, ou seja, o vetor que define o estado do sistema possui módulo unitário ($|a_0|^2 + |a_1|^2 + |a_2|^2 + |a_3|^2 = 1$). Generalizando, um sistema com n q-bits possui um estado global $|\psi\rangle = \sum_{i=0}^{2^n-1} a_i|i\rangle$. Cada valor a_i é chamado de amplitude do estado $|i\rangle$.

Para simplificar mais a notação, o estado $|\phi\rangle = |\psi_1\rangle \otimes |\psi_2\rangle$ pode ser escrito também como $|\phi\rangle = |\psi_1\rangle|\psi_2\rangle$.

2.3.2 Operadores Quânticos

Devido aos postulados da Mecânica Quântica, o único tipo de transformação possível em um sistema quântico fechado é a transformação unitária (operação linear que preserva a norma do vetor). Uma operação linear U em um espaço de Hilbert V é dita unitária se a matriz M_U associada a U é

inversível e $M_U^{-1} = M_U^\dagger$, onde $M_U^\dagger$ representa a matriz transposta conjugada de M_U.

Como as matrizes referentes aos operadores unitários são inversíveis, isto significa que os operadores quânticos são reversíveis. Cada operador reversível clássico está associado a uma matriz identidade com colunas permutadas. Isto significa que os circuitos reversíveis clássicos formam um subconjunto dos circuitos quânticos.

Além das portas reversíveis vistas na seção 2.2, há outras portas importantes. Além da identidade

$$I \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

e da porta NOT, já vista,

$$X \equiv \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

há infinitas portas que podem ser aplicadas a um sistema formado por um único q-bit. Por exemplo, pode-se construir qualquer porta cuja matriz é da forma

$$U_\theta \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{bmatrix},$$

para $0 \leq \theta < 2\pi$. Quando $\theta = \frac{\pi}{4}$, esta porta é chamada, por uma razão histórica, de $\frac{\pi}{8}$. Isto porque

$$T \equiv \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix} = e^{i\frac{\pi}{8}} \begin{bmatrix} e^{-i\frac{\pi}{8}} & 0 \\ 0 & e^{i\frac{\pi}{8}} \end{bmatrix}.$$

Se $\theta = \pi$ tem-se a porta

$$Z \equiv \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}.$$

A porta Z mantém a amplitude do estado $|0\rangle$ e inverte a amplitude do estado $|1\rangle$; se $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ então $Z|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$.

Um operador bastante importante é a porta Hadamard:

$$H \equiv \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Com esta porta, um q-bit que estiver no estado $|0\rangle$, passa a estar no estado $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ e se estiver no estado $|1\rangle$, passa a estar no estado $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, ou seja, em ambos casos, passa de um estado projetado para um estado em superposição.

Ao contrário do que acontece com os circuitos clássicos, não existe um conjunto discreto de operadores quânticos universal, ou seja, que implemente perfeitamente qualquer operador quântico arbitrário, visto que o conjunto de operadores unitários é contínuo. Mas se fixarmos uma precisão para o resultado da operação (módulo da diferença entre os vetores resultantes da aplicação do operador que se deseja simular e o operador construído a partir de um conjunto discreto de portas ao mesmo estado) é possível definir um conjunto discreto de operadores que seja universal.

Um conjunto universal de operadores é formado pelas portas Hadamard, fase $S \equiv \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$, CNOT e pela porta $\frac{\pi}{8}$. Uma demonstração da universalidade deste conjunto e de outros pode ser encontrada em [34, sec. 4.5]. Interessante é o fato de que no caso de circuitos reversíveis clássicos, somente é possível montar um conjunto universal, se houver alguma porta de 3 bits, enquanto que o conjunto universal para circuitos quânticos, com uma determinada precisão, pode ser montado com uma porta de dois bits (CNOT) junto com outras portas de um único bit.

2.3.3 Paralelismo Quântico

Diferentemente dos circuitos clássicos, os circuitos quânticos podem ser executados com várias entradas simultâneas sem nenhuma mudança no circuito e nem aumento de complexidade de tempo e de espaço.

Dado um sistema de n bits em que todos eles estão no estado zero, tem-se $|\psi_0\rangle = |0\rangle \otimes |0\rangle \otimes \ldots \otimes |0\rangle$. Caso sejam aplicadas portas Hadamard em todos os bits, o estado global passa a ser $|\psi_1\rangle = H|0\rangle \otimes H|0\rangle \otimes \ldots \otimes H|0\rangle = \frac{1}{\sqrt{2^n}}(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \ldots \otimes (|0\rangle + |1\rangle)$, fazendo o produto tensorial destes estados, tem-se que $|\psi_1\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle$. Pode se dizer que o estado global do sistema passa a ser uma superposição de todos os estados possíveis da base computacional para n bits.

Seja $f : \{0,1\}^n \rightarrow \{0,1\}^m$ a função associada a um circuito clássico irreversível, onde n é o tamanho da entrada e m o tamanho da saída. É possível definir um circuito clássico reversível $U_f : (x, y) \rightarrow (x, y \oplus f(x))$. Como as portas reversíveis clássicas formam um subconjunto das portas quânticas, este mesmo circuito pode ser projetado para um computador quântico, sendo que, se a entrada for clássica (estado projetado), a saída será igual a do circuito reversível clássico. Ou seja, $\forall i \in [0, 2^n - 1], U_f|i\rangle|0\rangle = |i\rangle|f(i)\rangle$. Suponha que tenhamos um estado $|\psi\rangle = \sum_{i=0}^{2^n-1} \alpha_i|i\rangle$ com $\alpha_i \in \mathbb{C}$ e $\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1$. O operador U_f aplicado a este estado é igual a

$$U_f|\psi\rangle|0\rangle = U_f \sum_{i=0}^{2^n-1} \alpha_i|i\rangle|0\rangle.$$

Como U_f é um operador linear, ele pode ser aplicado a cada termo do somatório. Logo,

$$U_f|\psi\rangle|0\rangle = \sum_{i=0}^{2^n-1} \alpha_i U_f|i\rangle|0\rangle.$$

Isto significa que

$$U_f|\psi\rangle|0\rangle = \sum_{i=0}^{2^n-1} \alpha_i|i\rangle|f(i)\rangle.$$

Isto significa que executando apenas uma única vez um operador, ele calcula o resultado para todos os estados que estão em superposição. O problema que surge é o fato de a saída também ser superposta. Apenas um dos estados que compõe a base será medido no final, perdendo a informação sobre todos os outros estados.

Capítulo 3

Algoritmos Aritméticos Reversíveis

Assim que Peter Shor criou o algoritmo quântico para fatoração de inteiros, em 1994 [40], começaram a surgir vários projetos de circuitos para a exponenciação modular. Esta operação é o procedimento que possui maior custo computacional neste algoritmo. A exponenciação no algoritmo de Shor não é usada com superposição de valores na base, mas apenas no expoente; sendo assim, é suficiente para resolver o problema da fatoração, algoritmo de multiplicação modular no qual um dos operandos seja fixo, como acontece nos algoritmos descritos em [2, 20, 46, 47]. Mas além de outras aplicações para a computação quântica, está crescendo o interesse pela computação reversível clássica principalmente devido à nanotecnologia. Consequentemente, está surgindo a necessidade de operações aritméticas reversíveis eficientes não apenas como base do algoritmo de Shor, mas para outros algoritmos que possam necessitar que ambos operandos sejam variáveis (não fixos).

As operações aritméticas básicas com inteiros, como soma, subtração, multiplicação e divisão, são fundamentais para muitos algoritmos. Em geral, o custo destas operações é considerado unitário, pois podem ser realizadas com uma quantidade constante de passos na maior parte das aplicações. Em

um computador usual, estas operações estão previamente definidas no seu conjunto de instruções, bastando carregar os operandos para os registradores, aplicar a operação padrão e devolver o resultado no registrador de saída. Mas isto é verdade apenas quando os operandos possuem tamanho pequeno, pois, se as entradas forem maiores do que o comprimento da palavra do computador, estas operações devem ser realizadas por partes.

Neste capítulo, mostramos como implementar as principais operações aritméticas, com inteiros não-negativos, no modelo reversível (incluindo o clássico) e com a possibilidade de que ambos operandos estejam em superposição de valores, quando implementadas em um computador quântico. No caso da multiplicação, além do algoritmo padrão, será mostrado o algoritmo Karatsuba Revesível, que possui complexidade de circuito $\Theta(n^{\log 3})$, sendo melhor do que o algoritmo padrão, que é $\Theta(n^2)$, além disso, há uma versão que possui profundidade $\Theta(\log^2 n)$. Assintoticamente, há um algoritmo mais eficiente do que o Karatsuba para multiplicação, algoritmo de Schönhage-Strassen [15, 38], mas devido ao fator que multiplica o maior termo da complexidade de tempo, ele é eficiente apenas para operandos com mais do que 32000 bits [32]. Como a soma é a base da multiplicação, inicialmente, na seção 3.1, apresentamos alguns procedimentos reversíveis para a adição de inteiros encontrados na literatura. A subtração, conforme será visto, deriva diretamente da soma. Em seguida, na seção 3.2, é mostrada nossa proposta para multiplicação regular de inteiros. Na literatura não se encontra o algoritmo para multiplicação regular, mas apenas para a multiplicação modular, pois esta última é implementada a partir da soma modular. Um circuito reversível para divisão de inteiros, também está ausente nos principais artigos de computação reversível e quântica, é proposto na seção 3.3. Em seguida, na seção 3.4, são mostrados circuitos para implementar algumas operações modulares. Na seção 3.5, é apresentado o algoritmo Karatsuba clássico e nossas propostas para uma versão reversível para a multiplicação usando o

algoritmo Karatsuba, e um circuito para a multiplicação modular baseado no algoritmo Karatsuba reversível.

3.1 Soma reversível

Na figura 2.8 do capítulo anterior, mostramos como pode ser construído um somador completo (soma de dois operandos formados por um bit cada, contendo um bit extra para somar com o próximo dígito) de forma reversível. Este circuito somador pode ser usado em cascata para a adição de operandos com múltiplos bits. Na figura 3.1, é mostrado o exemplo para a soma de operandos com 3 bits, usando o somador completo reversível.

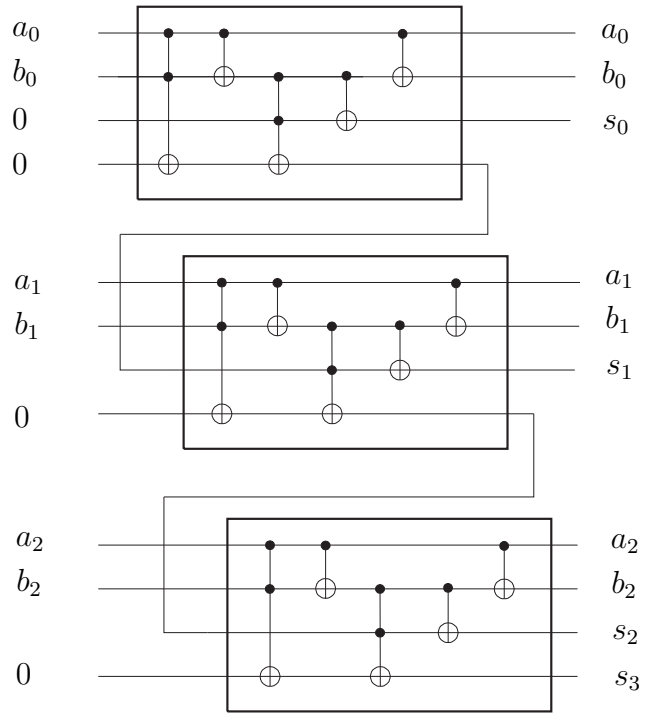

Figura 3.1: Somadores Reversíveis usados em série para soma de operandos a e b formados por 3 bits. O resultado é representado por $s = s_3 s_2 s_1 s_0$.

No circuito mostrado na figura 3.1, são usados 6 bits para os operandos de entrada e outros 4 bits para guardar o resultado da operação. No caso geral de operandos com n bits, são necessários $n + 1$ bits além dos $2n$ bits

das entradas. É possível construir um circuito somador no qual a saída é armazenada no lugar de um dos operandos; isto pode ser útil para economizar bits. Vedral, Barenco e Ekert mostram como implementar a soma reversível de dois operandos com n bits cada, com o resultado no segundo operando [46]. Esta implementação utiliza n bits extras que são zerados no final, conforme mostrado na figura 3.2. Este operador, chamado ADDER, executa a seguinte operação $(a, b) \rightarrow (a, a + b)$. Apesar de ambos algoritmos usarem aproximadamente $3n$ bits, a vantagem da versão de [46] é o fato de que os bits extras são zerados e podem ser reutilizados posteriormente. Como no caso do circuito da figura 3.1, a quantidade total de portas deste circuito é linear em relação ao número de bits dos operandos. Há outras implementações parecidas, e que utilizam pelo menos $3n$ bits, como é o caso de [16]. Beckman e outros em [2] propõem circuitos para soma que utilizam $3n + 1$ ou $3n + 3$, dependendo de qual das versões apresentadas é utilizada, incluindo a quantidade de bits necessária para armazenar um dos operandos que é fixo.

Para economizar espaço, Draper [10], em 2000, propôs um algoritmo para adição baseado na Transformada de Fourier Quântica [29] que utiliza apenas $2n + 1$ bits no total. O problema do algoritmo é a complexidade de tempo (tanto profundidade quanto tamanho do circuito): $\Theta(n \log n)$, na forma aproximada e $\Theta(n^2)$, na forma exata. Este algoritmo utiliza operações, cuja simulação em um modelo clássico é exponencial, sendo polinomial portanto apenas no modelo quântico.

Além da quantidade de bits extras e da quantidade total de portas do circuito, outra propriedade que se procura otimizar é o grau de paralelismo das operações, ou melhor, a maior seqüência de operações que não pode ser paralelizada, também chamada de *profundidade* do circuito, como comentado no capítulo anterior. Em 1998, Phill Gossett [16] propôs um algoritmo que possui profundidade $\Theta(\log n)$ para uma versão reversível, enquanto que os

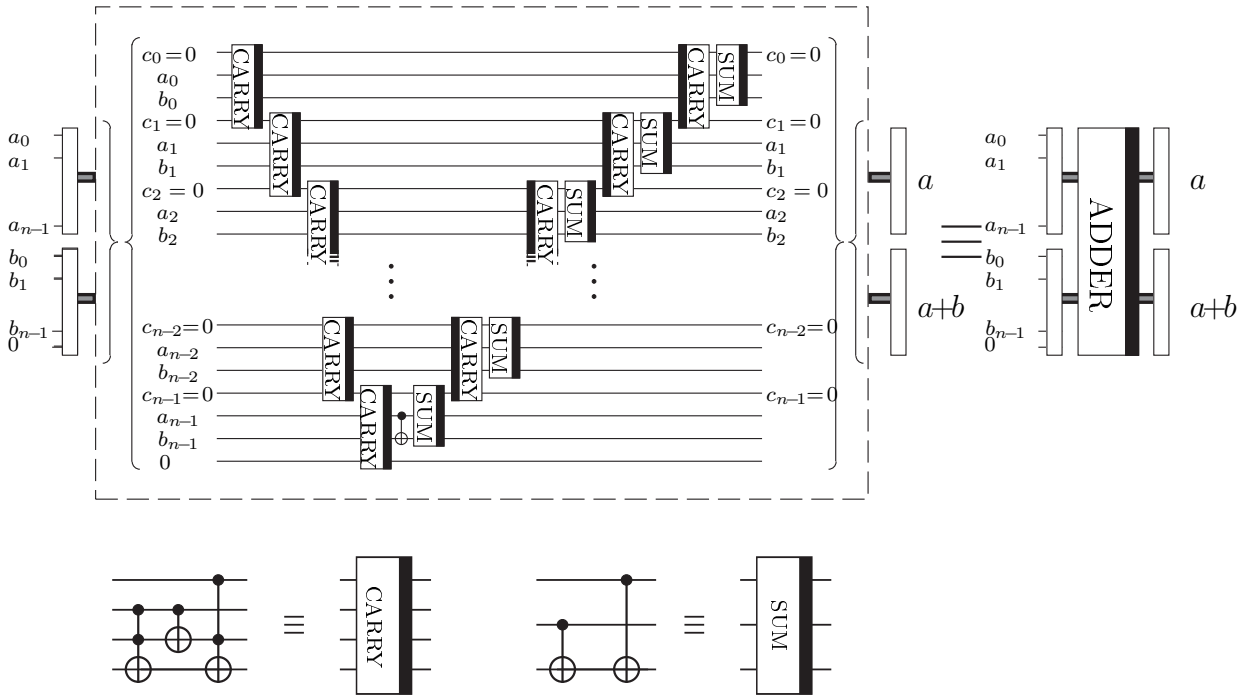

Figura 3.2: Circuito para soma reversível proposta por Vedral, Barenco e Ekert [46]. Na porta ADDER representada à direita, foi incluído mais um bit no segundo registrador com entrada igual a zero para armazenar o *carry* final da operação, que faltou na descrição original. Outra diferença desta figura para o diagrama original é que esta representa a soma de dois operandos de n bits cada, enquanto que a descrição original é para adição de operandos com $n + 1$ bits.

circuitos anteriormente apresentados para a soma ([10] e [46]) possuem profundidade $\Theta(n)$. O problema deste circuito é a necessidade de $\Theta(n^2)$ bits. Em 2004, o mesmo Draper junto com Kutin, Rains e Svore [11] fizeram a conversão de um somador convencional que possui profundidade $\Theta(\log n)$ para uma versão reversível, com uma complexidade de espaço linear em relação à quantidade de bits dos operandos.

Na tabela 3.1, é mostrada uma comparação entre as propriedades de alguns somadores. Uma análise comparativa da quantidade exata de portas e bits utilizados, entre estas e outras propostas de somadores reversíveis e quânticos, pode ser vista em [32].

referência	número de portas	profundidade de portas	espaço utilizado
[9]	$\Theta(n)$	$\Theta(n)$	$2n+1$
[10]	$\Theta(n \log n)$	$\Theta(n \log n)$	$2n+1$
[11]	$\Theta(n)$	$\Theta(\log n)$	$\Theta(n)$
[46]	$\Theta(n)$	$\Theta(n)$	$3n$

Tabela 3.1: Comparação entre alguns somadores reversíveis. A complexidade de portas de [10] é utilizando a versão aproximada da QFT. Se utilizar a forma exata da QFT, tanto o número de portas quanto a profundade passam a ser $\Theta(n^2)$.

A porta ADDER pode ser usada para fazer a subtração entre dois números inteiros, quando colocada de forma reversa no circuito. No artigo de Vedral, Barenco e Ekert, é dito erroneamente que, no sentido reverso, a porta ADDER faz o seguinte:

$$\begin{cases} (a,b) \rightarrow (a, a-b), & \text{se } a \geq b \\ (a,b) \rightarrow (a, 2^{n+1} - (b-a)), & \text{se } b > a. \end{cases}$$

Na realidade, a operação realizada é

$$\begin{cases} (a,b) \rightarrow (a, b-a), & \text{se } b \geq a \\ (a,b) \rightarrow (a, 2^{n+1} - (a-b)), & \text{se } a > b, \end{cases}$$

quando ambos operandos a e b possuem n bits.

3.1.1 Soma Reversível Controlada

Todas as operações reversíveis podem ser realizadas de forma controlada, ou seja, dado um operador $U : x \rightarrow y$, é possível construir um operador

$$C(U) : (c,x) \rightarrow \begin{cases} (c,y) & \text{se } c = 1 \\ (c,x) & \text{se } c = 0. \end{cases}$$

No caso de circuitos quânticos, isto pode ser feito com um novo registrador para guardar o bit de controle, acrescentando uma porta de deslocamento de

fase $\exp(i\alpha)$ e substituindo duas portas X por CNOT's, conforme explicado na seção 4.3 do livro [34]. No caso clássico, além do registrador extra para o bit de controle, um modo trivial de resolver o problema é substituindo todas as portas C^n-NOT, portas NOT controladas por n bits, por C^{n+1}-NOT. A implementação de uma porta C^{n+1}-NOT a partir de uma C^n-NOT, também pode ser vista no capítulo 4 de [34].

Concretamente para construir um somador controlado C-ADDER, a partir do somador da figura 3.2, é necessário um novo registrador com n bits inicialmente com valor zero. Cada novo bit será controlado, através de uma porta Toffoli, pelo novo bit de controle e o respectivo bit de uma das entradas conforme figura 3.3. São necessárias então $2n$ portas Toffoli adicionais.

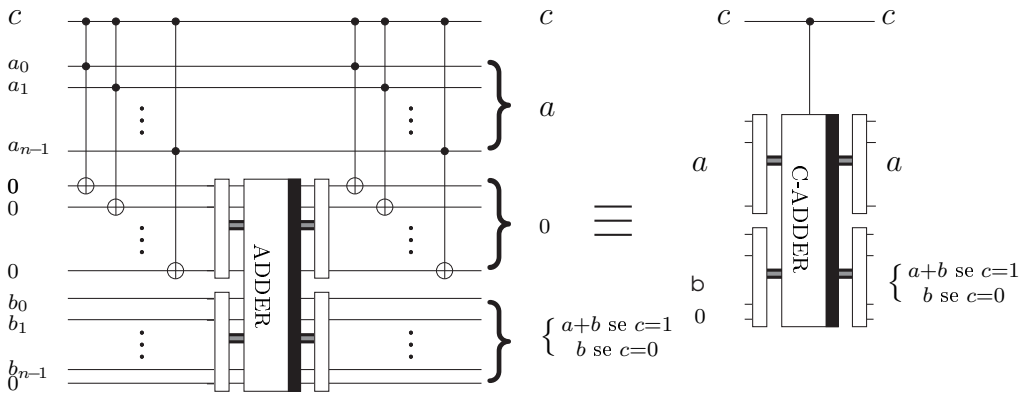

Figura 3.3: C-ADDER. Circuito para soma reversível controlada a partir da soma proposta por Vedral, Barenco e Ekert [46]. Todas as portas que aparecem antes do somador podem ser executadas simultaneamente, pois são aplicadas a bits distintos. O mesmo ocorre com as portas após o somador.

3.2 Algoritmo padrão para multiplicação reversível

Os textos [9, 10, 11] tratam apenas da operação de adição reversível. O artigo [46] faz apenas a multiplicação modular reversível. Em [20], há

implementação de um multiplicador de 2 bits. Florio e Picca [13] propõem um algoritmo para multiplicação reversível, mas este é exponencial em relação ao número de bits dos operandos, pois o que propõem é repetir a soma de a com o resultado acumulado b vezes, onde a e b são os operandos.

Uma maneira natural de fazer a multiplicação de números inteiros, com complexidade polinomial sobre o número de bits, é através de um processo parecido com o método para exponenciação modular. Sejam a e b os operandos com n bits cada, aplica-se adição sobre adição a partir do operando a e somam-se os resultados parciais de acordo com a decomposição binária de b, conforme apresentado no algoritmo 3.2.1.

Algoritmo 3.2.1 Algoritmo padrão para multiplicação

entradas: $a = a_{n-1} \cdots a_1 a_0$ e $b = b_{n-1} \cdots b_1 b_0$

$c \leftarrow a$

$mult \leftarrow 0$

para $i \leftarrow 0$ até $n-1$ **faça**

  se $b_i = 1$ **então**

    $mult \leftarrow mult + c$

  fim se

  $c \leftarrow c + c$

fim para

devolva $mult$

Se os operandos possuem n bits cada, podem ser necessários até $2n$ bits para guardar o resultado, sendo assim, o resultado da operação não pode ser armazenado completamente no lugar de um dos operandos como ocorre com a soma. Mas é possível, e será mostrado adiante, como construir o circuito com o resultado substituindo um dos operandos utilizando outros bits adicionais. A quantidade de bits extras usados em um circuito para multiplicação depende da forma como é feita a limpeza do lixo. Se não for

utilizado nenhum procedimento para restaurar o valor inicial dos bits extras, são necessários $\Theta(n^2)$ bits, que coincide com a complexidade de tempo do algoritmo.

O algoritmo utilizado para multiplicação nos computadores convencionais baseia-se no método padrão que se ensina no primário [15]. Sejam $A = a_{n-1}\cdots a_1 a_0$ e $B = b_{m-1}\cdots b_1 b_0$ dois números com n e m dígitos binários respectivamente. A multiplicação padrão de ambos consiste em multiplicar cada dígito do segundo operando pelos dígitos do primeiro e somar os termos similares sendo que, cada valor que excede o valor maximo de um dígito é somado ao termo seguinte. Isto pode ser visto com os operandos na forma: $A = 2^{n-1}a_{n-1} + \cdots + 2^1 a_1 + 2^0 a_0$ e $B = 2^{m-1}b_{m-1} + \cdots + 2^1 b_1 + 2^0 b_0$. Então,

$$\begin{aligned} A \cdot B &= (2^{n-1}a_{n-1} + \cdots + 2a_1 + a_0) \cdot (2^{m-1}b_{m-1} + \cdots + 2b_1 + b_0) \\ &= \textstyle\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} 2^{i+j} a_i b_j \end{aligned} \tag{3.1}$$

A complexidade desta operação é $n \cdot m$ multiplicações de dois dígitos binários (bits), ou seja $n \cdot m$ aplicações da porta AND entre estes bits, além de $\Theta(n \cdot m)$ somas com operandos de até n ou m bits.

Na figura 3.4, é mostrado como fazer a multiplicação de forma reversível a partir da soma. Este circuito é denominado MULT1.

O primeiro somador controlado soma a com 0 se $b_0 = 1$. Ou seja, o resultado é ab_0. O primeiro operando, neste caso a, permanece intacto. Em seguida, é aplicada novamente a soma, condicionada ao valor de b_1, sobre o resultado anterior e $2a$. A cada passo é acrescentado um bit com valor 0 em cada registrador. O resultado passa a ser $ab_0 + 2ab_1$. Após o i-ésimo somador, tem-se como resultado $ab_0 + 2ab_1 + \cdots + 2^{i-1}ab_{i-1}$. Como são realizadas n operações, o resultado final é $a(b_0 + 2b_1 + \cdots + 2^{n-1}b_{n-1})$. A entrada $2^j a$, para um j qualquer, é feita simplesmente através de j bits com o valor zero, justapostos aos bits de a, como bits menos significativos da entrada. Tendo em conta que $b_0 b_1 \cdots b_{n-1}$ é a expansão binária de b, temos como saída do circuito $a \cdot b$.

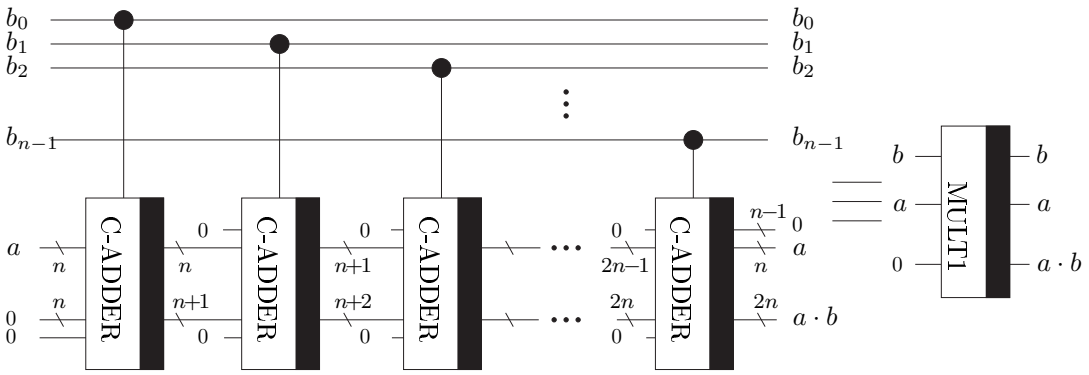

Figura 3.4: MULT1. Circuito reversível para a multiplicação de inteiros a e b de n bits, onde $b_0, \ldots, b_{n-1}$ é a decomposição binária de b. Os traços diagonais nas linhas indicam que elas são formadas por vários bits. A quantidade de bits é definida pelo valor junto a estas marcas.

Cada soma controlada possui complexidade $\Theta(n)$ e como são realizadas n adições, a complexidade do circuito é quadrática. A quantidade de bits utilizada é linear, sendo que o número exato de bits depende da implementação do somador controlado.

3.2.1 Multiplicação com a substituição de um dos operandos

A multiplicação produzida pelo circuito 3.4 é da forma $MULT1 : (a, b, 0) \rightarrow (a, b, ab)$. Assim como ocorre com a soma, é possível construir um circuito no qual o resultado substitui um dos operandos. A diferença ocorre em relação a necessidade de bits extras para armazenar o resultado. No caso da adição, o resultado possui no máximo um bit a mais do que o maior operando. Já na multiplicação, o resultado pode ter até o dobro do tamanho do operando substituído. Na figura 3.5, é mostrada a nossa proposta para a multiplicação com a substituição de um dos operandos MULT2:$(a, b, 0) \rightarrow (a, ab)$. O segundo e terceiro registradores passam a formar um único registrador.

O circuito 3.5 é muito parecido com o 3.4. Para cada adição controlada

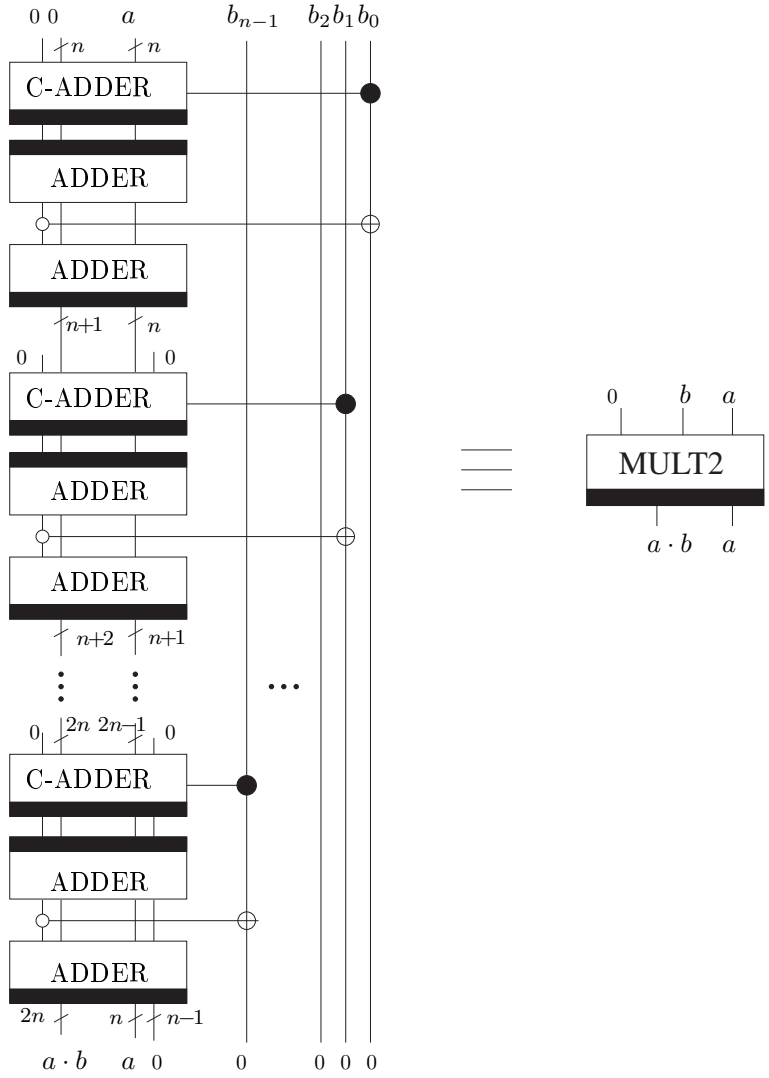

Figura 3.5: MULT2. Circuito reversível para a multiplicação de inteiros a e b de n bits, onde $b_0, \ldots, b_{n-1}$ é a decomposição binária de b. Apenas o operando a é mantido após a operação. Após aplicar a porta CNOT no bit b_i, este bit passa a ter o valor zero, podendo ser utilizado para outras operações.

foi acrescentada uma subtração e posterior adição que não altera o resultado. Entre ambas, há uma porta CNOT controlada negativamente pelo bit mais significativo do resultado cujo alvo é um dos bits de um dos operandos. Como foi comentado sobre o circuito 3.4, o resultado y_i após o i-ésimo passo é $a(b_0 + 2b_1 + \cdots + 2^{i-1}b_{i-1})$. Este resultado é o mesmo para i-ésimo somador condicional do circuito 3.5. Mesmo que todos os valores $b_0, \ldots, b_{i-1}$ fossem iguais a 1, tem-se que $y_i < a2^i$. Após o $i+1$-ésimo somador condicional, se $b_i = 1$, então $y_{i+1} = y_i + a2^i$ e conseqüentemente, $y_{i+1} \geq a2^i$. Caso contrário, $y_{i+1} = y_i$, o que implica em que $y_{i+1} < a2^i$. A operação seguinte é a subtração (reverso da soma) com $a2^i$ no primeiro registrador da soma e y_{i+1} no segundo registrador, que contém posteriormente o resultado da operação. Lembrando que, após a soma reversa, o último bit do segundo registrador é 0 se o segundo operando é maior do que o primeiro e 1 caso contrário. Pelo que foi dito antes, pode-se concluir que o último bit do segundo registrador será igual a 0 se $b_i = 1$ e 1 caso $b_i = 0$. Aplicando o CNOT controlado negativamente, tem-se que, em ambos os casos, b_i passa a ser 0. Esta análise pode ser feita para todos os bits $b_1, \ldots, b_{n-1}$.

O circuito com a substituição de um dos operandos possui várias vantagens em relação ao circuito que mantém ambos operandos. Uma delas, por exemplo, é permitir a operação inversa. No caso da soma, a operação inversa é a subtração, e no caso da multiplicação, a operação inversa é a divisão, caso o resultado seja múltiplo dos operandos.

3.3 Divisão reversível

Para fechar as quatro operações básicas com inteiros não negativos, mostramos na figura 3.6, a divisão de inteiros. Dados dois inteiros positivos a e b, este circuito retorna a div $b = \lfloor a/b \rfloor$, e a mod $b = a - (a \text{ div } b) \cdot b$.

A idéia do circuito é fazer subtrações sucessivas de b por a multiplicado

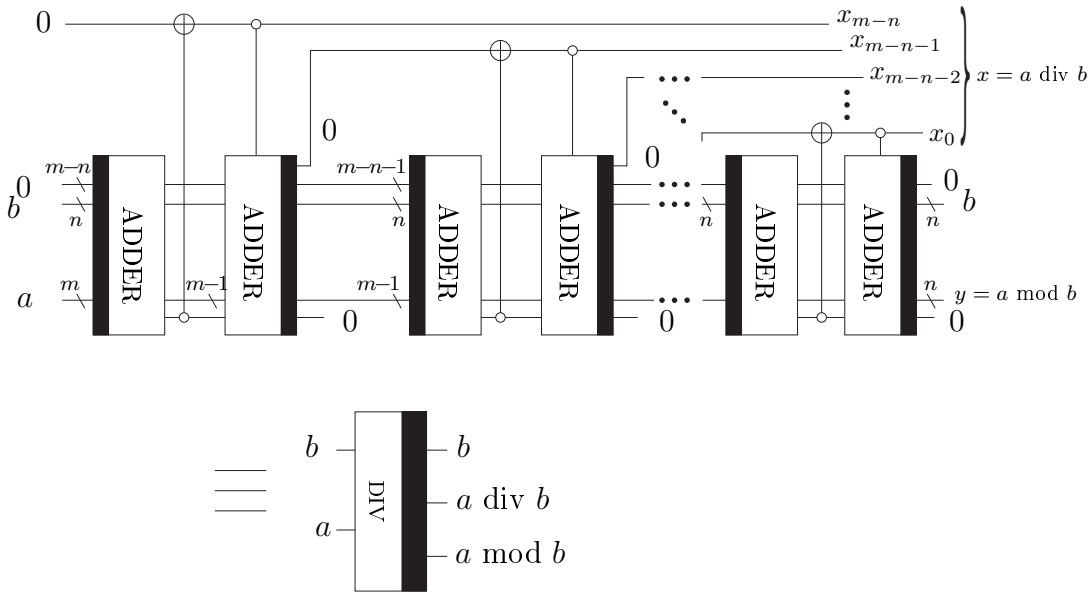

Figura 3.6: DIV. Circuito para a divisão de inteiros. A saída é composta por duas partes: uma guarda $\lfloor a/b \rfloor$ e a outra saída é o resto da divisão de a por b.

por potências de 2. Considere que a é formado por n bits e b por m. O maior divisor possível de b por a é composto por $n - m + 1$ bits.

Vejamos agora como isto é feito. Suponhamos que queremos dividir a por b, que possuem m e $n \leq m$ bits, respectivamente. Inicialmente é feita uma subtração de a por $b2^{m-n}$. Em seguida, é aplicado um CNOT controlado negativamente. O bit de controle é o bit mais significativo do último registrador.

Após esta operação, tem-se que

$$x_{m-n} = \begin{cases} 0, & \text{se } a < b2^{m-n} \\ 1, & \text{se } a \geq b2^{m-n}. \end{cases}$$

Se $a < b2^{m-n}$, a subtração é desfeita pelo segundo somador, que é controlado negativamente por x_{m-n}. Ou seja, a subtração ocorre apenas se $x_{m-n} = 1$. Sendo assim, podemos escrever o resultado como $a - (x_{m-n})b2^{m-n}$. Este valor é congruente a a mod b, pois, ou é igual ao valor a inicial ou é igual a a menos um múltiplo de b.

Considerando que $b2^{m-n}$ possui exatamente m bits, este resultado terá no máximo $m-1$ bits, pois a subtração entre dois operandos com a mesma quantidade de bits produz um resultado com pelo menos um bit a menos, quando se subtrai o menor do maior valor.

Em seguida, faz-se outra subtração sobre o resultado, mas o primeiro operando passa a ser $b2^{m-n-1}$ em vez de $b2^{m-n}$.

Isto implica em que

$$x_{m-n-1} = \begin{cases} 0, & \text{se } a - (x_{m-n})b2^{m-n} < b2^{m-n-1} \\ 1, & \text{se } a - (x_{m-n})b2^{m-n} \geq b2^{m-n-1}. \end{cases}$$

Esta relação pode ser escrita também como

$$x_{m-n-1} = \begin{cases} 0, & \text{se } a < b2^{m-n}x_{m-n} + b2^{m-n-1} \\ 1, & \text{se } a \geq b2^{m-n}x_{m-n} + b2^{m-n-1}. \end{cases}$$

Pelo mesmo raciocínio anterior, o resultado após a segunda subtração e posterior soma condicional será $a - (b2^{m-n}x_{m-n} + b2^{m-n-1}x_{m-n-1})$. Colocando b em evidência temos $a - b(2^{m-n}x_{m-n} + 2^{m-n-1}x_{m-n-1})$. Novamente, o resultado terá um bit a menos em relação ao resultado anterior.

Após i subtrações e somas, tem-se:

$$x_{m-n-i+1} = \begin{cases} 0, & \text{se } a < b(2^{m-n}x_{m-n} + 2^{m-n-1}xm - n - 1 + \cdots + 2^{m-n-i+1}) \\ 1, & \text{se } a \geq b(2^{m-n}x_{m-n} + 2^{m-n-1}xm - n - 1 + \cdots + 2^{m-n-i+1}). \end{cases}$$

E o resultado seguinte, no último registrador é $a - b(2^{m-n}x_{m-n} + 2^{m-n-1}x_{m-n-1} + \cdots + 2^{m-n-i+1}x_{m-n-i+1})$. Este resultado possui no máximo $m-i$ bits.

O resultado sempre permanece congruente a $a \bmod b$, pois após cada passo completo (subtração, CNOT e soma condicional), o resultado ou é igual ao resultado anterior ou houve uma subtração de um múltiplo de b.

Como são feitas $m-n+1$ subtrações seguidas de somas condicionais, podemos afirmar que o valor final no último registrador é $y = a - b(2^{m-n}x_{m-n} + 2^{m-n-1}x_{m-n-1} + \cdots + 2x_1 + x_0)$. Tem-se que y é formado por, no máximo,

$n-1$ bits, logo, $y < b$. Como y é o resultado de sucessivas subtrações de múltiplos de b a partir de a, tem-se que y é o resto da divisão de a por b. Chamando de $x = 2^{m-n}x_{m-n} + \cdots + 2x_1 + x_0$, tem-se que $y = a - bx$. Isto significa que $x = \lfloor a/b \rfloor$.

3.4 Operações Modulares

3.4.1 Soma modular reversível

A partir do operador reversível de adição de inteiros, é possível construir um operador para a soma modular da forma:

$$(a, b, N) \rightarrow (a, a + b \bmod N, N).$$

Na figura 3.7, é mostrado o circuito que implementa a soma modular reversível proposto por Vedral, Barenco e Ekert [46] com uma correção. No diagrama original, há duas portas NOT's, no último bit do segundo registrador, que indicam que o bit do último registrador é controlado negativamente pela primeira porta CNOT que aparece. Na realidade, este bit é controlado positivamente. No circuito original, a faixa preta da porta resultante ADDER-MOD está na esquerda, o correto é na direita, conforme mostrado na figura 3.7.

A idéia do circuito para a adição modular é a seguinte:

- Aplica-se a adição entre os operandos a e b utilizando a porta ADD.
- Utilizando a porta ADD de forma reversa, compare o resultado da operação anterior com o valor N.
- Caso o resultado da soma de a e b seja maior do que N, subtraia N deste resultado, utilizando outro somador de forma reversa.
- Faça a limpeza do lixo dos bits auxiliares, no caso, o bit que serviu para a comparação.

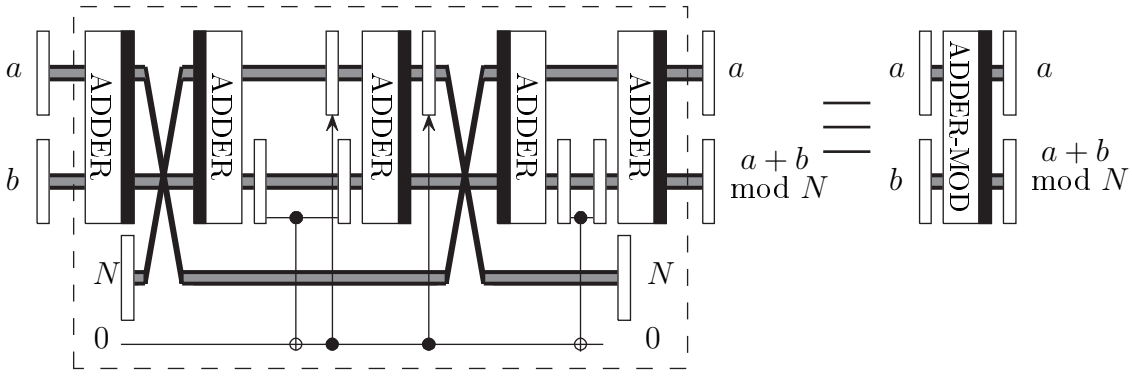

Figura 3.7: Circuito para soma modular reversível proposta por Vedral, Barenco e Ekert [46] com uma correção.

O circuito para soma modular realiza a seguinte operação:

$$U_N : (a, b) \rightarrow \begin{cases} (a, a+b) & \text{se } a+b < N \\ (a, a+b-N) & \text{se } a+b \geq N. \end{cases}$$

Subtração Modular

A operação $U_N : (a, b) \rightarrow (a, c)$, no caso em que $a + b < N$, implica em que $c = a+b$, e como $b \geq 0$, então, $c \geq a$. No caso complementar, $a+b \geq N$, tem-se que $c = a+b-N$, o que implica em que $b-N = c-a$. Como $b < N$, então, $c < a$. Isto significa que, o operador para a soma modular no sentido reverso, é definido por:

$$U_N^{\dagger} : (a, c) \rightarrow \begin{cases} (a, c-a) & \text{se } c \geq a \\ (a, N+c-a) & \text{se } c < a. \end{cases}$$

Em ambos os casos do uso do operador reverso de U_N, $U_N^{\dagger}$, temos que para a entrada (a, c), o segundo registrador possui o valor $c - a \bmod N$. Ou seja, a subtração modular pode ser feita através do circuito para a adição modular, desde que utilizado no sentido reverso.

3.4.2 Multiplicação Modular Reversível

Vedral, Barenco e Ekert, em [46], mostram como executar a multiplicação modular de forma reversível. O circuito é mostrado na figura 3.8.

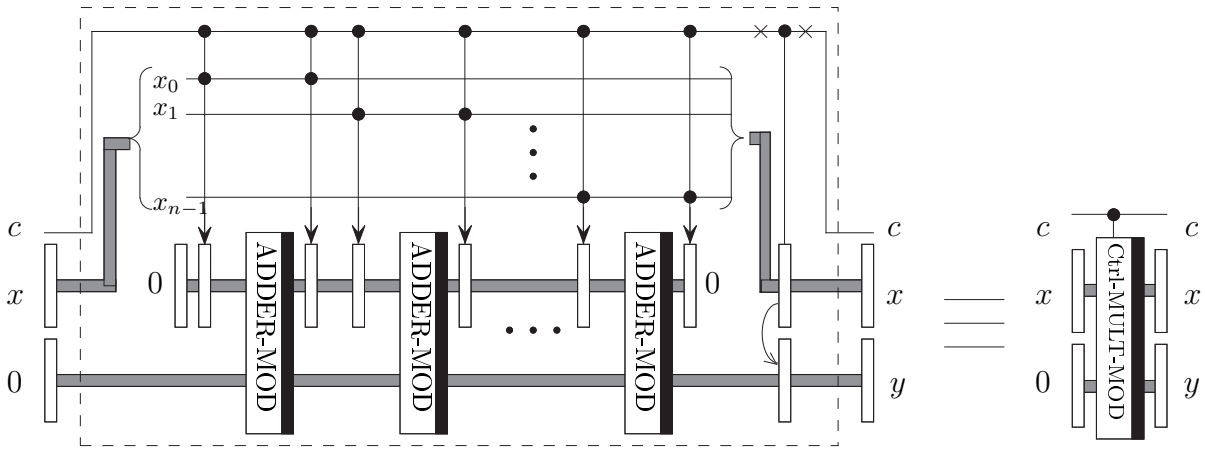

Figura 3.8: Circuito para a multiplicação modular controlada reversível proposta por Vedral, Barenco e Ekert [46]. Apenas um dos operandos pode estar em superposição de estados. A única diferença entre este circuito e o da figura 5 do artigo [46] é a quantidade de bits dos operandos. Na figura acima, os operandos possuem n bits, enquanto na figura do artigo os operandos possuem $n+1$ bits.

Este circuito pode servir de base para a exponenciação modular usada no algoritmo de Shor. Mas este circuito possui a limitação de que um dos operandos é fixo, ou seja, não pode ser usado em superposição.

Na figura 3.9, é mostrado o circuito MULT-MOD1:$(N, a, b, 0) \rightarrow (N, a, b, ab \bmod N)$, que permite a multiplicação modular com os dois operandos podendo estar em superposição.

Multiplicação Modular com substituição de um dos operandos

Caso algum dos operandos seja co-primo com N, é possível modificar o circuito MULT-MOD1 de forma que o resultado substitua este operando. Para isso, é necessário conhecer o inverso multiplicativo deste operando módulo N. Isto pode ser feito através de um circuito que implemente o algoritmo

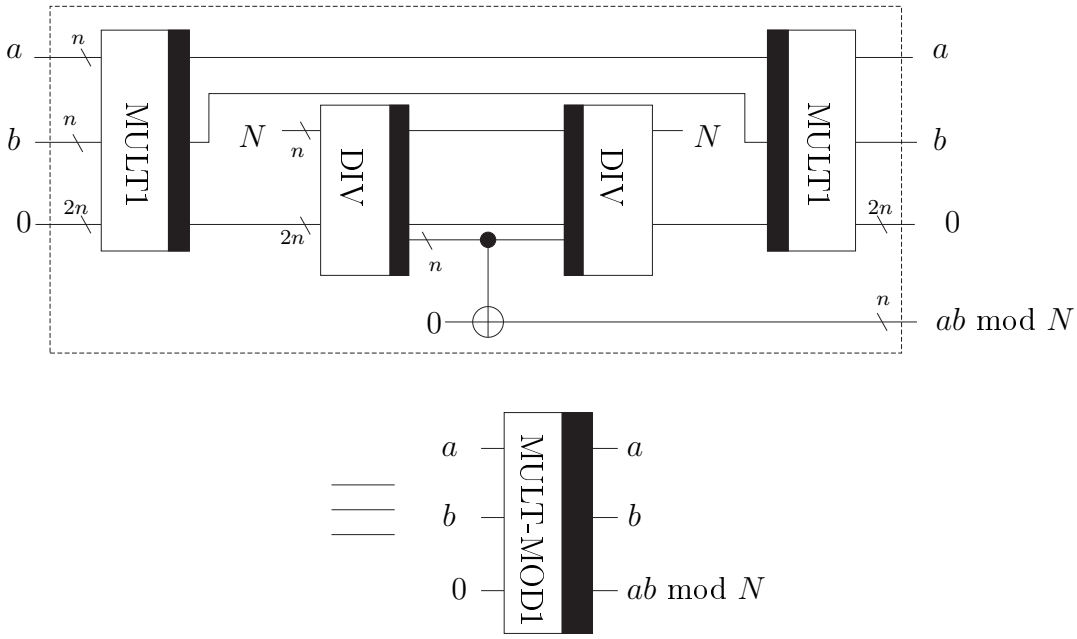

Figura 3.9: MULT-MOD1. Circuito para a multiplicação modular reversível. Ambos os operandos podem estar em superposição.

estendido de Euclides [22]. Em [19], é descrita uma forma de implementação reversível deste algoritmo. A complexidade total de portas deste procedimento é quadrático em relação ao número de bits dos operandos e o espaço utilizado é linear [19]. O operador executa a seguinte operação:

$$\text{Euclides-Ext} : (N, a, 0) \rightarrow (N, a, a^{-1} \bmod N).$$

De posse do inverso multiplicativo de a módulo N, tem-se a seguinte configuração de registradores: $(a^{-1}, a, b, N, 0, 0)$. Aplicando o MULT-MOD1 com os operandos nos registradores 2 e 3 e guardando o resultado no registrador 5, obtém-se: $(a^{-1}, a, b, N, 0, ab \bmod N)$. Aplicando o mesmo circuito MULT-MOD1 mas na ordem reversa com os operandos nos registradores 1 e 5 e guardando o resultado no registrador 3, obtém-se finalmente $(a^{-1}, a, ab \bmod N, N, 0, 0)$. Pode-se limpar o registrador que armazena a^{-1} executando o algoritmo de Euclides estendido na ordem inversa. Na figura 3.10 é descrito o circuito que implementa esta versão.

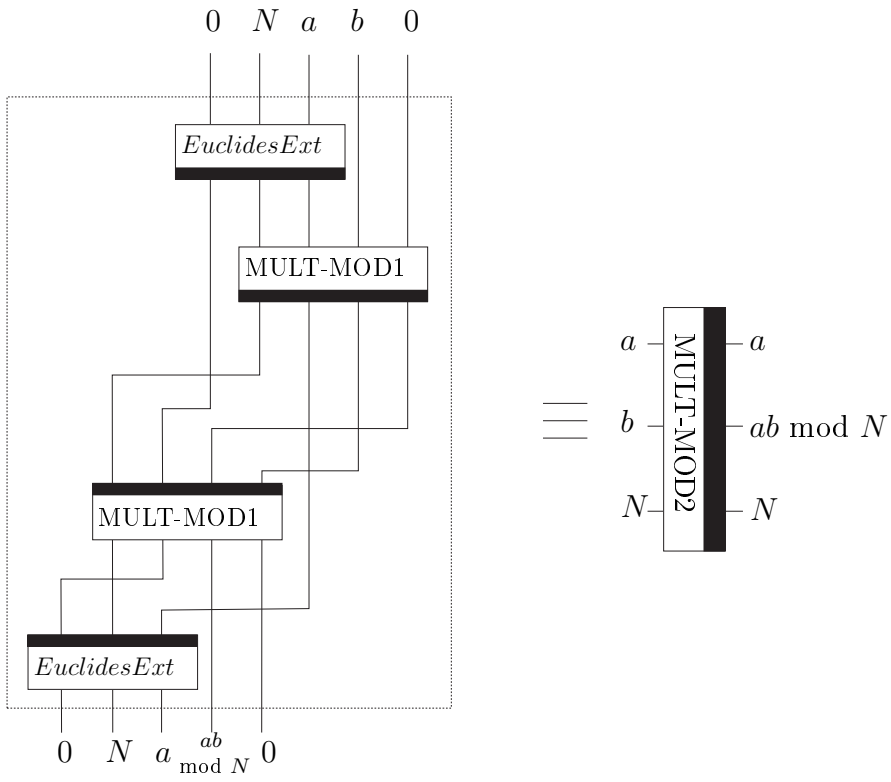

Figura 3.10: MULT-MOD2. Circuito para multiplicação modular reversível com o resultado substituindo um dos operandos. Ambos operandos podem estar em superposição.

Esta função seria definida como:

$$\text{MULT-MOD2} : (N, a, b) \rightarrow (N, a, ab\ mod N)$$

Uma variação útil da multiplicação modular é inserir um bit extra que define se a operação será executada ou não. A versão controlada para qualquer operação reversível é facilmente projetada conforme foi comentada na sub-seção 3.1.1.

3.5 Algoritmo Karatsuba para multiplicação

Ao descrevermos a multiplicação na equação (3.1) da seção anterior, os operandos A e B foram escritos de acordo com a sua decomposição binária. Se escrevermos os operandos A e B em uma outra base β, $A = a'_{n'-1} \cdots a'_1 a'_0$

e $B = b'_{m'-1} \cdots b'_1 b'_0$, onde $n' = \lceil n/\log_2 \beta \rceil$ e $m' = \lceil m/\log_2 \beta \rceil$ e $a'_{n'-1}, \ldots, a'_0$ e $b'_{m'-1}, \ldots, b'_0$ são os dígitos de A e B na nova base. Então

$$C = A \cdot B = \sum_{i=0}^{n'-1} \sum_{j=0}^{m'-1} \beta^{i+j} a'_i b'_j. \tag{3.2}$$

C terá $n' + m'$ dígitos na base β.

Se β possui o tamanho da palavra do computador, por exemplo 32, vemos que o número de multiplicações necessárias, neste computador, é $n'm' = \frac{nm}{64}$. Em geral, considera-se que ambos operandos possuem aproximadamente o mesmo tamanho($n \approx m$), neste caso, a ordem de complexidade do algoritmo padrão para a multiplicação é quadrática.

O procedimento padrão para a multiplicação também pode ser feito de forma recursiva. Sejam $A = a_{n-1} \cdots a_1 a_0$ e $B = b_{n-1} \cdots b_1 b_0$ dois valores escritos na forma binária. Tomando como base $\beta = 2^{\lfloor \frac{n}{2} \rfloor}$, eles podem ser expressados como $A = 2^{\lfloor \frac{n}{2} \rfloor} a_{n-1} \cdots a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor - 1} \cdots a_0$ e $B = 2^{\lfloor \frac{n}{2} \rfloor} b_{n-1} \cdots b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor - 1} \cdots b_0$.

Chamando de $A_1 \equiv a_{n-1} \cdots a_{\lfloor \frac{n}{2} \rfloor}$, $A_0 \equiv a_{\lfloor \frac{n}{2} \rfloor - 1} \cdots a_0$, $B_1 \equiv b_{n-1} \cdots b_{\lfloor \frac{n}{2} \rfloor}$ e $B_0 \equiv b_{\lfloor \frac{n}{2} \rfloor - 1} \cdots b_0$, tem-se que as expressões anteriores podem ser reescritas como $A = A_1\beta + A_0$ e $B = B_1\beta + B_0$. Logo,

$$A \cdot B = (A_1\beta + A_0) \cdot (B_1\beta + B_0) = A_1B_1\beta^2 + (A_1B_0 + A_0B_1)\beta + A_0B_0 \tag{3.3}$$

A multiplicação de dois operandos com n dígitos cada pode ser transformada em quatro multiplicações com operandos de $\lceil n/2 \rceil$ dígitos. Este processo pode ser feito de forma recursiva. Seja $T(n)$ a complexidade para a multiplicação de n bits cada , então

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 4T(\lceil \frac{n}{2} \rceil) + \Theta(n), & \text{se } n > 1. \end{cases}$$

Desta forma, tem-se que a complexidade permanece $\Theta(n^2)$.

Karatsuba junto com Ofmam, em 1963, propuseram uma variação deste algoritmo que diminui a ordem de complexidade de tempo para $\Theta(n^{\log_2 3})$ [18].

Como $(A_1 + A_0)(B_1 + B_0) = A_1B_1 + A_1B_0 + A_0B_1 + A_0B_0$, segue que

$$A_1B_0 + A_0B_1 = (A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0 \tag{3.4}$$

Eles, utilizando a propriedade (3.4) na equação (3.3), chegaram que

$$A \cdot B = A_1B_1\beta^2 + ((A_1 + A_0)(B_1 + B_0) - (A_1B_1 + A_0B_0))\beta + A_0B_0 \tag{3.5}$$

A equação (3.5) possui apenas três multiplicações distintas de tamanho, aproximadamente, $\frac{n}{2}$: A_1B_1, $(A_1 + A_0)(B_1 + B_0)$ e A_0B_0.

A versão recursiva do algoritmo padrão possui quatro chamadas recursivas de tamanho, aproximadamente, $\frac{n}{2}$, além de duas adições com operandos de tamanho n e outra de tamanho $2n$. A versão de Karatsuba possui três chamadas recursivas, duas adições com operandos de tamanho aproximadamente $\frac{n}{2}$, outras duas de tamanho n e uma de tamanho $2n$, além de uma subtração da ordem de n.

Em linhas gerais, o algoritmo Karatsuba é descrito em 3.5.1.

A relação de recorrência da complexidade do algoritmo Karatsuba é

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 3T(\lceil \frac{n}{2} \rceil) + \Theta(n), & \text{se } n > 1. \end{cases}$$

Assim, a complexidade do algoritmo de multiplicação proposto por Karatsuba é $\Theta(n^{\log_2 3})$. Isto é aproximadamente $\Theta(n^{1.59})$, melhor do que $\Theta(n^2)$ correspondente ao algoritmo trivial.

Tomamos como base da recursão o caso $n = 1$, mas em um computador convencional, a base pode ser o tamanho da palavra do processador, pois esta operação possui custo constante conforme foi visto anteriormente.

Pelo fato de o algoritmo Karatsuba fazer mais operações de adição e subtração em cada nível da recorrência, o algoritmo convencional possui um custo computacional menor para operandos de pequeno tamanho. Neste caso,

Algoritmo 3.5.1 Algoritmo $Karatsuba(n, A, B)$

Requer: $n \in \mathbb{Z}$, $A = a_{n-1} \cdots a_1 a_0$ e $B = b_{n-1} \cdots b_1 b_0$

se $n = 1$ **então**

devolva $A \cdot B$

fim se

$A_1 \leftarrow a_{n-1} \cdots a_{\lfloor \frac{n}{2} \rfloor}$

$A_0 \leftarrow a_{\lfloor \frac{n}{2} \rfloor - 1} \cdots a_0$

$B_1 \leftarrow b_{n-1} \cdots b_{\lfloor \frac{n}{2} \rfloor}$

$B_0 \leftarrow b_{\lfloor \frac{n}{2} \rfloor - 1} \cdots b_0$

$C_1 \leftarrow Karatsuba(\lceil \frac{n}{2} \rceil, A_1, B_1)$

$C_0 \leftarrow Karatsuba(\lfloor \frac{n}{2} \rfloor, A_0, B_0)$

$C \leftarrow C_1 * 2^{\lfloor \frac{n}{2} \rfloor}$

$C \leftarrow C + Karatsuba(\lceil \frac{n}{2} + 1 \rceil, A_1 + A_0, B_1 + B_0) - (C_1 + C_0)$

$C \leftarrow C * 2^{\lfloor \frac{n}{2} \rfloor}$

$C \leftarrow C + C_0$

devolva H

pode-se usar como base da recursão o algoritmo convencional para valores de n abaixo de um certo patamar.

3.5.1 Algoritmo Karatsuba Reversível

Para a descrição da versão reversível do algoritmo de multiplicação Karatsuba, pode-se utilizar a porta de adição reversível descrita na figura 3.2 ou alguma das outras que implementam a soma de inteiros como, por exemplo, a porta proposta por Draper e outros em [11] que utiliza $\Theta(n)$ portas, mas sua profundidade é de $\Theta(\log n)$. Nas figuras 3.11 a 3.15, esta porta é chamada de ADDER n, onde n é o número de bits dos operandos que estão sendo somados. Como foi feito na seção 2.2 do capítulo 2, a faixa preta na representação de portas em um circuito indica o sentido da operação. No caso da porta ADDER, a faixa no lado dos bits de saída computa a adição, enquanto que o sentido inverso computa a subtração se o primeiro operando for maior do que o segundo.

A figura 3.11 descreve um circuito intermediário chamado KARATSUBA1. É uma versão recursiva reversível do algoritmo Karatsuba. A porta KARATSUBA1 $\lceil \frac{n}{2} \rceil$ deve ser substituída por um circuito similar ao original, mas com o tamanho da entrada metade do original. Quando a complexidade do circuito recursivo KARATSUBA1 for maior do que o circuito padrão da multiplicação para entrada do mesmo tamanho, o circuito deve ser substituído por este último.

Na tabela 3.2, é mostrado o conteúdo dos registradores durante a evolução do algoritmo.

Após a execução do circuito KARATSUBA1, restam alguns bits extras com valores indesejáveis. Estes valores são zerados utilizando a primeira versão de coleta de lixo proposto por Bennett (explicado na seção 2.2 do capítulo anterior), onde o valor do resultado é copiado para novos bits utilizando por-

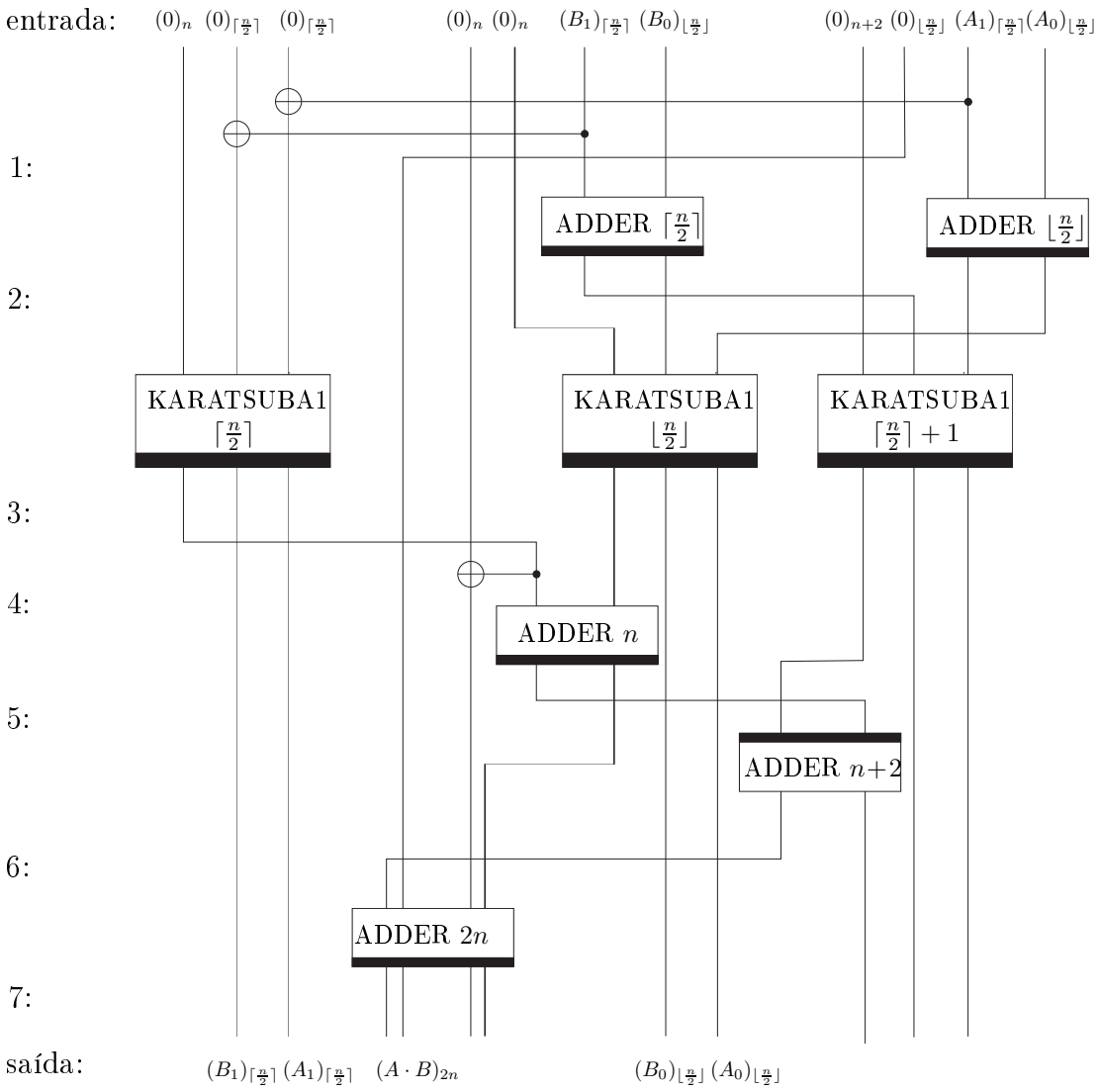

Figura 3.11: KARATSUBA1. Versão paralela do circuito Karatsuba reversível. A computação é feita de cima para baixo e o resultado da operação de cada porta é armazenado nos bits mais à esquerda.

entrada	0	0	0	0	0	B_1	B_0	0	0	A_1	A_0
1	0	B_1	A_1	0	0	B_1	B_0	0	0	A_1	A_0
2	0	B_1	A_1	0	0	B_1+B_0	B_0	0	0	A_1+A_0	A_0
3	B_1A_1	B_1	A_1	0	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$	0	A_1+A_0	A_0
4	B_1A_1	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$	0	A_1+A_0	A_0
5	B_1A_1 $+B_0A_0$	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$	0	A_1+A_0	A_0
6	B_1A_1 $+B_0A_0$	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$ $-(B_1A_1+B_0A_0)$	0	A_1+A_0	A_0
6	B_1A_1 $+B_0A_0$	B_1	A_1	$2^nB_1A_1$ $+B_0A_0$		B_1+B_0	B_0	$2^{\lfloor\frac{n}{2}\rfloor}(B_1A_0+A_1B_0)$		A_1+A_0	A_0
7	B_1A_1 $+B_0A_0$	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$2^{\lfloor\frac{n}{2}\rfloor}(B_1A_0+A_1B_0)$ $+2^nB_1A_1+B_0A_0$		A_1+A_0	A_0
saida	lixo	B_1	A_1	lixo			B_0	$A\cdot B$		lixo	A_0

Tabela 3.2: Evolução detalhada do circuito KARATSUBA1 mostrado na figura 3.11. A ordem dos registradores segue a ordem inicial. Além do lixo indicado na última linha, há o lixo devido às chamadas recursivas.

tas CNOT's, e o circuito é executado novamente no sentido inverso. A Figura 3.12 mostra o esquema deste algoritmo chamado KARATSUBA(1). Todas as versões do circuito KARATSUBA que tiverem a numeração entre parêntesis possuem os bits extras com valor nulo no final do procedimento.

A multiplicação de operandos com n bits é feita através de 5 portas ADDER's e 3 chamadas recursivas de tamanho aproximadamente $\frac{n}{2}$. Sendo assim, sendo $T_1(n)$ o tamanho do circuito KARATSUBA1,

$$T_1(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 3T_1(\lceil\frac{n}{2}\rceil) + \Theta(n), & \text{se } n > 1. \end{cases}$$

Desta relação de recorrência podemos afirmar que $T_1(n) = \Theta(n^{\log_2 3})$.

Conforme foi comentado antes, a soma de operandos com kn bits, onde k é uma constante qualquer, pode ser implementada com profundidade $\Theta(\log n)$. As três chamadas recursivas KARATSUBA1, em cada nível de recursão, podem ser feitas simultaneamente. Isto significa que a profundidade do circuito,

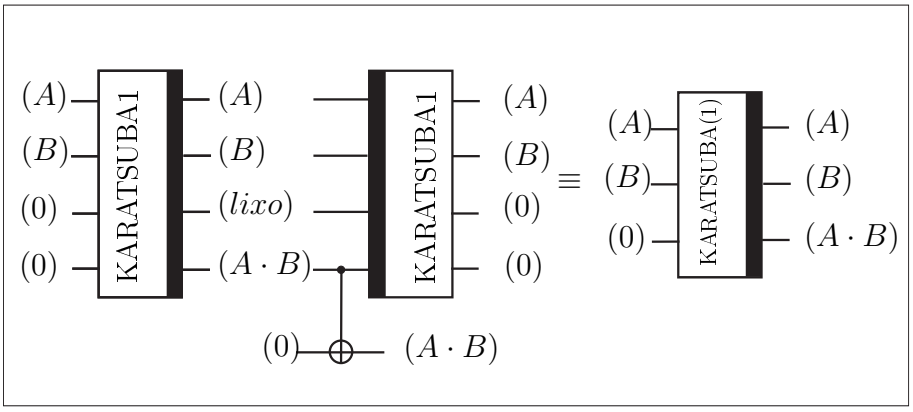

Figura 3.12: Circuito KARATSUBA(1) com os bits extras zerados.

$P_1(n)$, pode ser expressa como:

$$P_1(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ P_1(\lceil \frac{n}{2} \rceil) + \Theta(\log n), & \text{se } n > 1. \end{cases}$$

A partir desta relação tem-se que $P_1(n) = \Theta(\log^2 n)$.

Como estamos fazendo as chamadas recursivas em paralelo, não é possível reaproveitar os bits usados em uma chamada recursiva para o uso em outra chamada no mesmo nível. Sendo assim, a quantidade de espaço utilizado, $S_1(n)$, para entradas de tamanho n é

$$S_1(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 3S_1(\lceil \frac{n}{2} \rceil) + \Theta(n), & \text{se } n > 1. \end{cases}$$

Ou seja, o espaço total é da mesma ordem de complexidade do tamanho do circuito, $S_1(n) = \Theta(n^{\log_2 3})$. As ordens de complexidade tanto de espaço, quanto de profundidade e de tamanho total do circuito KARATSUBA(1) são as mesmas do circuito KARATUSUBA1, pois KARATSUBA(1) é formado por dois circuitos KARATSUBA1 e mais algumas portas CNOT, aumentando no máximo para o dobro do tamanho do circuito KARATSUBA1.

Caso se deseje minimizar o espaço em vez da profundidade, pode-se executar o circuito de forma seqüencial as chamadas recursivas, combinando com o primeiro ou segundo procedimentos de limpeza de lixo de Bennett

comentados no capítulo 2. Isto permite que os bits usados em uma chamada possam ser reaproveitos para as outras chamadas no mesmo nível de recursão. Na figura 3.13, é mostrada uma versão do circuito Karatsuba reversível na qual não há preocupação com o paralelismo. A vantagem desta versão para a anterior é o fato de não necessitar fazer nenhuma "cópia" e nem usar nenhuma porta além das que estão imbutidas nos somadores.

A seqüência de valores de cada registrador é dada na tabela 3.3.

entrada	0	B_1	A_1	0	B_0	A_0	0	0
1	B_1A_1	B_1	A_1	B_0A_0	B_0	A_0	0	0
2	B_1A_1	B_1+B_0	A_1+A_0	B_0A_0	B_0	A_0	0	0
3	B_1A_1 $+B_0A_0$	B_1+B_0	A_1+A_0	B_0A_0	B_0	A_0	$(B_1+B_0)\cdot(A_1+A_0)$	0
4	B_1A_1 $+B_0A_0$	B_1+B_0	A_1+A_0	B_0A_0	B_0	A_0	$(B_1+B_0)\cdot(A_1+A_0)$ $-B_1A_1+B_0A_0$	0
5	B_1A_1	B_1+B_0	A_1+A_0	B_0A_0	B_0	A_0	$(B_1+B_0)\cdot(A_1+A_0)$ $-B_1A_1+B_0A_0$	0
6	B_1A_1	B_1	A_1	B_0A_0	B_0	A_0	$2^{\lfloor \frac{n}{2} \rfloor}(B_1A_0+A_1B_0)$ $+2^nB_1A_1+B_0A_0$	
saida	lixo	B_1	A_1	lixo	B_0	A_0	$A\cdot B$	

Tabela 3.3: Evolução do circuito KARATSUBA2 mostrado na figura 3.13. Não é mostrado o lixo devido às chamadas recursivas.

Para operandos de n bits, a quantidade total de portas $T_2(n)$ e o espaço usado $S_2(n)$ do circuito KARATSUBA2 coincide com as complexidades equivalentes do circuito KARATSUBA1, ou seja, $T_2(n) = S_2(n) = \Theta(n^{\log_2 3})$. Como não há preocupação da execução em paralelo, a profundidade $P_2(n)$ também é $\Theta(n^{\log_2 3})$.

Da mesma forma como foi feita a limpeza de lixo para o circuito KARATSUBA1, pode ser feito para KARATSUBA2, gerando um novo circuito com KARATSUBA(2), com as mesmas ordens de complexidade do KARATSUBA2.

Em vez de se aplicar o primeiro método de Bennett, poderia ser aplicado o

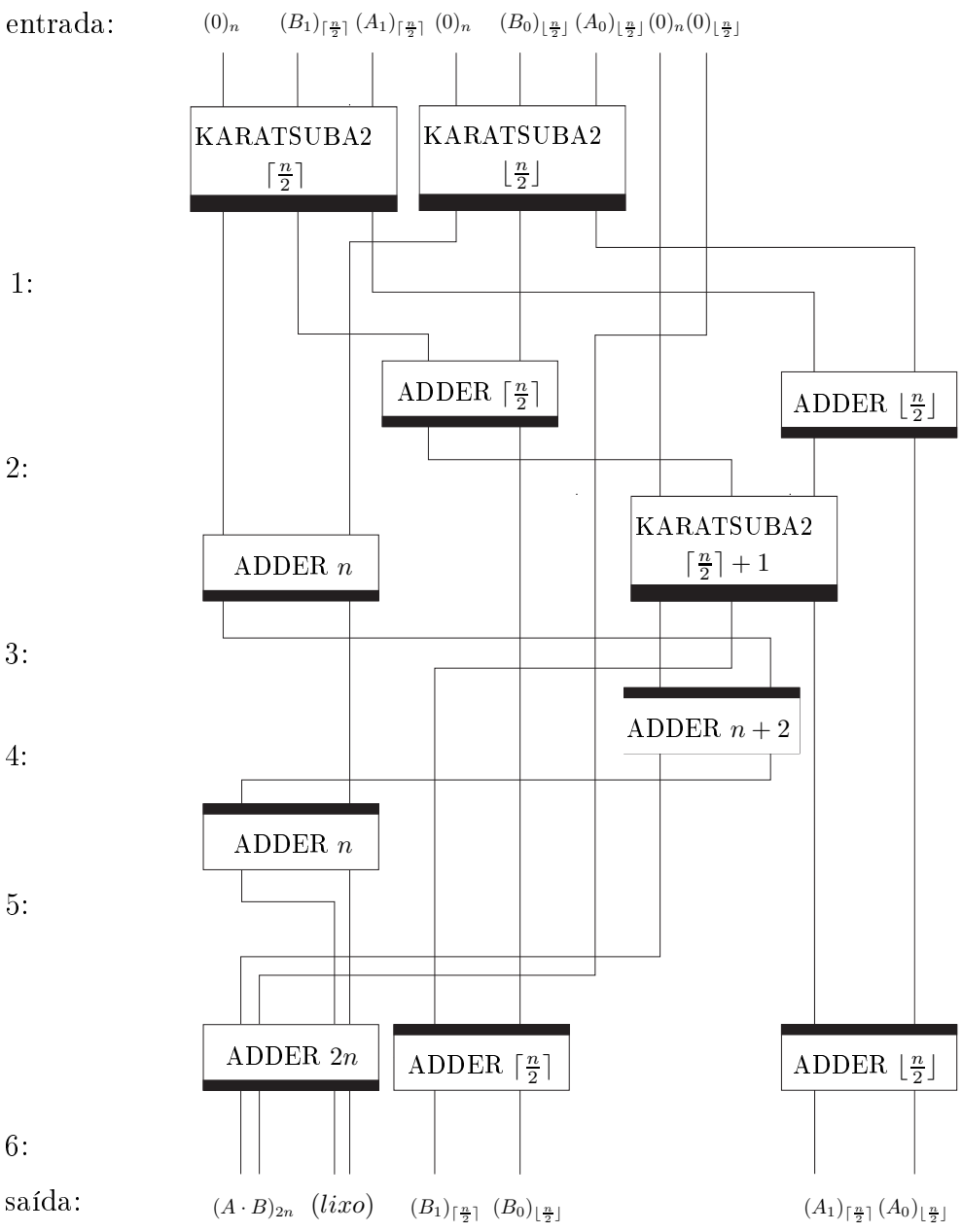

Figura 3.13: KARATSUBA2. Versão seqüencial do circuito Karatsuba reversível.

segundo método de limpeza de lixo de Bennett, mas o problema da aplicação deste último método é o fato de ser necessário dividir o circuito em passos do mesmo tamanho, o que não acontece com a divisão lógica do algoritmo.

No caso de algoritmos recursivos, pode-se fazer a limpeza de lixo recursivamente. A figura 3.14 mostra o circuito KARATSUBA(3), construído a partir do KARATSUBA1 com esquema de limpeza de lixo recursivo. Na tabela 3.4, é mostrado o conteúdo dos registradores do circuito KARATSUBA(3). Como no final todos os bits auxiliares são zerados, as chamadas recursivas não retornam lixo além do resultado.

entrada	0	0	0	0	0	B_1	B_0	0	0	A_1	A_0
1	0	B_1	A_1	0	0	B_1	B_0	0	0	A_1	A_0
2	0	B_1	A_1	0	0	B_1+B_0	B_0	0	0	A_1+A_0	A_0
3	B_1A_1	B_1	A_1	0	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$	0	A_1+A_0	A_0
4	B_1A_1	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$	0	A_1+A_0	A_0
5	B_1A_1 $+B_0A_0$	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$	0	A_1+A_0	A_0
6	B_1A_1 $+B_0A_0$	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$(B_1+B_0)\cdot(A_1+A_0)$ $-B_1A_1-B_0A_0$	0	A_1+A_0	A_0
6	B_1A_1 $+B_0A_0$	B_1	A_1	$2^nB_1A_1$ $+B_0A_0$		B_1+B_0	B_0	$2^{\lfloor\frac{n}{2}\rfloor}(B_1A_0+A_1B_0)$		A_1+A_0	A_0
7	B_1A_1 $+B_0A_0$	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$2^{\lfloor\frac{n}{2}\rfloor}(B_1A_0+A_1B_0)$ $+2^nB_1A_1+B_0A_0$		A_1+A_0	A_0
8	B_1A_1	B_1	A_1	B_1A_1	B_0A_0	B_1+B_0	B_0	$A\cdot B$		A_1+A_0	A_0
9	B_1A_1	B_1	A_1	0	B_0A_0	B_1+B_0	B_0	$A\cdot B$		A_1+A_0	A_0
10	0	B_1	A_1	0	0	B_1+B_0	B_0	$A\cdot B$		A_1+A_0	A_0
11	0	B_1	A_1	0	0	B_1	B_0	$A\cdot B$		A_1	A_0
saida	0	0	0	0	0	B_1	B_0	$A\cdot B$		A_1	A_0

Tabela 3.4: Evolução do circuito KARATSUBA(3) mostrado na figura 3.14. As chamadas recursivas não produzem lixo.

A complexidade total de portas do circuito KARATSUBA(3) para entradas de n bits, $T_3(n)$ é dada pela seguinte relação de recursão.

$$T_3(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 5T_1(\lceil\frac{n}{2}\rceil) + \Theta(n), & \text{se } n > 1. \end{cases}$$

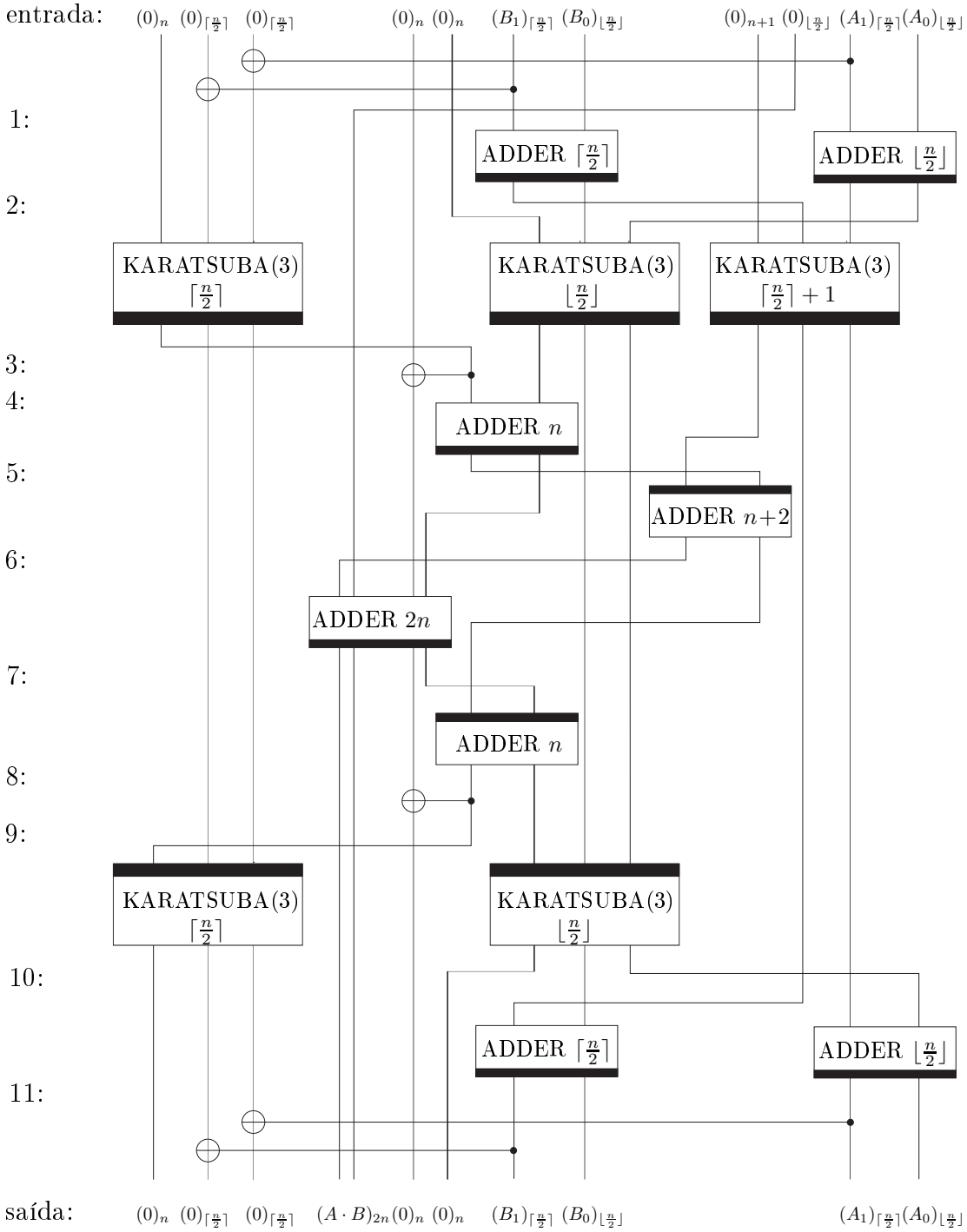

Figura 3.14: KARATSUBA(3) Procedimento recursivo para limpeza de lixo da versão paralela

Isto implica em que $T_3(n) = \Theta(n^{\log_2 5})$.

Em cada nível do circuito KARATSUBA(3), as três chamadas recursivas usadas para o cálculo da multiplicação podem ser executadas em paralelo, assim como ocorria no KARATSUBA(1). O mesmo acontece com as duas chamadas seguintes usadas para limpar os bits indesejáveis. A profundidade deste circuito, P_3, torna-se, então,

$$P_3(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ 2P_3(\lceil \frac{n}{2} \rceil) + \Theta(\log n), & \text{se } n > 1. \end{cases}$$

O que significa que a profundidade $P_3(n)$, para n bits de entrada é linear.

A quantidade de bits usados por KARATSUBA(3) é a mesma de KARATSUBA(1), ou seja, $S_3(n) = \Theta(n^{\log_2 3})$. Isto acontece porque apesar de fazer a limpeza em cada nível de recursão, os bits "zerados" não são usados devido ao paralelismo das outras chamadas recursivas.

Este método de limpeza de lixo recursivo, com as chamadas em paralelo, é elegante, mas aumenta consideravelmente a quantidade total de portas do circuito, tornando-o pior do que o algoritmo convencional. Caso a preocupação maior seja o espaço, é possível tornar o espaço linear sem a necessidade de aumentar tanto o tamanho do circuito como em KARATSUBA(3). Na figura 3.15, é mostrado o circuito KARATSUBA(4) construído a partir de KARATSUBA(2), que utiliza apenas $\Theta(n)$ bits, onde n é a quantidade de bits necessários para representar os operandos que se deseja multiplicar.

A única diferença do circuito KARATSUBA(4) para o circuito KARATSUBA2 é a adição de duas chamadas recursivas executadas na ordem reversa, que zeram os dois registradores que continham o resultado de duas multiplicações parciais. Logo, o circuito KARATSUBA(4) faz 5 chamadas recursivas em cada nível de recursão. Isto significa que a quantidade total de portas $T_4(n)$ é $\Theta(n^{\log_2 5})$. Como as chamadas recursivas são executadas em seqüência, a profundidade do circuito, $P_4(n)$ também é igual a $\Theta(n^{\log_2 5})$. Cada

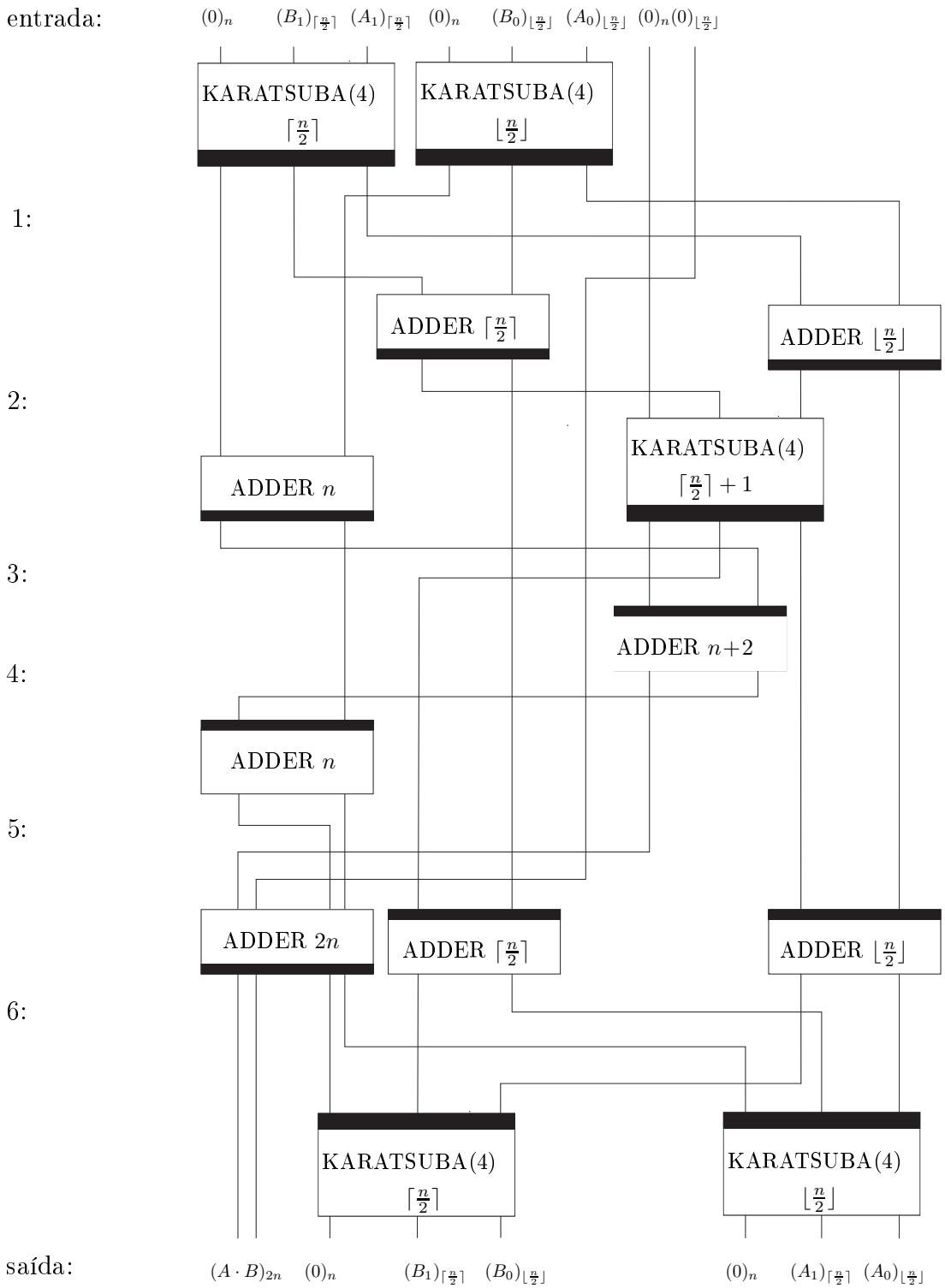

Figura 3.15: KARATSUBA(4) Procedimento recursivo para limpeza de lixo da versão seqüencial.

chamada recursiva, ao contrário do que ocorre nas outras versões, pode utilizar os bits extras das outras chamadas recursivas do mesmo nível. Sendo assim, o espaço utilizado, S_4 é

$$S_4(n) = \begin{cases} \Theta(1), & \text{se } n = 1 \\ S_4(\lceil \frac{n}{2} \rceil) + \Theta(n), & \text{se } n > 1. \end{cases}$$

Ou seja, o espaço utilizado por KARATSUBA(4) é linear.

O resumo da complexidade das quatro versões é dado na tabela 3.5.

esquema de limpeza do lixo	posição daso portas de multiplicação	
	paralelas	seqüenciais
primeiro método de Bennett	KARATSUBA (1)	KARATSUBA (2)
	$T_1(n) = \Theta(n^{\log 3})$	$T_2(n) = \Theta(n^{\log 3})$
	$S_1(n) = \Theta(n^{\log 3})$	$S_2(n) = \Theta(n^{\log 3})$
	$P_1(n) = \Theta(\log^2 n)$	$P_2(n) = \Theta(n^{\log 3})$
método recursivo	KARATSUBA (3)	KARATSUBA (4)
	$T_3(n) = \Theta(n^{\log 5})$	$T_4(n) = \Theta(n^{\log 5})$
	$S_3(n) = \Theta(n^{\log 3})$	$S_4(n) = \Theta(n)$
	$P_3(n) = \Theta(n)$	$P_4(n) = \Theta(n^{\log 5})$

Tabela 3.5: Complexidade das variações do algoritmo Karatsuba reversível

3.5.2 Algoritmo Karatsuba Reversível Modular

Supondo que se deseje obter uma multiplicação modular de dois operandos de n bits módulo N, onde N também é formado por n bits, isto pode ser feito através do circuito Karatsuba com uma pequena modificação.

A versão modular deste circuito pode ser construída simplesmente substituindo as portas ADDER por ADDER modulares. Desta forma, nenhum valor seria maior do que N. Na prática, esta substituição de portas não

precisa ser feita em todos os níveis da recursão, mas apenas naqueles em que os operandos são formados por n bits ou mais. Como a cada "chamada recursiva", o tamanho dos operandos é reduzido aproximadamente pela metade e, além disso, o resultado da multiplicação possui, no máximo, o dobro do tamanho dos operandos, a troca das portas ADDER por suas respectivas versões modulares precisa ser feita em até dois níveis de recursão. Após a primeira "chamada recursiva", o tamanho dos operandos é reduzido para $\lfloor \frac{n}{2} \rfloor$ e $\lceil \frac{n}{2} \rceil$. A multiplicação de operandos deste tamanho pode dar um resultado um pouco maior do que N. No próximo nível recursivo, cada parte do operando teria, no máximo, tamanho $\lceil \frac{n}{4} \rceil$, portanto não é mais necessário o uso de portas ADDER modulares substituindo as portas ADDER simples.

Não há alteração na ordem de complexidade com esta mudança de portas, pois a porta ADDER modular possui a mesma ordem de complexidade da porta ADDER simples. Ambas, dependendo da implementação, possuem complexidade de circuito e de espaço linear [46] em relação à quantidade de bits necessários para representar os parâmetros de entrada.

Com a modificação acima, o circuito Karatsuba reversível modular passa a executar a seguinte transformação:

$$ModKaratsuba(1) : (N, a, b, 0, 0) \rightarrow (N, a, b, 0, ab \bmod N),$$

onde a e b são os operandos e possuem no máximo n bits. Cada espaço entre as vírgulas representa um registrador. Todos eles possuem n bits com exceção do quarto registrador, que se refere aos bits extras necessários para a execução do circuito e que no final são novamente zerados. Este registrador é formado por $\Theta(n^{\log 3})$ bits auxiliares, caso seja usada a versão com $\Theta(n^{\log 3})$ portas (vide descrição do Karatsuba reversível). Os três primeiros registradores se referem aos parâmetros de entrada e o último mantém o resultado da multiplicação modular no final do processo.

Um inconveniente desta transformação é a necessidade de um registrador

inicialmente com o valor 0 para guardar o resultado. Isto também ocorria na versão não modular, mas naquele caso, o resultado possuía um tamanho praticamente o dobro do número de bits necessários para representar para cada operando e por isso não poderia estar contido em um dos registradores utilizados para os parâmetros de entrada. No caso modular, o resultado possui aproximadamente o mesmo tamanho dos operandos. A função associada a esta transformação é inversível no caso de os operandos serem co-primos com N. Assim, é possível armazenar o resultado em um dos registradores utilizados para definir os valores de entrada, de forma semelhante ao que foi feito para construir o circuito MULT-MOD2.

Capítulo 4

Resolução de Problemas de Otimização Discreta através de Computação Quântica

4.1 Definição do problema

No capítulo 2, comentamos a tese de Church-Turing [8, 45], que afirma que a classe de funções computáveis por uma máquina de Turing corresponde exatamente à classe de funções que naturalmente consideraríamos como computáveis por um algoritmo [34, seção 3.1]. Na prática, esta tese tem se confirmado após a demonstração da equivalência de diversos modelos de computabilidade como, por exemplo, máquina de Turing, Lambda cálculo [8, 21], funções recursivas, entre outros (*versão fraca*). Isto significa que os problemas que podem ser resolvidos em um determinado modelo, também o podem em qualquer outro igualmente geral. Mas há uma outra versão mais forte desta tese, que considera a equivalência dos modelos não apenas em termos de possibilidade da resolução de problemas, mas também em termos da ordem de complexidade de tempo (número de passos) com que estes problemas são resolvidos.

Com o advento da Computação Quântica, têm sido resolvidos problemas com ordem de complexidade inferior aos algoritmos de computação clássica [17, 40]. Por outro lado, todo procedimento clássico pode ser simulado em um computador quântico com a mesma ordem de complexidade, conforme foi mostrado no capítulo 2. Ou seja, toda função eficientemente computável classicamente também é eficientemente computável quanticamente, mas a recíproca não é verdadeira. Isto acontece, por exemplo, com o problema de Simon [43], que consiste em determinar se uma função $f : \{0,1\}^n \rightarrow \{0,1\}^n$, dada como uma caixa-preta, é uma função bijetora ou é uma função dois-para-um (se $\exists s \in \{0,1\}^n$, tal que $\forall x, x' \in \{0,1\}^n, f(x) = f(x') \leftrightarrow x = x' \oplus s$), onde $\oplus$ é o operador XOR aplicado bit-a-bit, neste último caso determina-se também s. Este problema pode ser resolvido classicamente apenas em tempo exponencial, enquanto que existem algoritmos quânticos exatos polinomiais para resolvê-lo [7, 33].

Na seqüência do texto, quando não for feita a distinção entre o modelo clássico e o modelo quântico, fica subentendido que estamos nos referindo ao modelo quântico, que é mais geral.

Neste texto, apresentamos alguns algoritmos quânticos para problemas de busca e otimização. Vamos formalizar estes problemas com as seguintes definições.

Definição 1 (Função Eficientemente Computável). *Dada uma função $f : X \rightarrow Y$ com domínio X e contra-domínio Y conhecidos, dizemos que f é uma função eficientemente computável em um determinado modelo computacional, se existir algum procedimento conhecido que, $\forall x \in X$, retorne o valor $f(x)$ em um número polinomial de passos sobre n neste modelo, onde $n = \lceil \log |X| \rceil$ é o número de bits necessários para representar o tamanho do domínio. Chamaremos de $P_f(n)$ a ordem de complexidade do procedimento que calcula f para o pior caso de uma entrada qualquer de tamanho*

n. Algumas vezes será utilizada a notação $P_f(N)$, onde $N = |X|$.

Definição 2 (Problema de Busca). *Dados uma função $f : X \to Y$ eficientemente computável e um valor $k \in Y$, chamaremos de problema de busca associado a uma função, o problema de encontrar um valor x, tal que $f(x) = k$.*

Definição 3 (Problema de Otimização). *Dada uma função $f : X \to Y$ eficientemente computável, chamaremos de problema de otimização, o problema de encontrar um valor x, tal que $\forall x' \in X$, $f(x) \leq f(x')$ ou $f(x) \geq f(x')$. No primeiro caso, o problema é de minimização e no seguinte, de maximização.*

Nas definições dos problemas de busca e otimização, a função f funciona como uma função de certificação. A rigor, não é necessário que f seja eficientemente computável, mas para efeitos práticos, estamos interessados nos problemas em que esta certificação pode ser feita em tempo polinomial, como é o caso, por exemplo, dos problemas NP-completos.

Na seção 4.2, é apresentado o algoritmo de Grover [17], idealizado para encontrar um determinado elemento em uma lista não-ordenada, mas que pode ser usado para resolver qualquer problema de busca [1, 34]. Considerando que N é o tamanho do domínio da função f associada ao problema e t é o número de valores que satisfazem ao critério de busca, o algoritmo possui complexidade de tempo $\Theta(\sqrt{N/t}) \cdot P_f(N)$, caso t seja conhecido. Caso contrário, apenas após $\Theta(\sqrt{N}) \cdot P_f(N)$ passos pode-se finalizar o algoritmo e medir o resultado. Na seção 4.3, é descrito o algoritmo BBHT98 [6], baseado no algoritmo de Grover, que resolve o problema de busca em $\Theta(\sqrt{N/t}) \cdot P_f(N)$, mesmo que t seja desconhecido. Esta ordem de complexidade é o limite inferior para o problema, se não for conhecida mais nenhuma propriedade da função f [48]. Este algoritmo será a base para os algoritmos de otimização que serão comentados adiante. Na seção 4.4, mostramos como construir

um operador, denominado Oráculo Desigualdade, que substitui o oráculo de Grover nos algoritmos de busca para possibilitá-los a resolver o problema de otimização. Na seção 4.5, é mostrado o algoritmo quântico de Christoph Dürr e Peter Høyer (DH96) [12], para encontrar o menor (ou maior) valor em uma lista não-ordenada. De acordo com a própria descrição do algoritmo, ele realiza $22,5\sqrt{N}$ consultas ao oráculo em uma lista com N elementos, com pelo menos 50% de sucesso. Mostramos, ainda nesta seção, como adaptar este algoritmo para resolver o problema de otimização, usando o Oráculo Desigualdade. Com mesma chance de sucesso, esta versão, alterando também um parâmetro do DH96 original, possui complexidade de tempo $11,48\sqrt{N} \cdot P_f(N)$, onde N é o tamanho do domínio da função f. A ordem de complexidade deste procedimento é o limite inferior do problema, pois encontrar o menor valor, que é desconhecido nos problemas de otimização, não pode ser mais fácil do que encontrar um determinado valor conhecido em uma seqüência não-ordenada nos problemas de busca, cuja complexidade é $\Theta(\sqrt{N}) \cdot P_f(N)$, quando há apenas uma única solução. O número de medidas, em ambas as versões (consultas à lista ou cálculo de f), é quadrático sobre n, o número de q-bits necessários para representar a entrada. Na seção 4.6, propomos um algoritmo quântico, que realiza $8,27\sqrt{N}$ consultas ao oráculo para chegar a uma solução correta com a igual chance de sucesso do algoritmo DH96. Em relação ao número de medidas (observações do sistema), o algoritmo proposto é linear sobre o número de q-bits necessários para representar a entrada. Por isso, denominamos o nosso algoritmo como *Algoritmo Medida-linear*. É possível efetuar apenas uma medição no final, em ambos algoritmos, mas para cada medição parcial de n q-bits, é necessário ter mais um registrador para armazenar este valor até o fim. Ou seja, o algoritmo DH96 passa a usar espaço cúbico e o algoritmo Medida-Linear passa a usar espaço quadrático.

4.2 Método de Grover para busca

Para o bom entendimento dos algoritmos quânticos de otimização discreta, inicialmente revisaremos o algoritmo de Grover (algoritmo 4.2.1).

O algoritmo de Grover foi desenvolvido inicialmente para a busca em uma lista não-ordenada [17], mas pode ser usado para resolver um problema de busca qualquer [1, 34]. Suponha que tenhamos um problema de busca definido por uma função f e um valor k. A partir de f, pode-se construir um procedimento reversível $U_f : |x\rangle|k\rangle \rightarrow |x\rangle|k \oplus f(x)\rangle$, onde $\oplus$ representa o operador XOR aplicado bit-a-bit. Este operador U_f possui complexidade $P_f(n)$, que é a mesma ordem de complexidade do procedimento irreversível clássico que calcula f [3].

Na formulação original do algoritmo de Grover [17, 36], é utilizada uma função (que chamaremos de $g(x, k)$, para não confundir com f anteriormente definida) que indica se x é solução do problema ou não. No caso da busca em uma lista, indica se k pertence à lista ou não. Em relação ao problema de busca anteriormente definido, esta função pode ser descrita como:

$$g(x,k) = \begin{cases} 1, & \text{se } f(x) = k \\ 0, & \text{se } f(x) \neq k. \end{cases}$$

A partir desta função, é definido um operador unitário $U_g : |x\rangle|k\rangle|-\rangle \mapsto (-1)^{g(x,k)}|x\rangle|k\rangle|-\rangle$, denominado de oráculo de Grover. O operador U_g pode ser construído a partir do operador U_f, de acordo com a figura 4.1. Cada linha representa um conjunto de q-bits. A linha superior (1^o registrador) é formada por $n = \lceil \log N \rceil$ q-bits, onde N é o tamanho do domínio de f. Este registrador é utilizado para armazenar os valores de entrada. A linha intermediária (2^o registrador) é o conjunto de q-bits que armazena a saída de f, e a linha inferior é formada por um único q-bit auxiliar. É possível construir o operador (e todo o circuito de Grover) sem este q-bit auxiliar, mas sua presença facilita a compreensão do operador (e do circuito de Grover). Se

a entrada for um estado projetado, a porta Z controlada pelo 2^o registrador inverte a amplitude do estado, caso o valor do 2^o registrador seja igual a $|0\rangle$ na saída inferior do operador U_f. A porta Z controlada pode ser implementada através de uma porta Toffoli generalizada com o q-bit alvo no estado $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. São omitidos outros q-bits auxiliares cujos valores de saída são iguais aos de entrada.

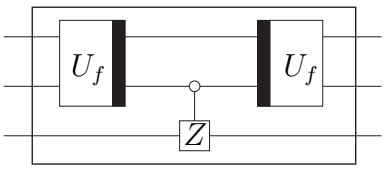

Figura 4.1: Circuito que implementa U_g a partir de U_f.

4.2.1 Descrição do algoritmo

O procedimento geral do algoritmo de Grover é mostrado no algoritmo 4.2.1. O símbolo $\lfloor x \rceil$ significa o inteiro mais próximo de x.

Algoritmo 4.2.1 Algoritmo de Grover

1) Preparação do estado inicial: $|\psi_e\rangle = |0\rangle_n|k\rangle_n|1\rangle_1$
2) $|\psi_1\rangle$: Superposição de todos os estados da base computacional, no primeiro registrador, necessários para representar o domínio de f e aplicação de Hadamard no último registrador sobre o estado $|\psi_e\rangle$.
3) **para** $j \leftarrow 1$ até $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rceil$ **faça**
4) Aplicação do Oráculo: $|\psi_{2j}\rangle = U_g|\psi_{(2j-1)}\rangle$
5) Inversão sobre a média: $|\psi_{2j+1}\rangle = (2|\psi_1\rangle\langle\psi_1| - I)|\psi_{2j}\rangle$
6) **fim para**
7) Observe o estado do primeiro registrador

O oráculo, seguido pela inversão sobre a média, é conhecido como operador de Grover.

Quando a dimensão do domínio de f é potência de 2, a superposição de todas as entradas possíveis de f, passo 2 do algoritmo 4.2.1, pode ser feita através do uso de portas Hadamard em cada linha do 1^o registrador, conforme comentado na seção 2.3. Caso não seja potência de 2, deve-se utilizar a Transformada de Fourier Quântica como descrito em [29].

A relação entre a chance de sucesso do algoritmo e o número de repetições do operador de Grover é dado pela proposição 2.

Proposição 2. *[34, seção 6.1.3] Dada uma lista L com N elementos, dos quais $t \leq \frac{N}{2}$ satisfazem a condição de busca do oráculo, a probabilidade de medir um elemento procurado após m iterações do oráculo, sem medição intermediária, é*

$$P_m = \text{sen}^2\left(\frac{2m+1}{2}\theta\right),$$

onde $\theta = \text{arcsen}(2\sqrt{t(N-t)}/N)$.

A proposição 3 indica a quantidade de iterações necessárias para maximizar a probabilidade de sucesso.

Proposição 3. *[34, eq.6.17] Dada uma lista L com N elementos, dos quais $t \leq \frac{N}{2}$ satisfazem a condição de busca do oráculo, um limite superior para a quantidade estimada de execuções do oráculo para obter um elemento marcado, sem nenhuma medição intermediária, é*

$$m \leq \left\lceil \frac{\pi}{4}\sqrt{\frac{N}{t}} \right\rceil.$$

4.2.2 Aplicação do algoritmo de Grover ao problema das Equações Binárias Quadráticas

Vejamos agora o funcionamento do algoritmo de Grover aplicado a um problema concreto. Seja a função de otimização:

$$f(x) = x^2 \bmod \beta, \text{ com } 0 \leq x \leq \gamma, \tag{4.1}$$

onde β e γ são números inteiros.

Dados f, γ e um valor α, encontrar um valor $x \in \{0, \ldots, \gamma\}$, tal que $f(x) = \alpha$, é denominado de resolução de Equações Binárias Quadráticas (EBQ). Este problema é NP-completo [14, 30].

Tomemos, por exemplo, $f(x) = x^2 \bmod 63$ com $\gamma = 15$ e $\alpha = 37$. Isto significa, que se deseja saber se existe $x \in [0, 15]$, tal que $f(x) = 37$. Pela tabela 4.1, podemos verificar que o único elemento do domínio de f que satisfaz a condição de busca é $x = 10$.

x	0	1	2	3	4	5	6	7	8	9	<u>10</u>	11	12	13	14	15
$f(x)$	0	1	4	9	16	25	36	49	1	18	<u>37</u>	58	18	43	7	36

Tabela 4.1: tabela com as imagens de $f(x) = x^2 \bmod 63$ para $x < 16$.

Um possível operador U_f associado ao problema é $U_f : |x\rangle|\alpha\rangle \rightarrow |x\rangle|\alpha \oplus (x^2 \bmod 63)\rangle$. Este operador é facilmente implementável usando um operador multiplicação modular que possui complexidade $O(n^{\log 3})$, onde $n = \log \beta$, usando o algoritmo Karatsuba reversível modular, descrito no capítulo 3. Não se pode utilizar o algoritmo de multiplicação modular, descrito por Vedral, Barenco e Ekert [46], pois este operador permite superposição de estados em apenas um dos operandos. O operador U_g é construído de acordo com a figura 4.1, com o segundo registrador inicialmente no estado $|37\rangle$. É possível projetar o operador U_f de outras formas, como, por exemplo, $U_f : |x\rangle|\alpha\rangle \rightarrow |x\rangle|(x^2 - \alpha) \bmod 63\rangle$. Em qualquer uma destas formas de implementar U_f, após uma aplicação da mesma, o segundo registrador terá o valor *zero* para os estados da base computacional (e apenas para estes) que forem solução do problema. Em ambas as formas de implementação de U_f, o circuito projetado para a busca de um valor α_1 não precisa ser modificado para a pesquisa de um novo valor α_2. O que difere é o estado inicial do sistema que, em vez de ser $|\psi_e\rangle = |0\rangle|\alpha_1\rangle|1\rangle$, passa a ser $|\psi_e\rangle = |0\rangle|\alpha_2\rangle|1\rangle$.

A complexidade de portas de U_f é $O(n^{\log 3})$, definida pela ordem de complexidade da multiplicação modular. O operador U_g possui mesma ordem de complexidade de U_f.

Considerando que as únicas entradas válidas são as menores do que γ, o número de aplicações necessárias de U_g no algoritmo 4.2.1 para resolver a função do problema que estamos tratando é $\lfloor \frac{\pi}{4}\sqrt{\gamma} \rceil$. No exemplo considerado, a quantidade de entradas é igual a 16, logo são necessárias 3 iterações do algoritmo.

Como todas as entradas da função f são menores do que 16 e as saídas possíveis são menores do que 64, os três registradores principais terão 4, 6 e 1 q-bits, respectivamente.

O estado inicial é

$$|\psi_e\rangle = |0\rangle_4|37\rangle_6|1\rangle_1$$

Fazendo a superposição de todos os estados da base computacional no primeiro e terceiro registradores (passo 2 do algoritmo 4.2.1), tem-se que:

$$|\psi_1\rangle = H^{\otimes 4}|0\rangle_4|37\rangle_6 H|1\rangle_1 = \frac{1}{4}\sum_{x=0}^{15}|x\rangle_4|37\rangle_6|-\rangle.$$

Aplicando o operador U_g, o estado do sistema passa a ser:

$$\begin{aligned}|\psi_2\rangle \;=\; & \tfrac{1}{4}(|0\rangle_4|37\rangle_6|-\rangle + |1\rangle_4|37\rangle_6|-\rangle + |2\rangle_4|37\rangle_6|-\rangle + |3\rangle_4|37\rangle_6|-\rangle + \\ & |4\rangle_4|37\rangle_6|-\rangle + |5\rangle_4|37\rangle_6|-\rangle + |6\rangle_4|37\rangle_6|-\rangle + |7\rangle_4|37\rangle_6|-\rangle + \\ & |8\rangle_4|37\rangle_6|-\rangle + |9\rangle_4|37\rangle_6|-\rangle - |\mathbf{10}\rangle_\mathbf{4}|\mathbf{37}\rangle_\mathbf{6}|-\rangle + |11\rangle_4|37\rangle_6|-\rangle + \\ & |12\rangle_4|37\rangle_6|-\rangle + |13\rangle_4|37\rangle_6|-\rangle + |14\rangle_4|37\rangle_6|-\rangle + |15\rangle_4|37\rangle_6|-\rangle).\end{aligned}$$

Apenas o estado $|10\rangle_4|37\rangle_6|-\rangle$ tem a amplitude invertida, pois $f(10) = 37$.

Aparentemente, o segundo registrador não tem seu estado alterado após aplicação de U_g, mas, na realidade, há uma mudança durante a aplicação de U_g e, em seguida, é restituído o valor original. Como

$$U_g = (U_f^\dagger \otimes I_1)(I_n \otimes C^n Z)(U_f \otimes I_1),$$

então,

$$U_g|x\rangle|k\rangle|-\rangle = (U_f^\dagger \otimes I_1)(I_n \otimes C^nZ)|x\rangle|k \oplus f(x)\rangle|-\rangle,$$

onde C^nZ é a porta Z controlada negativamente e I_n é o operador identidade aplicados em n q-bits. Isto implica em que, após a execução de U_f, o segundo registrador passa a guardar $k \oplus f(x)$ para cada valor de x. Aplicando em seguida a porta Z controlada, a amplitude dos estados, nos quais o valor no segundo registrador é igual a zero, é invertida. No nosso exemplo, isto acontece apenas com o estado $|10\rangle|0\rangle|-\rangle$, que passa a ser $-|10\rangle|0\rangle|-\rangle$, já que $37 \oplus f(10) = 0$. Aplicando U_f de forma reversa (último operador de U_g), o valor no segundo registrador passa a ser novamente 37 para todas as entradas. O sinal da amplitude permanece negativo para o estado cujo valor no primeiro registrador é 10 e positivo para os demais estados.

O estado $|\psi_2\rangle$ pode ser reescrito como:

$$|\psi_2\rangle = \frac{1}{4}\sum_{x=0}^{15}|x\rangle_4|37\rangle_6|-\rangle - \frac{2}{4}|10\rangle_4|37\rangle_6|-\rangle = |\psi_1\rangle - \frac{2}{4}|10\rangle_4|37\rangle_6|-\rangle.$$

Calculando $\langle\psi_2|\psi_1\rangle$, tem-se:

$$\langle\psi_2|\psi_1\rangle = \frac{1}{16}(16-2) = \frac{7}{8}.$$

Aplicando a inversão sobre a média, o estado global passa a ser:

$$\begin{aligned}
|\psi_3\rangle &= (2|\psi_1\rangle\langle\psi_1| - I)|\psi_2\rangle \\
&= (2\langle\psi_1|\psi_2\rangle|\psi_1\rangle - |\psi_2\rangle = \tfrac{14}{8}|\psi_1\rangle - |\psi_2\rangle \\
&= \tfrac{7}{4}|\psi_1\rangle - |\psi_1\rangle + \tfrac{1}{2}|10\rangle_4|37\rangle_6|-\rangle \\
&= \tfrac{3}{16}\textstyle\sum_{x=0}^{15}|x\rangle_4|37\rangle_6|-\rangle + \tfrac{1}{2}|10\rangle_4|37\rangle_6|-\rangle \\
&= \tfrac{1}{4}(\tfrac{3}{4}\textstyle\sum_{x=0,x\neq 10}^{15}|x\rangle_4|37\rangle_6|-\rangle + \tfrac{11}{4}|10\rangle_4|37\rangle_6|-\rangle).
\end{aligned}$$

A amplitude do estado procurado passa de $\frac{1}{16}$ para $\frac{11}{16}$, depois da primeira rodada do algoritmo de Grover. Com isso, a chance deste elemento ser lido é $|\langle\psi_3|10, 37, -\rangle|^2 = (\frac{11}{16})^2 \cong 47,26\%$.

Na segunda rodada, faz-se novamente a inversão do elemento procurado no algoritmo do Grover, obtendo-se

$$|\psi_4\rangle = \frac{1}{4}\left(\frac{3}{4}\sum_{x=0,x\neq 10}^{15}|x\rangle_4|37\rangle_6|-\rangle - \frac{11}{4}|10\rangle_4|37\rangle_6|-\rangle\right).$$

Como $\langle\psi_4|\psi_1\rangle = \frac{1}{4}(\frac{3}{16}15 - \frac{11}{16}) = \frac{34}{64} = \frac{17}{32}$, invertendo o estado anterior sobre a média, tem-se:

$$\begin{aligned}
|\psi_5\rangle &= (2|\psi_1\rangle\langle\psi_1| - I)|\psi_4\rangle \\
&= (2\langle\Psi_1|\Psi_4\rangle|\Psi_1\rangle - |\Psi_4\rangle = \tfrac{17}{16}|\psi_1\rangle - |\psi_4\rangle \\
&= \tfrac{5}{64}\textstyle\sum_{x=0,x\neq 10}^{15}|x\rangle_4|37\rangle_6|-\rangle + \tfrac{61}{64}|10\rangle_4|37\rangle_6|-\rangle).
\end{aligned}$$

A chance de medição do elemento procurado depois desta rodada é, aproximadamente, 90,8%.

Depois da terceira rodada do Grover, o produto escalar $\langle\psi_6|\psi_1\rangle = \frac{7}{128}$ e a amplitude do estado procurado passa para $\frac{251}{256}$. Com isso, a probabilidade deste elemento ser lido é de aproximadamente 96%.

Não adianta aplicar outras rodadas, pois a amplitude da projeção do estado $|\psi_8\rangle = U_g|\psi_7\rangle$ sobre o estado $|\psi_1\rangle$ é negativa ($-\frac{13}{256}$), assim, a amplitude do estado procurado começa a diminuir caso sejam aplicadas mais iterações do algoritmo.

4.3 Algoritmo BBHT98

Conforme foi dito inicialmente, dado um problema de busca associado a uma função f e um valor k, o algoritmo de Grover é capaz de retornar uma das t soluções corretas depois de $\Theta(\sqrt{\frac{N}{t}}) \cdot P_f(N)$ passos, caso t seja conhecido. Mas, normalmente, não se conhece este valor de antemão. Como não é possível fazer medições intermediárias sem alterar o estado do sistema, não se pode finalizar o algoritmo no momento certo. O operador de Grover faz uma rotação de $\theta = arcsen(\frac{t}{N})$ no espaço bidimensional definido pelo estado

formado pela superposição de todos os estados da base computacional com mesmas amplitudes e o estado formado pela superposição de todas as soluções com mesmas amplitudes [34, seção 6.1.3]. Por isso, o comportamento do algoritmo é cíclico. Depois que o estado do sistema aproxima ao máximo do estado formado pela superposição dos estados com as soluções, ele volta a se afastar no mesmo rítmo. Isto significa que se for executado o dobro de iterações necessárias para se chegar ao estado composto pelas soluções, a probabilidade de se obter uma solução volta a ser próxima da probabilidade do estado formado pela superposição de todos os estados da base computacional, revelando pouco sobre uma solução correta. Caso haja 4 soluções, tem-se que o ângulo de rotação é $\theta \approx \frac{\pi}{2}$ após $\left\lfloor \frac{\pi}{4}\sqrt{\frac{N}{4}} \right\rfloor$ iterações. Sendo assim, se executarmos o dobro de iterações, ou seja, $\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor$, que é o número de iterações necessárias para encontrar o elemento procurado quando este é único, a probabilidade de sucesso é $\text{sen}^2 2\theta$, que é próximo de zero.

4.3.1 Descrição do algoritmo

Michel Boyer, Gilles Brassard, Peter Höyer e Alain Tapp resolvem este problema, alternando a execução do operador de Grover e a medição dos registradores, de forma controlada [6]. O número de vezes que o operador de Grover é executado entre duas medições consecutivas aumenta em uma progressão geométrica, de razão λ. No algoritmo 4.3.1, é apresentada a descrição do procedimento desenvolvido por eles.

Da forma como está descrito o algoritmo 4.3.1, seguindo a descrição original em [6], falta definir um limite superior para o número de iterações, para o caso de o elemento não existir.

Segundo a descrição do algoritmo, o parâmetro λ pode ser escolhido no intervalo $\left(1, \frac{4}{3}\right)$. Como será visto adiante, este parâmetro pode ter valores maiores, mantendo a complexidade do algoritmo em $O\left(\sqrt{\frac{N}{t}}\right)$, onde t é a

Algoritmo 4.3.1 Algoritmo BBHT98

1: Inicialize $m \leftarrow 1$ e $\lambda \leftarrow \frac{8}{7}$ $\{serve\ qualquer\ valor\ 1 < \lambda < \frac{4}{3}\}$

2: **repita**

3: Escolha aleatoriamente, com distribuição uniforme, um inteiro j, tal que $0 \leq j < m$.

4) Inicialize o sistema no estado $|0\rangle|k\rangle|1\rangle$, onde k é o elemento procurado.

5) Faça superposição de todos os estados da base computacional e aplique uma porta H no último registrador.

6) Aplique j iterações do operador de Grover.

7) Meça o primeiro registrador, chamando de i o resultado.

8: $m \leftarrow$ mínimo$(\lambda m, \sqrt{N})$.

9: **até que** i-ésima posição contenha o valor k.

10: Retorne o último valor medido caso tenha encontrado.

quantidade de soluções.

Os passos, cuja numeração está sucedida por dois pontos, em vez de parêntese, podem ser executados classicamente, mostrando que, a execução propriamente dita no computador quântico está compreendida entre os passos 4 e 7 do algoritmo.

4.3.2 Análise da corretude e complexidade

Considerando que há t soluções para o problema de busca, a análise do comportamento do BBHT98 será dividida em três casos:

1. $t = 0$,
2. $1 \leq t \leq \frac{N}{2}$,
3. $\frac{N}{2} < t \leq N$.

No primeiro caso, não existe o elemento procurado. O algoritmo 4.3.1 executa um número limite de iterações conforme comentário feito na sub-seção anterior, retornando um valor qualquer. Este limite deve ser proporcional a $\sqrt{N}$, para o algoritmo não terminar prematuramente nos casos seguintes.

O último caso ocorre quando pelo menos 50% dos elementos satisfazem à condição de busca. Nesta situação, não se consegue ampliar muito a amplitude dos estados que satisfazem à condição de procura. No entanto, como a quantidade destes estados é grande, a probabilidade de faha ao medir algum deles é baixa, da ordem de $(\frac{1}{2})^k$, onde k é o número de medições do algoritmo 4.3.1. Assim, após uma única medição já temos 50% de chance de obter o resultado. A quantidade de medições esperadas é menor ou igual a duas.

O segundo caso é justamente o caso resolvido pelo algoritmo propriamente dito. A seguir, é mostrado o tempo estimado para este caso.

Proposição 4. *[6] A probabilidade de se obter um elemento marcado pelo oráculo executando, sem medições intermediárias, $j < m$ vezes o operador de Grover, onde j é escolhido com distribuição uniforme é*

$$P_m = \frac{1}{2} - \frac{sen(4m\theta)}{4msen(2\theta)},$$

onde $sen^2\theta = t/N$. Em particular, $P_m \geq 1/4$, quando $m \geq 1/sen(2\theta)$.

Proposição 5. *Pela proposição 4, pode-se concluir que, para qualquer valor $m \geq \frac{N}{2\sqrt{(N-t)t}}$, onde m é o limite da quantidade de execuções do oráculo antes de cada medição, a probabilidade de sucesso é, no mínimo, 25%.*

Lema 6. *Dado um algoritmo quântico de busca que faça medições após $j < m$ passos, onde j é escolhido com distribuição uniforme e m cresce em uma progressão geométrica de razão λ, a quantidade total esperada de repetições do oráculo para encontrar um determinado elemento, quando há t soluções,*

é, no máximo,

$$S = \frac{1}{2}\frac{\lambda^4}{\lambda - 1}m_0,$$

onde $m_0 = \left\lceil \frac{N}{2\sqrt{(N-t)t}} \right\rceil$. *Em particular, se* $\lambda = \frac{4}{3}$, *então,* $S = 4,75m_0$.

Demonstração. Sendo $m_0 = \lceil 1/\text{sen}(2\theta) \rceil$, então, quando $m = m_0$, pela proposição 4, o algoritmo retornará o valor procurado com pelo menos 25% de chance de sucesso.

O número médio esperado de iterações até atingir à faixa em que $P_m \geq 1/4$ é

$$S_a = \frac{1}{2}\sum_{i=0}^{\lceil \log_\lambda m_0 \rceil} \lambda^i. \tag{4.2}$$

Dado que a fórmula para a soma de uma progressão geométrica finita de razão λ, começando de a_1 com n termos, é

$$S_n = \frac{a_1\lambda^n - a_1}{\lambda - 1}. \tag{4.3}$$

E, utilizando ambas as fórmulas (4.2) e (4.3), temos que:

$$S_a = \frac{1}{2}\frac{\lambda^{\lceil \log_\lambda m_0 \rceil + 1} - 1}{\lambda - 1} = \frac{1}{2}\frac{\lambda m_0 - 1}{\lambda - 1}. \tag{4.4}$$

O número esperado de execuções do oráculo no algoritmo BBHT98, até atingir a faixa em que cada medição passa a ter pelo menos 25% de chance de sucesso, é

$$S_a < \frac{1}{2}\frac{\lambda}{\lambda - 1}m_0. \tag{4.5}$$

Após o algoritmo ter entrado na faixa crítica, cada nova medição possui pelo menos $\frac{1}{4}$ de chance de sucesso. Sendo assim, o número esperado de novas medições para se obter o resultado é menor ou igual a 4.

O número total de execuções do oráculo passa a ser

$$S = \frac{1}{2}\sum_{i=0}^{\lceil \log_\lambda m_0 \rceil + 3} \lambda^i. \tag{4.6}$$

Utilizando a fórmula para soma de uma progressão geométrica finita, tem-se que

$$S = \frac{1}{2}\frac{\lambda^{\lceil \log_\lambda m_0 \rceil + 4} - 1}{\lambda - 1} = \frac{1}{2}\frac{\lambda^4 m_0 - 1}{\lambda - 1}. \tag{4.7}$$

Podemos concluir que

$$S < \frac{1}{2}\frac{\lambda^4}{\lambda - 1} m_0. \tag{4.8}$$

Derivando e igualando a zero a expressão acima, vemos que S possui valor mínimo para $\lambda = \frac{4}{3}$. Para $\lambda = \frac{4}{3}$, tem-se que $S < 4,75 m_0$. □

Teorema 7. *Dada uma lista com N elementos, dos quais t são soluções, a quantidade total estimada de execuções do operador de Grover no algoritmo BBHT98 é, no máximo, $3,36\sqrt{\frac{N}{t}}$. Se há apenas uma solução, esta quantidade pode ser reduzida para $2,38\sqrt{N}$.*

Demonstração. Considerando que há t soluções e $\frac{1}{\text{sen}(2\theta)} = \frac{N}{2\sqrt{(N-t)t}}$, tem-se, $m_0 = \left\lceil \frac{N}{2\sqrt{(N-t)t}} \right\rceil = \left\lceil \frac{1}{2}\sqrt{\frac{N}{N-t}}\sqrt{\frac{N}{t}} \right\rceil$.

Se $t = 1$, $m_0 \approx \frac{1}{2}\sqrt{N}$ e, pelo lema 6, como $S = 4,75 m_0$, então, a quantidade de iterações esperada é menor do que $2,38\sqrt{N}$.

Considerando $1 \leq t \leq \frac{N}{2}$ pode-se verificar facilmente que

$$m_0 \leq \frac{1}{\sqrt{2}}\sqrt{\frac{N}{t}}. \tag{4.9}$$

O que indica, pelo lema 6, uma quantidade de iterações esperada menor do que $3,36\sqrt{\frac{N}{t}}$.

O caso em que $t > \frac{N}{2}$ é resolvido em tempo constante conforme foi dito anteriormente. □

Pelo teorema 7, pode-se concluir, que o tempo estimado do algoritmo será $O(m_0) = O(\sqrt{N/t})$.

4.3.3 Exemplo

Consideremos o exemplo dado na sub-seção 4.2.2 da seção anterior: $f(x) = x^2 \ mod \ 63$ com $0 \leq x \leq 15$. Desejamos saber se existe x, tal que $f(x) = 37$.

Usando o algoritmo de Grover, são necessárias 3 iterações. No algoritmo BBHT98, a quantidade total de aplicações do operador de Grover não pode ser determinada *a priori*, pois depende das medições intermediárias ou de um limite pré-definido. O valor m, no algoritmo 4.3.1, limita a quantidade máxima de iterações. Nas seis primeiras rodadas, $m < 2$. Sendo assim, nestas primeiras iterações, será feita apenas uma aplicação do operador de Grover antes de cada medida. Para o exemplo dado, conforme foi dito na sub-seção 4.2.2, a chance de se obter o estado correto após uma aplicação do operador é $47,26\%$. Após a segunda rodada, a probabilidade será $0,4726 + 0,5274 \cdot 0,4726 = 72,18\%$. Após a terceira rodada, será $85,33\%$. Após a quinta rodada, a probabilidade será $95,91\%$, ou seja, próxima da probabilidade de se obter a solução usando o algoritmo de Grover com 3 iterações.

Se, em vez de procurarmos $f(x) = 37$, caso em que há apenas uma solução, procurássemos x, tal que $f(x) = 18$, caso em que há duas soluções, a quantidade de iterações do algoritmo de Grover seria $\left\lfloor \frac{\pi}{4}\sqrt{\frac{16}{2}} \right\rfloor = 2$, atingindo $94,53\%$ de chance de sucesso. Com o mesmo número de chamadas do operador de Grover, o algoritmo BBHT98 atinge $95,21\%$.

4.4 Oráculo Desigualdade

Para os problemas de otimização, onde se procura o menor (ou maior) elemento de um conjunto, não se pode utilizar o oráculo de Grover, pois este inverte os estados da base computacional associados a um determinado valor k. É necessário um oráculo que inverta os estados relativos a valores menores (ou maiores) do que k, dependendo se o problema for de minimização ou de maximização. Nesta seção, mostramos como pode ser feita a implementação deste operador, que denominaremos de *Oráculo Desigualdade*.

4.4.1 Descrição do operador

Dados $f : X \to Y$ uma função eficientemente computável e um valor $k \in Y$, é definida uma função $g'(x,k)$ conforme descrição a seguir:

$$g'(x,k) = \begin{cases} 1, & \text{se } f(x) \leq k \\ 0, & \text{se } f(x) > k. \end{cases}$$

A partir desta função, é definido um operador quântico $U_{g'} : |x\rangle|k\rangle|-\rangle \mapsto (-1)^{g'(x,k)}|x\rangle|k\rangle|-\rangle$, que é justamente o Oráculo Desigualdade. Assim, é possível inverter a amplitude dos estados da base computacional, cuja imagem seja menor do que um determinado valor.

Se $k = 2^j$, para um j inteiro qualquer, com o operador $U_f : |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$ e usando mais um registrador com o valor $|-\rangle$, pode-se construir o Oráculo Desigualdade $U_{g'}$ através de uma porta Toffoli generalizada cujos controles estão nos q-bits mais significativos do segundo registrador e o alvo no q-bit do terceiro registrador, deixando os j bits menos significativos livres. O diagrama para este oráculo pode ser visto na figura 4.2.

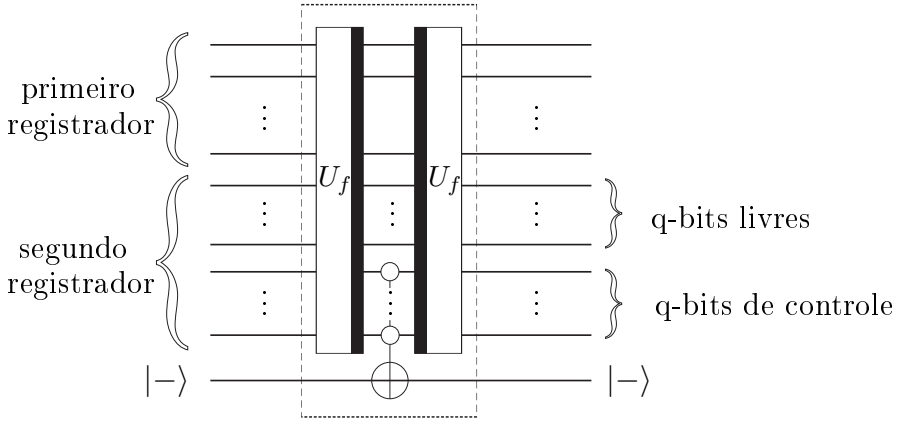

Figura 4.2: Circuito que implementa o Oráculo Desigualdade a partir de U_f, com k sendo uma potência de 2. O primeiro registrador contém a entrada e o segundo registrador a saída da função f. Os $j = \log k$ q-bits menos significativos do segundo registrador estão representados na parte superior do mesmo.

Caso k não seja potência de 2, com mais um registrador auxiliar, pode-se construir o oráculo utilizando um circuito somador na ordem reversa.

A soma de dois operandos formados por n bits cada pode ter como resultado um valor formado por $n+1$ bits. Por isso, um circuito reversível para esta operação necessita de um bit extra com valor inicial zero para ajudar a guardar o resultado da adição, lembrando que acrescentar zeros como dígitos mais significativos de qualquer operando não altera o resultado da adição. A soma reversível pode ser feita mantendo o valor do primeiro operando no primeiro registrador e substituindo o segundo operando pelo resultado da soma de ambos no segundo registrador (incluindo o bit extra) conforme foi comentado no capítulo 3. Considerando que os operandos são sempre não-negativos, após uma soma, o valor no segundo registrador é sempre maior ou igual ao valor contido no primeiro registrador. Caso seja feita a soma na ordem reversa (como se fosse feito de trás para frente) sobre dois valores a e b, ambos de n bits, tem-se que o bit extra mantém o valor *zero* se $a \leq b$, e passa a ter o valor *um*, caso contrário. Isto acontece porque o resultado da soma reversa com entrada (a, b) é $(a, 2^{n+1} - (a - b))$ se $b \leq a$. Assim, um somador pode ser usado como um comparador de dois valores inteiros.

O circuito para o Oráculo Desigualdade, com k podendo assumir um valor qualquer, é dado na figura 4.3.

O custo computacional deste circuito é o dobro do custo de implementação de U_f mais o custo da aplicação de dois somadores que podem ser executados em tempo linear sobre a quantidade de q-bits. Na prática, o custo da soma é bem menor do que $P_f(n)$, portanto o custo computacional do circuito que implementa o Oráculo Desigualdade é $P_f(n)$.

Se o problema for de maximização, em vez de minimização, basta substituir a porta CNOT central da figura 4.3 controlada negativamente por outra porta CNOT controlada positivamente, ou seja, que altera o q-bit alvo se o q-bit de controle possuir estado $|1\rangle$. No caso anterior, seriam invertidas as

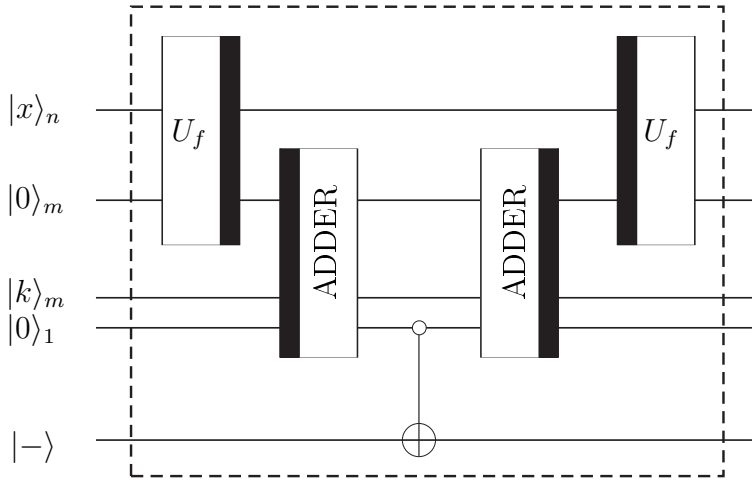

Figura 4.3: Circuito que implementa o Oráculo Desigualdade a partir de U_f, para k qualquer. n é o tamanho da entrada da função f e m é a quantidade de q-bits necessários para representar a saída de f. O segundo registrador do somador está dividido em duas partes: uma formada pelos q-bits necessários para representar k e outra formada por um único q-bit utilizado para indicar se o primeiro operando ($f(x)$) é maior do que o segundo (k).

amplitudes dos estados relativos a valores menores ou iguais a k, e com a mudança no circuito, serão invertidas as amplitudes dos estados relativos a valores maiores do que k.

4.4.2 Exemplo

Tomemos um exemplo parecido com o da seção 4.2.2 a fim de ilustrar melhor o funcionamento do operador Oráculo Desigualdade.

Voltemos ao caso das Equações Binárias Quadráticas. Seja $f(x) = x^2 \bmod 63$, com $0 \leq x \leq 15$ e $k = 37$. Deseja-se encontrar algum valor x tal que $f(x) \leq 37$.

Na tabela 4.2, estão indicados em negrito os valores que correspondem à condição desejada.

x	**0**	**1**	**2**	**3**	**4**	**5**	**6**	7	**8**	**9**	**10**	11	**12**	13	**14**	**15**
$f(x)$	0	1	4	9	16	25	36	49	1	18	37	58	18	43	7	36

Tabela 4.2: tabela com as soluções de $f(x) = x^2 \bmod 63 \leq 37$ com $x < 16$.

Podemos tomar $U_f : |x\rangle|\alpha\rangle \rightarrow |x\rangle|\alpha \oplus x^2 \bmod 63\rangle$.

Como k não é potência de 2, devemos usar o somador conforme o circuito da figura 4.4.

A seguir é feita, gradualmente, a evolução do sistema sobre uma superposição de todas as entradas.

O estado no qual é aplicado o operador é

$$|\psi_0\rangle = \frac{1}{4}\sum_{x=0}^{15} |x\rangle_4|0\rangle_6|37\rangle_6|0\rangle_1|-\rangle.$$

Em seguida, são aplicadas portas CNOT entre os q-bits do primeiro e segundo registradores obtendo:

$$|\psi_1\rangle = \frac{1}{4}\sum_{x=0}^{15} |x\rangle_4|x\rangle_4|0\rangle_2|37\rangle_6|0\rangle_1|-\rangle.$$

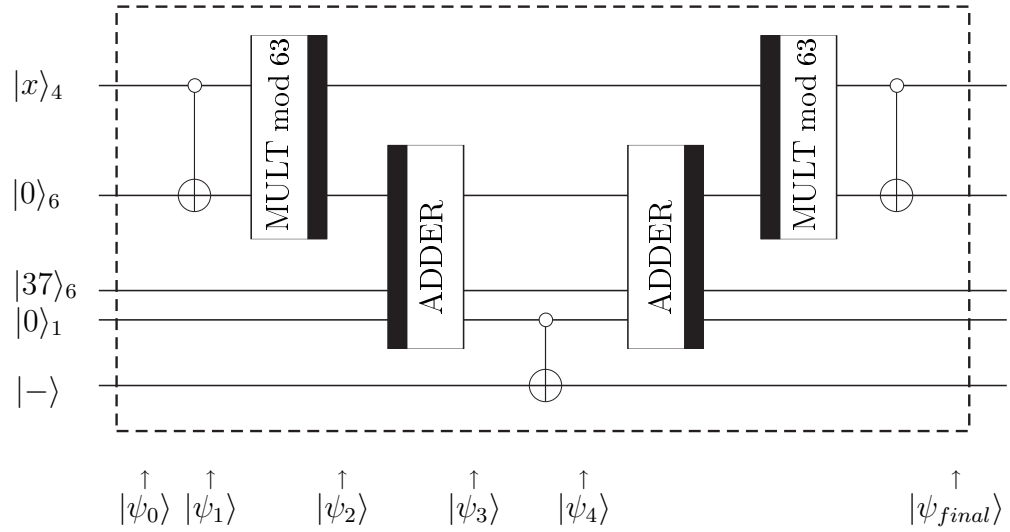

Figura 4.4: Circuito com a implementação do Oráculo Desigualdade para $f(x) = x^2 \bmod 63$, com $0 \leq x \leq 15$ e $k = 37$. As portas CNOT's no início e no fim do circuito indicam que o i-ésimo q-bit do primeiro registrador controla o i-ésimo q-bit do segundo registrador, onde $1 \leq i \leq 4$. Os dois q-bits mais significativos do segundo registrador não são controlados.

A seguir, é feita a multiplicação modular:

$$|\psi_2\rangle = \frac{1}{4}\sum_{x=0}^{15} |x\rangle_4 |x^2 \bmod 63\rangle_6 |37\rangle_6 |0\rangle_1 |-\rangle.$$

Este estado pode ser escrito como:

$$\begin{aligned} |\psi_2\rangle = \tfrac{1}{4}(\ & |0\rangle|0\rangle + |1\rangle|1\rangle + |2\rangle|4\rangle + |3\rangle|9\rangle + \\ & |4\rangle|16\rangle + |5\rangle|25\rangle + |6\rangle|36\rangle + |7\rangle|49\rangle + \\ & |8\rangle|1\rangle + |9\rangle|18\rangle + |10\rangle|37\rangle + |11\rangle|58\rangle + \\ & |12\rangle|18\rangle + |13\rangle|43\rangle + |14\rangle|7\rangle + |15\rangle|36\rangle\ )|37\rangle|0\rangle|-\rangle. \end{aligned}$$

O passo seguinte é a adição de forma reversa do segundo com o terceiro registrador. Representando junto as duas partes do segundo registrador da

porta ADDER, tem-se que o estado global do sistema passa a ser:

$$|\psi_3\rangle = \tfrac{1}{4}(\,|0\rangle|0\rangle|37\rangle + |1\rangle|1\rangle|36\rangle + |2\rangle|4\rangle|33\rangle + |3\rangle|9\rangle|28\rangle + \\ |4\rangle|16\rangle|21\rangle + |5\rangle|25\rangle|12\rangle + |6\rangle|36\rangle|1\rangle + \underline{|7\rangle|49\rangle|116\rangle} + \\ |8\rangle|1\rangle|36\rangle + |9\rangle|18\rangle|19\rangle + |10\rangle|37\rangle|0\rangle + \underline{|11\rangle|58\rangle|107\rangle} + \\ |12\rangle|18\rangle|19\rangle + \underline{|13\rangle|43\rangle|122\rangle} + |14\rangle|7\rangle|30\rangle + |15\rangle|36\rangle|1\rangle\,)|0\rangle|-\rangle.$$

Aplicando a porta CNOT no último q-bit do terceiro registrador, será invertida a amplitude dos estados menores do que 2^6. Após esta operação teremos:

$$|\psi_4\rangle = \tfrac{1}{4}(\,-|0\rangle|0\rangle|37\rangle - |1\rangle|1\rangle|36\rangle - |2\rangle|4\rangle|33\rangle - |3\rangle|9\rangle|28\rangle + \\ -|4\rangle|16\rangle|21\rangle - |5\rangle|25\rangle|12\rangle - |6\rangle|36\rangle|1\rangle + |7\rangle|49\rangle|116\rangle + \\ -|8\rangle|1\rangle|36\rangle - |9\rangle|18\rangle|19\rangle - |10\rangle|37\rangle|0\rangle + |11\rangle|58\rangle|107\rangle + \\ -|12\rangle|18\rangle|19\rangle + |13\rangle|43\rangle|122\rangle - |14\rangle|7\rangle|30\rangle - |15\rangle|36\rangle|1\rangle\,)|0\rangle|-\rangle.$$

Os operadores seguintes servem apenas para restaurar o valor original nos segundo e terceiro registradores.

O estado final global do sistema será:

$$|\psi_{final}\rangle = \tfrac{1}{4}(\,-|0\rangle - |1\rangle - |2\rangle - |3\rangle - |4\rangle - |5\rangle - |6\rangle + |7\rangle - |8\rangle + \\ -|9\rangle - |10\rangle + |11\rangle - |12\rangle + |13\rangle - |14\rangle - |15\rangle)|0\rangle_6|37\rangle_6|0\rangle_1|-\rangle.$$

Isto significa que foram invertidas as amplitudes dos estados desejados, ou seja, aqueles cuja imagem de f do valor no primeiro registrador é menor ou igual a 37.

4.5 Algoritmo DH96

Christoph Dürr e Peter Høyer propuseram, em 1996, um algoritmo quântico para encontrar o valor mínimo de uma lista não-ordenada [12]. Este algoritmo é baseado no algoritmo BBHT98 [6]. Considerando que a lista possui N elementos, o algoritmo possui complexidade $O(\sqrt{N})$. O procedimento geral é descrito na seção seguinte.

4.5.1 Descrição do algoritmo DH96 original

Algoritmo 4.5.1 Versão original do algoritmo DH96: encontrar o menor elemento dentro de uma lista não-ordenada T com N elementos

1: Escolha aleatoriamente, com distribuição uniforme, um valor inteiro x menor do que N.
2: **enquanto** número de passos $< \frac{45}{2}\sqrt{N} + \frac{14}{10}\log^2 N$ **faça**
3: Marque na lista todos os elementos j tais que $T[j] < T[x]$.
4) Execute o algoritmo BBHT98 procurando elementos marcados a partir do estado $\sum_j \frac{1}{\sqrt{N}}|j\rangle|x\rangle$.
5) Observe o primeiro registrador e guarde o resultado em x'.
6: **se** $T[x'] < T[x]$ **então**
7: $x \leftarrow x'$
8: **fim se**
9: **fim enquanto**
10: **devolva** x

Assim como o algoritmo de Grover, o algoritmo DH96 trabalha com inversões da amplitude: inversão da amplitude dos estados associados aos elementos marcados e inversão sobre a média.

A parte do algoritmo 4.5.1 que é executada em um computador quântico está compreendida entre os passos 4 e 5.

4.5.2 Descrição do Algoritmo DH96 adaptado para problemas de otimização combinatória

Com o uso do Oráculo Desigualdade, pode-se adaptar o algoritmo DH96 para resolver problemas de otimização discreta.

Na versão original, faz-se uma manipulação da lista, marcando os itens procurados, antes de cada execução do BBHT98. No caso do problema de otimização associado, o custo desta operação, marcar os itens procurados,

está embutido em cada chamada do oráculo. Isto altera a complexidade do algoritmo.

Outra questão importante é o fato de que o algoritmo BBHT98 utiliza o oráculo de Grover e, como foi comentado na seção anterior, este oráculo inverte apenas os estados associados a um determinado valor fixo. No caso dos problemas de otimização, é necessária a inversão de estados associados a valores que podem ser distintos, concretamente, os estados associados a valores menores do que um determinado parâmetro que varia nas diversas etapas do algoritmo. Isto pode ser feito utilizando o Oráculo Desigualdade, descrito na seção 4.4.

O algoritmo 4.5.2 descreve uma variação do algoritmo DH96 que propomos para resolver o problema de minimização de uma função $f : X \to Y$.

O valor de λ utilizado no algoritmo DH96 original é $8/7$. Isto não está explícito na descrição original do algoritmo, mas é utilizado este valor para definir o número necessário de operações para atingir 50% de chance de sucesso. Na descrição da nossa variação, foi utilizado $\lambda = 4/3$, baseado no lema 6 da página 75.

4.5.3 Exemplo

Para compreendermos melhor o funcionamento do algoritmo 4.5.2, daremos um exemplo. Para isso, transformaremos o problema de decisão EBQ, descrito na sub-seção 4.2.2, para um problema de otimização, através da inclusão do parâmetro α na função f, dado na expressão (4.1). O problema passa a ser minimizar a função

$$f(x) = x^2 - \alpha \bmod \beta, \text{ com } 0 \leq x \leq \gamma, \tag{4.10}$$

onde α, β e γ são números inteiros.

Este problema é NP-difícil, pois dua resolução implica na resolução do

Algoritmo 4.5.2 Adaptação do DH96 para problemas de otimização discreta

1: Escolha aleatoriamente, com distribuição uniforme, um valor x inteiro menor do que N.
2: $y \leftarrow f(x)$
3: $i \leftarrow 0$
4: **enquanto** $i < 11,48\sqrt{N}$ **faça**
5: Inicialize m e $\lambda \leftarrow \frac{4}{3}$.
6: **repita**
7: Escolha aleatoriamente, com distribuição uniforme, um inteiro positivo j, tal que $j < m$.
8) Prepare o sistema no estado inicial $|0\rangle_n|y\rangle_l|1\rangle_1$.
9) Faça a superposição de todas os estados relacionados com o domínio de f, no primeiro registrador, e aplique a porta H no terceiro registrador.
10) **para** j iterações **faça**
11) Aplique o Oráculo Desigualdade para inverter os estados cujas imagens sejam menores do que y.
12) Aplique a inversão sobre a média.
13) **fim para**
14) Observe o primeiro registrador e guarde em x' o valor medido.
15: $i \leftarrow i + j$
16: $m \leftarrow$ mínimo$(\lambda m, \sqrt{N})$
17: **até que** $(f(x') < y)$ ou $(i > 11,48\sqrt{N})$
18: **se** $f(x') < y$ **então**
19: $y \leftarrow f(x')$
20: $x \leftarrow x'$
21: **fim se**
22: **fim enquanto**
23: **devolva** x.

problema em 4.2.2, que é NP-completo.

Para manter o exemplo de forma similar ao dado anteriormente, tomemos $f(x) = x^2 - 38 \bmod 63$, com $0 \leq x \leq 15$. Desejamos saber qual é o mínimo global de f. Pela tabela 4.3, podemos verificar que o mínimo é *5*, dado por $f(13)$.

x	f(x)
0	25
1	26
2	29
3	34
4	41
5	50
6	61
7	11
8	26
9	43
10	62
11	20
12	43
13	5
14	32
15	61

Tabela 4.3: tabela com as soluções de $f(x) = x^2 - 38 \bmod 63$, com $x < 16$.

Neste caso, γ faz o papel de N no algoritmo, pois γ limita o tamanho da entrada.

Um operador para U_f associado ao problema é

$$U_f : |x\rangle|\alpha\rangle \rightarrow |x\rangle|(x^2 - \alpha) \bmod \beta\rangle.$$

Neste caso, o custo computacional de f é o custo de uma multiplicação modular, $O(n^{\log 3})$, se for usado o algoritmo Karatsuba modular reversível.

Como há medições durante a execução do algoritmo e a quantidade de aplicações do oráculo depende destas medições, cada execução pode ter um comportamento bem distinto das outras. O ideal seria mostrar todas as possíveis execuções, mas isto é inviável devido à grande quantidade de execuções possíveis. Por isso, mostraremos apenas uma execução possível com comportamento intermediário entre o melhor e o pior caso.

Para facilitar a visualização da evolução dos estados, é mostrado na tabela 4.4 os valores que caracterizam cada estado, com o valor correspondente da imagem de f e com a probabilidade de medição (quadrado do módulo da amplitude do estado). Os estados estão ordenados pela imagem da função .

Inicialmente, todos os estados possuem a mesma chance de serem lidos. Suponhamos que seja escolhido $x = 3$. Isto implica na amplificação da amplitude de todos os estados cuja imagem é menor do que 34.

O algoritmo finaliza sempre depois de $11,48\sqrt{N}$ chamadas do oráculo. Como $N = 16$, o algoritimo 4.5.2 fará 46 chamadas do oráculo. Isto é mais do que a quantidade necessária de execuções da função para resolver classicamente o problema, pois o tamanho de entrada é pequeno. Se N fosse igual a 10000, seriam necessárias 1250 execuções da função do algoritmo quântico, em contraposição às 10000 chamadas do algoritmo trivial clássico.

O algoritmo 4.5.2 irá executar o Oráculo Desigualdade até encontrar algum estado com imagem menor do que 34 ou quando chegue no limite superior do número de execuções do oráculo.

Monta-se o sistema no estado inicial $|0\rangle_4|38\rangle_6|0\rangle$. Defina $m = \frac{4}{3}$. Como o número de rodadas deve ser menor do que m, o único valor possível para a

x	f(x)	probabilidade
13	5	6,25%
7	11	6,25%
11	20	6,25%
0	25	6,25%
1	26	6,25%
8	26	6,25%
2	29	6,25%
14	32	6,25%
3	34	6,25%
4	41	6,25%
9	43	6,25%
12	43	6,25%
5	50	6,25%
6	61	6,25%
15	61	6,25%
10	62	6,25%

Tabela 4.4: tabela com as soluções de $f(x) = x^2 - 38 \bmod 63$ com $x < 16$, ordenadas pela imagem da função e a probabilidade inicial de obtenção de cada valor.

quantidade de rodadas antes da medição é igual a um.

Fazendo a superposição de todas as entradas e do último registrador tem-se que:

$$|\psi_1\rangle = H^{\otimes 4}|0\rangle_4|38\rangle_6 H|1\rangle = \frac{1}{4}\sum_{x=0}^{15}|x\rangle_4|38\rangle_6|-\rangle.$$

Aplicando o Oráculo Desigualdade $U_{g'}$, conforme sua descrição na fi-

gura 4.3, o estado do sistema passa a ser:

$$\begin{aligned}|\psi_2\rangle = \tfrac{1}{4}(\quad & -|\mathbf{0}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle - |\mathbf{1}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle - |\mathbf{2}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + |3\rangle_4|38\rangle_6|-\rangle \\ & +|4\rangle_4|38\rangle_6|-\rangle + |5\rangle_4|38\rangle_6|-\rangle + |6\rangle_4|38\rangle_6|-\rangle - |\mathbf{7}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle \\ & -|\mathbf{8}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + |9\rangle_4|38\rangle_6|-\rangle + |10\rangle_4|38\rangle_6|-\rangle - |\mathbf{11}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle \\ & +|12\rangle_4|38\rangle_6|-\rangle - |\mathbf{13}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle - |\mathbf{14}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + |15\rangle_4|38\rangle_6|-\rangle).\end{aligned}$$

Contando os estados acima em negrito, vemos que metade dos estados tiveram sua amplitude invertida. Quando isto acontece, temos que $\langle\psi_1|\psi_2\rangle = 0$. Isto significa que, após a inversão sobre a média, serão invertidas as amplitudes de todos os estados que compõem $|\psi_2\rangle$, ou seja, os estados que possuem amplitude negativa, em $|\psi_3\rangle$ terão amplitude positiva e os que possuiam amplitude positiva terão amplitude negativa em $|\psi_3\rangle$. Em módulo, a amplitude de todos os estados permanece a mesma, consequentemente, todos os estados do sistema continuarão com a mesma probabilidade de serem observados, conforme a tabela 4.4.

Se for medido um estado cuja imagem correspondente seja maior ou igual a 34, faz-se $m \leftarrow m \cdot \frac{4}{3}$, monta-se novamente o estado inicial e faz-se novamente a superposição de todas as entradas. Como o valor de m continua menor do que 2, é executado uma vez mais o oráculo e apenas uma medição. Suponha que desta vez seja medido $x = 8$. Como $f(8) = 26 < 34$, o valor 26 passa a ser o mínimo, e o laço interno será executado procurando elementos com imagem menor do que 26. Toma-se novamente $m \leftarrow \frac{4}{3}$ e é feita a superposição de todas as entradas a partir do estado inicial, obtendo-se:

$$\begin{aligned}|\psi_2\rangle \quad = \quad & \tfrac{1}{4}(-|\mathbf{0}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + |1\rangle_4|38\rangle_6|-\rangle + |2\rangle_4|38\rangle_6|-\rangle + |3\rangle_4|38\rangle_6|-\rangle + \\ & |4\rangle_4|38\rangle_6|-\rangle + |5\rangle_4|38\rangle_6|-\rangle + |6\rangle_4|38\rangle_6|-\rangle - |\mathbf{7}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + \\ & |8\rangle_4|38\rangle_6|-\rangle + |9\rangle_4|38\rangle_6|-\rangle + |10\rangle_4|38\rangle_6|-\rangle - |\mathbf{11}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + \\ & |12\rangle_4|38\rangle_6|-\rangle - |\mathbf{13}\rangle_{\mathbf{4}}|\mathbf{38}\rangle_{\mathbf{6}}|-\rangle + |14\rangle_4|38\rangle_6|-\rangle + |15\rangle_4|38\rangle_6|-\rangle).\end{aligned}$$

Desta vez, $\langle\psi_1|\psi_2\rangle = 1/2$. O estado do sistema após a inversão sobre a média é $|\psi_3\rangle = |\psi_1\rangle - |\psi_2\rangle$. A amplitude dos estados marcados passa de

1/4 para 1/2, e a amplitude dos estados não marcados passa a ser nula. A probabilidade de se obter um determinado estado na próxima medição é dada na tabela 4.5.

x	f(x)	probabilidade
13	5	25%
7	11	25%
11	20	25%
0	25	25%
1	26	0
8	26	0
2	29	0
14	32	0
3	34	0
4	41	0
9	43	0
12	43	0
5	50	0
6	61	0
15	61	0
10	62	0

Tabela 4.5: Probabilidade da medição após aplicação do Oráculo Desigualdade para valores menores do que 26.

Quando são amplificados exatamente 1/4 dos estados, a amplitude dos estados restantes passa a ser nula, assim, apenas os valores que satisfazem a condição de busca podem ser medidos. Supondo que seja medido, $x = 0$, isto implica que $y = 25$. Inicia-se novamente $m = \frac{4}{3}$ e aplica-se o oráculo procurando x tal que $f(x) < 25$. Após uma aplicação do oráculo e da inversão

sobre a média, teremos a probabilidade de cada estado dada pela tabela 4.6.

x	f(x)	probabilidade
13	5	31,64%
7	11	31,64%
11	20	31,64%
0	25	0,39%
1	26	0,39%
8	26	0,39%
2	29	0,39%
14	32	0,39%
3	34	0,39%
4	41	0,39%
9	43	0,39%
12	43	0,39%
5	50	0,39%
6	61	0,39%
15	61	0,39%
10	62	0,39%

Tabela 4.6: Probabilidade da medição depois de aplicar o Oráculo Desigualdade para valores menores do que 25.

Aos poucos, vão restando apenas os estados mais próximos do mínimo. O problema é que na medida em que vão restando menos estados marcados, o número de iterações necessárias para medir estes estados vai aumentando, pois é da ordem de $O\left(\sqrt{\frac{N}{t}}\right)$, onde t é o número de estados marcados.

Mesmo que a amplitude dos estados não invertidos pelo oráculo tenha sido nula em algum momento, isto não significa que permanecerão nulas, pois, entre outras coisas, o sistema deve ser reiniciado após cada medição.

4.5.4 Análise de corretude do algoritmo 4.5.2

Basicamente, o algoritmo 4.5.2, no interior do laço mais externo, procura um novo candidato a mínimo, e no laço intermediário, procura amplificar a amplitude dos estados correspondentes aos possíveis mínimos. Supondo que o algoritmo fosse executado indefinidamente, ele poderia não funcionar corretamente se o(s) mínimo(s) real(is) não tivesse(m) a amplitude de seu(s) estado(s) amplificado(s).

Definição 4 (Mínimo Potencial). *Ao executarmos um algoritmo de minimização sobre uma lista L, denominamos Mínimo Potencial a um elemento de L obtido pelo algoritmo, que seja menor do que todos os valores de L encontrados anteriormente nesta execução.*

Até se encontrar um novo mínimo potencial, a probabilidade de medição dos candidatos cresce até atingir um valor próximo a 100%.

A cada novo mínimo potencial encontrado, diminui a relação dos candidatos a mínimo, e os mínimos reais permanecem como candidatos. Assim, um mínimo real não será encontrado apenas se o algoritmo terminar prematuramente.

Considerando que a função f é injetora, que é o pior caso como comentado anteriormente, a quantidade estimada de chamadas do oráculo E é, no máximo:

$$\begin{cases} E(N,1) = S(N,1) \\ E(N,t) = S(N,t) + \frac{1}{t}\sum_{i=1}^{t-1} E(N,i), \end{cases} \tag{4.11}$$

onde N é o tamanho do domínio, t é o número de candidatos ao mínimo e $S(N,t)$ é a quantidade estimada de iterações do oráculo para encontrar um dos t elementos marcados pelo oráculo.

Tomando o valor de $S(N,t) = 3,36\sqrt{\frac{N}{t}}$ dado pelo teorema 7 na página 77, tem-se que:

$$E(N,t) = \frac{1}{t}\sum_{i=1}^{t-1} E(N,i) + 3,36\sqrt{\frac{N}{t}} \tag{4.12}$$

e

$$E(N,t-1) = \frac{1}{t-1}\sum_{i=1}^{t-2} E(N,i) + 3,36\sqrt{\frac{N}{t-1}}. \tag{4.13}$$

Multiplicando a equação (4.12) por t e (4.13) por $t-1$, obtém-se

$$tE(N,t) = \sum_{i=1}^{t-1} E(N,i) + 3,36\sqrt{Nt} \tag{4.14}$$

e

$$(t-1)E(N,t-1) = \sum_{i=1}^{t-2} E(N,i) + 3,36\sqrt{N(t-1)}. \tag{4.15}$$

Subtraindo a equação (4.15) de (4.14) e dividindo tudo por t, tem-se

$$E(N,t) = E(N,t-1) + 3,36\frac{\sqrt{N}}{t}(\sqrt{t} - \sqrt{t-1}). \tag{4.16}$$

Escrevendo a mesma equação para $t-1,\ldots,2$ e somando todas estas equações, chega-se a

$$E(N,t) = E(N,1) + 3,36\sqrt{N}\sum_{i=2}^{t}\frac{1}{i}(\sqrt{i} - \sqrt{i-1}), \tag{4.17}$$

que pode ser escrito como

$$E(N,t) < 2,38\sqrt{N} + 3,36\sqrt{N}\sum_{i=2}^{t}\frac{1}{\sqrt{i}}(1 - \sqrt{1-\frac{1}{i}}), \tag{4.18}$$

considerando que $E(N,1) = 2,38\sqrt{N}$. Aplicando a relação $(1-x)^n > (1-nx)$ e substituindo a somatória por uma integral, tem-se

$$E(N,t) < 2,38\sqrt{N} + 3,36\sqrt{N}\int_{i=1}^{t}\frac{1}{2z\sqrt{z}}dz, \tag{4.19}$$

que resolvida chega a

$$E(N,t) < 2,38\sqrt{N} + 3,36\sqrt{N}(1 - \frac{1}{\sqrt{t}}). \tag{4.20}$$

O limite superior é dado quando $t = N-1$, assim,

$$E(N,t) < 2,38\sqrt{N} + 3,36\sqrt{N}(1 - \frac{1}{\sqrt{N-1}}). \tag{4.21}$$

Resolvendo a relação de recorrência (4.11), temos que

$$E(N,N-1) < 5,74\sqrt{N}. \tag{4.22}$$

A desigualdade de Markov afirma que, dados uma variável aleatória x e um valor positivo a, $Pr(|x|) \geq a) \leq \frac{E(|x|)}{a}$, onde Pr é probabilidade e E é o valor esperado. Isto significa que se executarmos o oráculo duas vezes o valor esperado, $E(N, N-1)$, ou seja, $11,48\sqrt{N}$ aplicações do oráculo, temos pelo menos 50% de chance de obter o mínimo real.

4.5.5 Análise da complexidade do algoritmo 4.5.2

O algoritmo BBHT98 é encerrado após encontrar o elemento ou atingir um limite superior de passos, que não está explícito na descrição do algoritmo. Já o algoritmo 4.5.2 sempre finaliza a sua execução antes de ultrapassar uma quantidade de passos pré-definidos. O algoritmo é encerrado antes de fazer $11,48\sqrt{N}$ chamadas do Oráculo Desigualdade sobre a função de otimização f. Como cada chamada deste oráculo possui complexidade $P_f(N)$, portanto a complexidade de tempo do algoritmo, ou seja do tamanho do circuito, é $11,48\sqrt{N} \cdot P_f(N)$, conforme descrição do algoritmo.

Lema 8. *A quantidade de medições realizadas é $\Omega(n^2)$ onde n é a quantidade de bits que representa o tamanho do domínio da função.*

Demonstração. Seja $M(N,t)$ a estimativa do número de medições realizadas pelo algoritmo para encontrar algum dos t elementos marcados em um conjunto de N elementos.

Quando há apenas um elemento marcado, para encontrar este elemento, que é o mínimo absoluto, são realizadas um número crescente de repetições do oráculo entre as medições, em uma progressão geométrica, então tem-se

$$S(N,t) = \frac{1}{2} \sum_{i=1}^{M(n,1)} \lambda^i = \frac{1}{2} \left(\frac{\lambda^{M(N,1)} - 1}{\lambda - 1} \right). \tag{4.23}$$

Isto implica em que

$$M(N,1) = \log_\lambda(2S(N,1) \cdot (\lambda - 1) + 1). \tag{4.24}$$

Da equação anterior, podemos afirmar que,

$$M(N,1) > \log_\lambda S(N,1). \tag{4.25}$$

Após encontrar um mínimo potencial qualquer, supondo que hajam t elementos menores do que este mínimo, o número de medições esperado M^* é

$$M^*(N,t) = \log_\lambda(2S(N,t)\cdot(\lambda-1)+1). \tag{4.26}$$

Então,

$$M^*(N,t) > \log_\lambda S(N,t). \tag{4.27}$$

Considerando as medições executadas anteriormente para encontrar todos os mínimos potenciais até o momento (com as devidas probabilidades de ocorrência), tem-se:

$$\begin{cases} M(N,1) = \log_\lambda(2S(N,1)\cdot(\lambda-1)+1) \\ M(N,t) = M^*(N,t) + \sum_{i=1}^{t-1} \frac{M^*(N,i)}{i}, \text{ para } t \geq 2, \end{cases} \tag{4.28}$$

Substituindo as equações (4.25) e (4.27) na equação anterior, tem-se que:

$$\begin{cases} M(N,1) > \log_\lambda S(N,1) \\ M(N,t) > \log_\lambda S(N,t) + \sum_{i=1}^{t-1} \frac{\log_\lambda S(N,i)}{i}, \text{ para } t \geq 2, \end{cases} \tag{4.29}$$

Pelo lema 6 da página 75,

$$S(N,t) = \frac{1}{2}\frac{\lambda^4}{\lambda-1}\left\lceil \frac{N}{2\sqrt{t(N-t)}} \right\rceil. \tag{4.30}$$

Isto significa que

$$\frac{1}{2}\lambda^3\sqrt{\frac{N}{t}} \leq S(N,t) \leq \frac{1}{\sqrt{2}}\frac{\lambda^4}{\lambda-1}\sqrt{\frac{N}{t}}. \tag{4.31}$$

Logo,

$$M(N,t) > \log_\lambda\left(\frac{\lambda^3}{2}\sqrt{\frac{N}{t}}\right) + \sum_{i=1}^{t-1}\frac{1}{i+1}\log_\lambda\left(\frac{\lambda^3}{2}\sqrt{\frac{N}{i}}\right). \tag{4.32}$$

Separando alguns membros da somatória, a equação anterior torna-se

$$M(N,t) > \log_\lambda\left(\frac{\lambda^3}{2}\sqrt{\frac{N}{t}}\right) + \sum_{i=1}^{t-1}\frac{1}{i+1}\log_\lambda\left(\frac{\lambda^3}{2}\sqrt{N}\right) - \sum_{i=1}^{t-1}\frac{\log_\lambda\sqrt{i}}{i+1}. \tag{4.33}$$

Isto implica em

$$M(N,t) > \log_\lambda \left(\frac{\lambda^3}{2}\sqrt{N}\right) - \frac{\log_\lambda t}{2} + \log_\lambda \left(\frac{\lambda^3}{2}\sqrt{N}\right) \sum_{i=1}^{t-1} \frac{1}{i+1} - \sum_{i=1}^{t-1} \frac{\log_\lambda i}{2(i+1)}. \tag{4.34}$$

Rearranjando os termos de uma somatória e considerando que $\log_\lambda 1 = 0$ e $\frac{1}{i+1} < \frac{1}{i}$, temos que

$$M(N,t) > \log_\lambda \left(\frac{\lambda^3}{2}\sqrt{N}\right) - \frac{\log_\lambda t}{2} + \log_\lambda \left(\frac{\lambda^3}{2}\sqrt{N}\right) \sum_{i=2}^{t} \frac{1}{i} - \sum_{i=2}^{t-1} \frac{\log_\lambda i}{2i}. \tag{4.35}$$

A somatória de $\frac{1}{i}$ com i variando de 2 até t é igual ao t-ésimo número harmônico H_t-1. Sendo assim,

$$M(N,t) > \log_\lambda \left(\frac{\lambda^3}{2}\sqrt{N}\right) H_t - \frac{\log_\lambda t}{2} - \sum_{i=2}^{t-1} \frac{\log_\lambda i}{2i}. \tag{4.36}$$

Minorando a somatória por uma integral, obtemos

$$M(N,t) > \log_\lambda \left(\frac{\lambda^3}{2}\sqrt{N}\right) H_t - \frac{\log_\lambda t}{2} - \frac{1}{2}\int_{i=1}^{t-1} \frac{\log_\lambda i}{i} di. \tag{4.37}$$

Fazendo mudança de base do logaritmo e calculando a integral, temos que

$$M(N,t) > \log_\lambda \frac{\lambda^3}{2} H_t + \frac{1}{2\ln\lambda} \ln N H_t - \frac{\log_\lambda t}{2} - \frac{\ln^2(t-1)}{4\ln\lambda}. \tag{4.38}$$

Minorando H_t, chegamos a

$$M(N,t) > \log_\lambda \frac{\lambda^3}{2} \ln(t+1) + \frac{1}{2\ln\lambda} \ln N \ln(t+1) - \frac{\log_\lambda t}{2} - \frac{\ln^2(t-1)}{4\ln\lambda}. \tag{4.39}$$

Como o caso limite para o algoritmo ocorre com $t = N-1$, a equação anterior a passa a ser

$$M(N,N-1) > \log_\lambda \frac{\lambda^3}{2} \ln N + \frac{\ln^2 N}{2\ln\lambda} - \frac{\log_\lambda(N-1)}{2} - \frac{\ln^2(N-2)}{4\ln\lambda}. \tag{4.40}$$

Da equação anterior, chega-se que

$$M(N,N-1) > \log_\lambda \frac{\lambda^3}{2} \ln N + \frac{\ln^2 N}{4\ln\lambda} - \frac{\ln N}{2\ln\lambda}. \tag{4.41}$$

De onde podemos concluir que

$$M(N,N-1) > \frac{\ln^2 N}{4\ln\lambda} + \ln N(\log_\lambda \frac{\lambda^3}{2} - \frac{1}{2\ln\lambda}). \tag{4.42}$$

Como λ é constante, $M(N,N-1) \in \Omega(\log^2 N)$. □

4.6 Algoritmo Medida-Linear

Além das adaptações necessárias ao algoritmo DH96, para sua utilização em problemas combinatórios, pode-se otimizá-lo de forma a diminuir o número de medições de $\Omega(\log^2 N)$ para $O(\log N)$. Uma forma simples para isso, seria usar o algoritmo de Grover original (algoritmo 4.2.1) no lugar do BBHT98 (algoritmo 4.3.1), executando sempre $\Theta(\sqrt{N})$ rodadas do oráculo antes de cada medição. Isto evita medições que possuem pouca chance de sucesso, mas, por outro lado, faz com que a complexidade de tempo do algoritmo suba para $\Theta(\log N\sqrt{N}) \cdot P_f(N)$. É possível diminuir o número de medições para apenas uma medição, através da adição de novos registradores, um para cada possível medição intermediária. São necessários $\Theta(\log^2 N)$ registradores adicionais, cada um com $\Theta(\log N)$ q-bits. Assim, o algoritmo passa a usar espaço cúbico em relação ao número de q-bits necessários para representar a entrada.

Nesta seção, apresentamos o algoritmo que estamos propondo, denominado *Medida-Linear* (ML), para encontrar o mínimo com $O(\log N)$ medições e $8,27\sqrt{N}$ execuções do oráculo. Caso se deseje fazer apenas uma medição no final da execução, são necessários $O(\log N)$ registradores com $\Theta(\log N)$ q-bits cada. Neste caso, o espaço passa a ser quadrático.

4.6.1 Descrição do algoritmo

Este algoritmo, assim como o DH96, é baseado no BBHT98. Enquanto o algoritmo DH96 chama o algoritmo BBHT98 várias vezes para buscar um elemento na lista, o algoritmo Medida-Linear é uma variação do próprio BBHT98. A idéia do algoritmo que estamos propondo é fazer uma adaptação no BBHT98 de forma que, em vez de o algoritmo encerrar após encontrar um elemento que satisfaça a condição de busca, o algoritmo continue procurando elementos que satisfaçam um novo critério. Este critério é, no caso de um

problema de minimização, encontrar algum elemento que seja menor do que o menor elemento encontrado até agora (mínimo potencial).

Uma diferença prática entre o algoritmo Medida-Linear e o DH96 é que, a cada novo mínimo potencial encontrado, o DH96 recomeça a procura de um mínimo absoluto, repetindo poucas vezes o operador de Grover entre duas medições consecutivas, enquanto que, no algoritmo Medida-Linear, a quantidade de repetições deste operador, antes de cada nova medida, é sempre crescente até chegar a $\Theta(\sqrt{N})$.

No algoritmo 4.6.1, é apresentada a descrição deste procedimento.

Para mostrar os resultados a seguir, consideramos o caso em que todos os elementos da lista são distintos. Este é o pior caso do algoritmo, pois caso haja elementos repetidos, se algum destes se tornar um mínimo potencial, na iteração seguinte, todos os outros iguais a ele não são mais marcados pelo Oráculo Desigualdade, e o algoritmo ML converge mais rapidamente para um mínimo real.

As definições seguintes são usadas na demonstração dos próximos resultados.

Definição 5. *Chamamos de $y_1, y_2, \ldots, y_l$ a seqüência de mínimos potenciais encontrados pelo algoritmo ML. Pela própria definição de mínimo potencial, esta seqüência é estritamente decrescente.*

Chamamos de t_k, a quantidade de elementos da lista com valores menores do que y_k, para $1 \leq k \leq l$.

4.6.2 Análise da corretude

A análise de corretude do algoritmo Medida-Linear pode ser feita através da verificação de que, após a quantidade de iterações determinadas pelo algoritmo, a chance de encontrar um mínimo real é, de pelo menos, 50%.

As proposições 2 e 3 da página 68, demonstradas em [6] e em [34, cap.6],

Algoritmo 4.6.1 Algoritmo Medida-Linear

1: Escolha aleatoriamente, com distribuição uniforme, um valor x inteiro não-negativo menor do que N.
2: $y \leftarrow f(x)$
3: $i \leftarrow 0$
4: Inicialize $\lambda \leftarrow \frac{13}{12}$ e $m \leftarrow \lambda$.
5: **enquanto** $i < 8,27\sqrt{N}$ **faça**
6: Escolha aleatoriamente com distribuição uniforme um inteiro positivo j, tal que $j < m$.
7: Prepare o sistema no estado inicial $|\psi_0\rangle = |0\rangle_n|y\rangle_l|1\rangle$.
8: Aplique a porta H no terceiro registrador.
9: Faça a superposição de todos os estados da base computacional no primeiro registrador.
10: **para** j iterações **faça**
11: Aplique o Oráculo Desigualdade para inverter os estados cuja imagem seja menor do que y.
12: Aplique a inversão sobre a média.
13: **fim para**
14: Observe o primeiro registrador e guarde em x' o valor medido.
15: **se** $f(x') < y$ **então**
16: $y \leftarrow f(x')$
17: **fim se**
18: $i \leftarrow i + j$
19: $m \leftarrow min(\lambda m, \sqrt{N})$
20: **fim enquanto**
21: **devolva** y e o valor correspondente do primeiro registrador.

se referem ao oráculo de Grover, mas podem ser aplicadas a qualquer outro oráculo que marca os estados através da inversão da amplitude dos mesmos, como é o caso do Oráculo Desigualdade.

Proposição 9. *[6] Dada uma lista L com N elementos, dos quais t são marcados pelo oráculo, caso sejam realizadas medidas intercaladas por um número crescente de passos numa progressão geométrica de razão λ, a quantidade total estimada de execuções do oráculo para medir algum dos t elementos marcados com, pelo menos 25% de chance, é, no máximo*

$$E_{0,25}(N,t) = \frac{1}{4}\frac{\lambda}{\lambda-1}\frac{N}{\sqrt{(N-t)t}}.$$

Em particular, se $\lambda = \frac{13}{12}$, então $E_{0,25}(N,t) < \frac{13}{4}\frac{N}{\sqrt{(N-t)t}}$.

Teorema 10. *Dada uma lista L com N elementos, a quantidade estimada de execuções do oráculo necessárias para o algoritmo ML encontrar um mínimo absoluto da lista, com menos de 50% de erro é, no máximo, $8,27\sqrt{N}$.*

Demonstração. Classicamente, após l medições aleatórias, espera-se que haja no máximo $\frac{N}{l}$ elementos da lista que sejam menores do que último mínimo potencial encontrado. Como a amplitude de cada estado não marcado pelo oráculo em nenhum momento é maior do que a de um estado não marcado quando o mínimo potencial for menor do que metade da lista, o algoritmo quântico não pode ser pior do que a amostragem clássica. Se $\lambda = \frac{13}{12}$, as 8 primeiras medições são precedidas por apenas uma execução do oráculo. Por isso, após quatro medições, estima-se que haja aproximadamente, $\frac{N}{4}$, elementos na lista que sejam menores do que o último mínimo potencial encontrado. Quando há aproximadamente $\frac{N}{4}$ elementos que satisfazem a busca do oráculo, pela proposição 2, após uma iteração do oráculo, haverá quase 100% de chance de obter um novo mínimo potencial. Todos os elementos marcados pelo oráculo possuem a mesma chance de serem medidos. Então,

após a quinta medição, estima-se que haja, aproximadamente, $\frac{N}{8}$ elementos na lista menores do que este último mínimo potencial encontrado.

Chamando de l o número total de medições feitas até encontrar o mínimo potencial y_k, tem-se que $m = \lambda^l$. Isto significa que contando todas as l medições, foram realizadas, na média, $S_l = \frac{1}{2}\sum_{s=1}^{l} \lambda^s$ iterações do oráculo. Utilizando a fórmula da soma de uma progressão geométrica, e considerando $\lambda = \frac{13}{12}$, temos que: $S_l < 6,5m$.

Chamemos de m_{t_k} o valor limite do número de iterações escolhido com distribuição uniforme, que garanta pelo menos 25% de chance de sucesso para encontrar um mínimo potencial, quando o oráculo marca t_k elementos. Pela proposição 5 na página 75, temos que

$$m_{t_k} = \frac{N}{2\sqrt{(N-t_k)t_k}}.$$

Podemos separar duas situações possíveis para o valor da variável m ao obter o k-ésimo mínimo potencial:

$$\begin{cases} a) & m \geq m_{t_k} \\ b) & m < m_{t_k}. \end{cases}$$

Se $m \geq m_{t_k}$ significa que a chance de encontrar um novo mínimo potencial na próxima medição é de, pelo menos, 25%. Cada nova medida até encontrar o novo mínimo potencial tem também, isoladamente, $\frac{1}{4}$ de chance de sucesso. Logo, são esperadas outras 3 medições para obter um novo mínimo potencial. Isto significa que, o número estimado de repetições do oráculo, a partir deste momento até encontrar o próximo mínimo potencial, é $(m\lambda + m\lambda^2 + m\lambda^3)/2$. Para $\lambda = \frac{13}{12}$, é no máximo $1,77m$.

No caso (b), em que $m < \frac{N}{2\sqrt{(N-t_k)t_k}}$, ainda não se chegou ao número de iterações em que há garantia de probabilidade de sucesso maior ou igual a 25%. Neste caso, mesmo que m continue aumentando e se torne igual a m_{t_k}, garantindo pelo menos 25% de sucesso, o número estimado de iterações,

incluindo as anteriores, não ultrapassa $\frac{\lambda}{2(\lambda-1)}m_{t_k}$. Ou seja, para $\lambda = \frac{13}{12}$, o número de iterações é no máximo $6,5m_{t_k}$. Para este valor de λ, a estimativa das iterações restantes é menor do que $3,27m_{t_k}$, pelo mesmo raciocínio do caso anterior; perfazendo um total estimado de, no máximo, $8,27m_{t_k}$ iterações até encontrar o $(k+1)$-ésimo mínimo potencial.

Vejamos agora, por indução que, para $\lambda = \frac{13}{12}$, a partir do momento em que $t_k \leq \frac{N}{11}$, entre duas medições, a situação esperada é o segundo caso, ou seja, após obter o k-ésimo mínimo potencial, ainda não se chegou na faixa em que há pelo menos 25% de chance de encontrar o $(k+1)$-ésimo mínimo potencial. Isto implica em que o tempo total esperado do algoritmo até chegar ao $(k+1)$-ésimo mínimo potencial é, no máximo, $6,5\sqrt{\frac{N}{(N-t_k)t_k}}$ execuções do oráculo.

Conforme foi comentado no início desta demonstração, após 4 passos, a estimativa é de que, a partir deste passo, sejam marcados no máximo $\frac{N}{8}$ elementos da lista. Portanto, mesmo que a amplitude dos estados marcados não fosse aumentada, estima-se que após 11 passos já se atinja a condição de que $t_k \leq \frac{N}{11}$, e o número de passos, $\frac{8,27}{2}\sqrt{\frac{N}{(N-t_k)t_k}}$, é maior do que 11 para qualquer valor de $t_k \leq \frac{N}{2}$.

Por hipótese, $m < m_{t_k} = \frac{N}{2\sqrt{(N-t_k)t_k}}$, quando se atinge 25% de chance de obter o k-ésimo mínimo potencial. Todas as medições após este ponto, têm, cada uma, pelo menos 25% de chance de sucesso para encontrar o próximo mínimo potencial. Assim, estima-se que são necessárias outras quatro medições. Isto significa que o valor estimado para m, quando encontra o k-ésimo mínimo potencial é $m = \lambda^4 m_{t_k}$.

Como todos os elementos marcados possuem a mesma chance de serem lidos, dado que há t_k elementos menores do que o k-ésimo mínimo potencial, o valor estimado para t_{k+1} é $t_{k+1} = \frac{t_k}{2}$.

Assim, tem-se que

$$m_{t_{k+1}} = \frac{N}{2\sqrt{(N - t_{k+1})t_{k+1}}} = \frac{N}{2\sqrt{(N - \frac{t_k}{2})\frac{t_k}{2}}}. \tag{4.43}$$

O algoritmo está no caso (b) se $m = \lambda^4 m_{t_k} < m_{t_{k+1}}$, ou seja, se

$$\frac{m_{t_{k+1}}}{m_{t_k}} > \lambda^4. \tag{4.44}$$

Como $\lambda = \frac{13}{12}$, precisamos ter

$$\left(\frac{m_{t_k}}{m_{t_{k+1}}}\right)^2 < 0,527. \tag{4.45}$$

Tem-se que:

$$\frac{m_{t_k}}{m_{t_{k+1}}} = \frac{\frac{N}{2\sqrt{(N-t_k)t_k}}}{\frac{N}{2\sqrt{(N-\frac{t_k}{2})\frac{t_k}{2}}}}. \tag{4.46}$$

Isto significa que

$$\left(\frac{m_{t_k}}{m_{t_{k+1}}}\right)^2 = \frac{(N - \frac{t_k}{2})\frac{1}{2}}{N - t_k} = \frac{(N - t_k + \frac{t_k}{2})}{2(N - t_k)}. \tag{4.47}$$

A expressão anterior pode ser escrita como

$$\left(\frac{m_{t_k}}{m_{t_{k+1}}}\right)^2 = \frac{1}{2} + \frac{t_k}{4(N - t_k)}, \tag{4.48}$$

Seja $c = \frac{N}{t_k}$. Isto significa que

$$\left(\frac{m_{t_k}}{m_{t_{k+1}}}\right)^2 = \frac{1}{2} + \frac{N}{4(cN - N)} = 0,5 + \frac{1}{4(c-1)}, \tag{4.49}$$

Pela condição inicial de que $t_k < \frac{N}{11}$, temos que $c > 11$. Assim,

$$\left(\frac{m_{t_k}}{m_{t_{k+1}}}\right)^2 < 0,5 + \frac{1}{40}, \tag{4.50}$$

Podemos concluir que,

$$\left(\frac{m_{t_k}}{m_{t_{k+1}}}\right)^2 < 0,525, \tag{4.51}$$

Isto significa que ocorre o caso (b).

Sendo assim, o número total esperado de execuções do oráculo para encontrar o k-ésimo mínimo potencial é $E(N,k) = 8,27m_{t_k}$.

O pior caso ocorre quando há um único mínimo absoluto na lista. Neste caso, a quantidade máxima esperada de iterações é $4,133\sqrt{N}$.

Para obter 50% de chance de sucesso, usando a desigualdade de Markov, são necessárias, aproximadamente, $8,27\sqrt{N}$ execuções do oráculo. □

4.6.3 Análise da complexidade

A complexidade de tempo é fixada pela condição no passo 5 do algoritmo ML: $8,27\sqrt{N}$ iterações do Oráculo Desigualdade. O custo de cada execução do oráculo depende do custo de verificação da função f a ser otimizada.

A complexidade de espaço também depende da implementação de f.

A ordem de complexidade do número de medições é dado pelo teorema seguinte.

Teorema 11. *A quantidade de medições do algoritmo ML é linear, em relação ao número de bits necessários para representar o tamanho da lista.*

Demonstração. A variável j, que define o número de repetições do oráculo antes de cada medição, é escolhida aleatoriamente, com distribuição uniforme entre os valores inteiros no intervalo $[1, m]$. Isto significa que o valor estimado para j é $\frac{\lfloor m \rfloor}{2}$, onde m cresce numa progressão geométrica (p.g.) de razão λ entre duas medições, a partir de λ. Sendo assim, o valor estimado para j também cresce numa p.g. de razão λ, mas começando de $\frac{\lambda}{2}$.

Somando todos os valores que j assume durante a execução do algoritmo espera-se que sejam executadas S_m iterações do oráculo, onde:

$$S_m = \sum_{k=1}^{M} \frac{\lambda^k}{2} \tag{4.52}$$

e M é o número de medições realizadas.

Utilizando a fórmula para soma de uma p.g. finita, tem-se que

$$S_m = \frac{1}{2}\frac{\lambda}{\lambda - 1}(\lambda^M - 1). \tag{4.53}$$

Como estamos considerando $\lambda = \frac{13}{12}$, a expressão anterior torna-se:

$$S_m = 6,5\left(\frac{13}{12}\right)^M - 6,5. \tag{4.54}$$

A expressão anterior pode ser escrita como

$$\frac{S_m}{6,5} + 1 = \left(\frac{13}{12}\right)^M. \tag{4.55}$$

Isolando a variável M que indica o número de medições, tem-se que

$$M = \log_{\frac{13}{12}}\left(\frac{S_m}{6,5} + 1\right). \tag{4.56}$$

Pela descrição do passo 5 do algoritmo ML, são realizadas $S_m = 8,27\sqrt{N}$ chamadas do oráculo. Logo,

$$M = \log_{\frac{13}{12}}\left(\frac{8,27\sqrt{N}}{6,5} + 1\right) \approx 1,985 + 4,33\log_2 N. \tag{4.57}$$

Isto significa que o número de medições no algoritmo ML é $O(n)$, onde n é o número de bits necessários para representar o tamanho N da lista. □

4.6.4 Considerações sobre o exemplo

Em relação ao exemplo dado na sub-seção 4.5.3, o algoritmo 4.6.1 proposto possui comportamento parecido com o do algoritmo 4.5.2. Isto acontece porque no início é feita apenas uma aplicação do Oráculo Desigualdade antes de cada medida. A diferença começa a surgir quando a proporção de elementos marcados começa a diminuir. Neste caso, a cada novo mínimo encontrado, o algoritmo 4.6.1 não reinicializa o valor m que define o número de chamadas em cada nova busca como faz o algoritmo 4.5.2. Assim, o algoritmo 4.6.1 evita fazer medições e aplicações do oráculo que possuem pouca chance de revelar novas informações.

Capítulo 5

Simulação dos Algoritmos Quânticos para problemas de minimização

Uma forma de comparar o desempenho dos algoritmos é através de simulações. As simulações também ajudam a entender melhor o funcionamento dos algoritmos aplicados a problemas concretos, principalmente considerando que atualmente não existem computadores quânticos de porte prático. Mas os algoritmos quânticos possuem um obstáculo considerável para serem simulados: a quantidade exponencial de bits necessária para representar o estado global do sistema quântico em um sistema clássico. Isto acontece porque o estado de um sistema com n q-bits é um vetor normalizado do espaço de Hilbert de dimensão 2^n e, neste caso, é necessário armazenar a amplitude de cada elemento da base. O cálculo da evolução deste sistema necessita de um número exponencial de passos em um computador clássico [31]. Sendo assim, no caso de problemas NP-difíceis e outros problemas em que as soluções não são obtidas em tempo polinomial, é possível simular adequadamente apenas pequenas instâncias.

Em se tratando dos algoritmos para resolver problemas de otimização

referidos no capítulo anterior, em cada instante, há apenas dois valores distintos de amplitude dos estados: uma para todos os estados que satisfazem a condição de procura do oráculo e outra para os estados que não satisfazem esta condição. Sendo assim, é necessário armazenar apenas dois valores distintos de amplitude, em cada iteração. Isto facilita a simulação destes algoritmos, mas não resolve o problema, pois o cálculo destas amplitudes depende da informação da quantidade de elementos marcados e do número de repetições do oráculo; ou seja, depende do conhecimento do número de soluções do problema.

Nas simulações são abstraídos vários aspectos da realidade que se deseja simular. Concretamente nas simulações dos algoritmos tratados no capítulo anterior, a questão relativa à forma como os dados são obtidos, se a partir de uma função ou a partir de uma lista, está em um segundo plano, diminuindo as diferenças entre as versões dos algoritmos para lista e para problemas de otimização. Ou seja, é abstraída a forma como é implementado o oráculo. A principal diferença que permanece nas simulações é o valor da constante λ, que define a progressão geométrica do número de iterações do oráculo entre medições sucessivas. No caso do algoritmo DH96 original, $\lambda = \frac{8}{7}$, valor não explícito na descrição do algoritmo, mas herdado do algoritmo BBHT98, enquanto que na versão para problemas de otimização combinatória, definimos na formulação do algoritmo $\lambda = \frac{4}{3}$, que dá um resultado melhor.

Primeiramente é mostrado o comportamento dos algoritmos DH96 [12] e Medida-Linear aplicados a problemas NP-difíceis. Para diminuir o custo computacional da simulação, em vez de calcular a função de otimização em cada aplicação do oráculo, as soluções foram todas pré-calculadas e armazenadas em um vetor ordenado crescentemente.

Os problemas tratados são: a versão de otimização de uma variação do 3-Sat [14] e resolução de equações binárias quadráticas, problema descrito na sub-seção 4.2.2. Para ambos problemas, foram consideradas instâncias de

até 16 bits, visto que a simulação destes problemas é exponencial conforme comentado anteriormente. Foram consideradas as versões infinitas de ambos algoritmos, ou seja, o algoritmo termina assim que encontra um mínimo absoluto e não apenas após um número fixo de passos.

Uma forma de verificar o comportamento assintótico de ambos algoritmos é tomar, como função de otimização f uma função estritamente crescente. Para todos os efeitos, o algoritmo não utiliza esta propriedade. Mas com esta informação, a cada novo mínimo potencial encontrado, é possível saber quantos elementos são menores do que ele e, consequentemente, calcular a probabilidade de encontrar um novo mínimo potencial na medição seguinte. Como foi visto no capítulo 4, o pior caso para os algoritmos ocorre quando f é uma função injetora. Sendo assim, também fizemos uma simulação sobre funções afim da forma $f(x) = ax+b$, onde a e b foram escolhidos aleatoriamente dentro de um intervalo com instâncias de até 2^{100} entradas. Estas simulações foram feitas sobre o algoritmo 4.5.2, versão de otimização do algoritmo DH96, mas em vez de finalizar o processo após um número pré-determinado de passos, o algoritmo termina quando encontra o mínimo absoluto, que é único e é o primeiro elemento do domínio. A idéia de executar as versões infinitas dos algoritmos é conferir se as médias das execuções do oráculo necessárias para encontrar um mínimo absoluto concordam com a análise teórica. Utilizando a função afim é possível simular um caso pior, em termos de complexidade para os algorítmos, do que o caso das funções relacionadas com problemas NP-difíceis, pois, neste último caso, pode haver mais de uma solução (mínimo absoluto), o que não acontece com a função afim. Além disso, como foi visto no capítulo anterior, o caso em que todos os elementos são distintos é o pior caso para o algoritmo DH96 e para o Mínimo-Linear.

Os parâmetros comparados foram:

- Quantidade de aplicações do oráculo

- Quantidade de Medições

Para cada um destes parâmetros, foi calculada a média para cada tamanho distinto de entrada.

Além destes parâmetros, também foi calculada a proporção de execuções que ultrapassam um patamar. Este patamar é o número de execuções do oráculo definido na formulação dos algoritmos. No caso do DH96, é utilizado o número de iterações definidos na formulação original do algoritmo: $22,5\sqrt{N}$, onde N é a quantidade de elementos do domínio da função de otimização ou o tamanho da lista dependendo do problema considerado, quando $\lambda = \frac{8}{7}$ e $11,48\sqrt{N}$, para $\lambda = \frac{4}{3}$. O patamar para o algoritmo Medida-Linear, conforme definido na descrição do algoritmo 4.6.1, é $8,27\sqrt{N}$.

Para cada instância, foram realizadas 50 execuções de cada algoritmo.

Para a simulação dos dois problemas NP-difíceis, foi montada uma lista com todas as soluções ordenadas crescentemente em relação à imagem da função. Quanto à simulação da função afim, não foi montada a lista, pois a função já é crescente. A ordenação não era necessária, e esta informação foi utilizada apenas para facilitar a contagem dos estados amplificados e seleção dos candidatos à mínimo.

Foram montadas apenas instâncias de tamanho potências de 2. Para cada instância do problema, os parâmetros da função eram escolhidos aleatoriamente dentro de uma faixa de valores. De posse da função, calculou-se as imagens de todas as entradas, ordenando-as.

Todas as simulações, referidas neste capítulo, foram feitas usando o software Pari GP versão 2.2.10 [44].

Os procedimentos para simulação do DH96 e do algoritmo Medida-Linear estão descritos nos algoritmos 5.0.2 e 5.0.3 respectivamente. A descrição da simulação de ambos é bastante parecida, a diferença está na reinicialização da variável m que é feita após cada novo mínimo potencial encontrado (passo 11

do algoritmo 5.0.2), enquanto que, no algoritmo 5.0.3, ela possui valor igual a um apenas no início (passo 10). Estes algoritmos estão escritos para os problemas sobre listas. Para os problemas sobre funções, deve substituir $F[x]$ por $f(x)$, além de omitir o passo referente à ordenação de F.

Para efeitos comparativos, também é mostrado o resultado da simulação do algoritmo BBHT98, mas como se trata de um algoritmo de busca, é informado ao algoritmo qual é o valor do mínimo absoluto. O procedimento para a simulação do algoritmo BBHT98 é mostrado no algoritmo 5.0.4. A descrição deste algoritmo é muito parecida com a do algoritmo de simulação 5.0.3. As diferenças estão nos passos 19 e 29, que se referem ao número de estados marcados pelo oráculo e condição de parada das buscas, respectivamente.

Nas simulações dos três problemas considerados, o algoritmo DH96 foi executado com $\lambda = \frac{8}{7}$ e $\lambda = \frac{4}{3}$. O algoritmo Medida-Linear foi simulado com $\lambda = \frac{13}{12}$, apesar de que, como se verá adiante, há outros valores de λ que dão melhores resultados neste último algoritmo. Para mostrar a relação entre λ e o desempenho dos algoritmos, no final da última seção deste capítulo é mostrado o resultado de uma simulação variando o valor de λ em ambos algoritmos para um determinado tamanho de entrada.

5.1 Simulação do problema de otimização associado ao 3-Sat

O problema de decisão 3-Satisfabilidade, mais conhecido como 3-Sat, consiste em, dados um conjunto U de variáveis lógicas e uma coleção C de cláusulas (disjuntivas) sobre U, cada uma contendo exatamente 3 literais, saber se há uma combinação de valores para as variáveis em U, que satisfaça todas as cláusulas em C [14].

Há uma variação deste problema, que inclui um valor k, e deseja-se saber

Algoritmo 5.0.2 Simulação do Algoritmo DH para encontrar o mínimo.

1: Definição do tamanho da entrada N.
2: Definição dos parâmetros da função f.
3: Definição do valor de λ.
4: Montagem de uma lista F com os resultados de f para cada entrada.
5: Ordenação crescente de F.
6: $medidas \leftarrow 1$
7: $cont \leftarrow 0$
8: $x \leftarrow \lfloor \frac{N}{2} \rfloor$
9: $mínimo \leftarrow F[x]$
10: **enquanto** $mínimo > F[1]$ **faça**
11: $m \leftarrow 1$ $\{m \text{ é reinicializado}\}$
12: **repita**
13: $m \leftarrow m \cdot \lambda$
14: **se** $m > \sqrt{N}$ **então**
15: $m \leftarrow \sqrt{N}$
16: **fim se**
17: $j \leftarrow random(m)$
18: $cont \leftarrow cont + j$
19: $t \leftarrow$ quantidade de estados menores do que $F[x]$
20: $\theta \leftarrow asen(\sqrt{\frac{t}{N}})$
21: $prob \leftarrow sen^2(j \cdot \theta)$
22: **se** $prob \cdot 2^{31} > random(2^{31})$ **então**
23: $x \leftarrow random(t) + 1$
24: **senão**
25: $x \leftarrow random(N - t) + t + 1$
26: **fim se**
27: $y \leftarrow F[x]$
28: $medidas \leftarrow medidas + 1$
29: **até que** $y < mínimo$
30: **fim enquanto**
31: **devolva** y

Algoritmo 5.0.3 Simulação do Algoritmo Medida-Linear para encontrar o mínimo.

1: Definição do tamanho da entrada N.
2: Definição dos parâmetros da função f.
3: Definição do valor de λ.
4: Montagem de uma lista F com os resultados de f para cada entrada.
5: Ordenação crescente de F.
6: $medidas \leftarrow 1$
7: $cont \leftarrow 0$
8: $x \leftarrow \lfloor \frac{N}{2} \rfloor$
9: $mínimo \leftarrow F[x]$
10: $m \leftarrow 1$
11: **enquanto** $mínimo > F[1]$ **faça**
12: **repita**
13: $m \leftarrow m \cdot \lambda$
14: **se** $m > \sqrt{N}$ **então**
15: $m \leftarrow \sqrt{N}$
16: **fim se**
17: $j \leftarrow random(m)$
18: $cont \leftarrow cont + j$
19: $t \leftarrow$ quantidade de estados menores do que $F[x]$
20: $\theta \leftarrow asen(\sqrt{\frac{t}{N}})$
21: $prob \leftarrow sen^2(j \cdot \theta)$
22: **se** $prob \cdot 2^{31} > random(2^{31})$ **então**
23: $x \leftarrow random(t) + 1$
24: **senão**
25: $x \leftarrow random(N - t) + t + 1$
26: **fim se**
27: $y \leftarrow F[x]$
28: $medidas \leftarrow medidas + 1$
29: **até que** $y < mínimo$
30: **fim enquanto**
31: **devolva** y

Algoritmo 5.0.4 Simulação do Algoritmo BBHT para encontrar o mínimo conhecido.

1: Definição do tamanho da entrada N.
2: Definição dos parâmetros da função f.
3: Definição do valor de λ.
4: Montagem de uma lista F com os resultados de f para cada entrada.
5: Ordenação crescente de F.
6: $medidas \leftarrow 1$
7: $cont \leftarrow 0$
8: $x \leftarrow \lfloor \frac{N}{2} \rfloor$
9: $mínimo \leftarrow F[x]$
10: $m \leftarrow 1$
11: **enquanto** $mínimo > F[1]$ **faça**
12: **repita**
13: $m \leftarrow m \cdot \lambda$
14: **se** $m > \sqrt{N}$ **então**
15: $m \leftarrow \sqrt{N}$
16: **fim se**
17: $j \leftarrow random(m)$
18: $cont \leftarrow cont + j$
19: $t \leftarrow$ quantidade de estados iguais a $F[1]$
20: $\theta \leftarrow asen(\sqrt{\frac{t}{N}})$
21: $prob \leftarrow sen^2(j \cdot \theta)$
22: **se** $prob \cdot 2^{31} > random(2^{31})$ **então**
23: $x \leftarrow random(t) + 1$
24: **senão**
25: $x \leftarrow random(N - t) + t + 1$
26: **fim se**
27: $y \leftarrow F[x]$
28: $medidas \leftarrow medidas + 1$
29: **até que** $y = F[1]$
30: **fim enquanto**
31: **devolva** y

se, dada uma instância do 3-Sat, k cláusulas são satisfeitas ou não.

O problema de otimização relacionado a esta variação do 3-Sat consiste em saber qual é o maior número de cláusulas que podem ser satisfeitas para as diversas combinações de valores para as variáveis em U.

Foram criadas instâncias deste problema de até $n = 16$ literais, o que implica em entradas de tamanho até $N = 2^{16}$. Foram rodadas 50 instâncias para cada tamanho de entrada, que variava entre as potências de 2 a partir 2^3 até 2^{16}. De posse de n, foi escolhido aleatoriamente, com distribuição uniforme, o número de cláusulas entre 1 e $max_n = \frac{n(n-1)(n-2)}{3}$, onde max_n é a quantidade máxima de cláusulas distintas possíveis para n literais. As cláusulas eram geradas também aleatoriamente: escolhiam-se, com distribuição uniforme, 3 literais distintos definindo também aleatoriamente se cada um destes literais seriam negados ou não. Em seguida foram removidas as cláusulas redundantes (iguais).

5.2 Simulação do problema de Equações Binárias Quadráticas

Assim como no caso do 3-Sat, foram montadas 50 instâncias para cada tamanho de entrada que variou entre 2^3 até 2^{16} para o problema de Equações Binárias Quadráticas (EBQ). Diferentemente do problema 3-Sat, a quantidade de entradas não precisaria ser potência de 2, mas para facilitar a comparação, foram usados apenas valores desta forma. Os parâmetros de cada problema são: α, β e γ. Deseja-se encontrar $x < \gamma$ que minimize a expressão $x^2 - \alpha \bmod \beta$. As instâncias foram montadas da seguinte forma: Escolhe-se um valor de γ. Escolhe-se aleatoriamente um valor $\beta > \gamma$, e em seguida, escolhe-se também aleatoriamente $\alpha < \beta$. Para cada instância, como no caso anterior, foram feitas 50 execuções de cada algoritmo, tomando a média de

iterações e de medições.

Nas figuras 5.1 e 5.2, são mostrados os gráficos comparativos do número de vezes que o Oráculo Desigualdade é chamado em cada um dos algoritmos para o problema das equações binárias quadráticas e para a versão de otimização do 3-Sat. Nos dois gráficos, ambas as escalas são logarítmicas. O valor médio obtido para o número de chamadas do oráculo no algoritmo DH96, em ambas versões, para cada tamanho de entrada é dado pelas linhas tracejadas e com o marcador no formato de diamante $\Diamond$, e o respectivo valor para o algoritmo Medida-Linear é dado pela linha inferior e que contém marcador no formato de triângulo $\triangle$. A linha com o resultado da simulação do algoritmo BBHT98 possui marcador $\bigcirc$. Para efeitos comparativos, são mostradas duas linhas adicionais sem marcadores. A linha superior se refere à função $5,74 \cdot 2^{n/2}$, onde n é o número de bits da entrada. Esta é a estimativa teórica para o algoritmo DH96 com $\lambda = \frac{4}{3}$. A outra linha se refere à função $4,133 \cdot 2^{n/2}$, número de passos estimados para o algoritmo Medida-Linear encontrar um mínimo absoluto do problema. No gráfico 5.2, é omitida a primeira linha $(5,74 \cdot 2^{n/2})$.

Nas figuras 5.3 e 5.4, são mostrados os gráficos comparativos do número de medições feitas por ambos algoritmos até chegar ao valor mínimo para os problemas EBQ e 3-Sat respectivamente. O gráfico está em função do número de bits da entrada.

5.3 Simulação de grandes instâncias

Um dos maiores problemas para a simulação de problemas NP-difíceis é a identificação da quantidade de estados marcados, ou seja, aqueles que têm a amplitude invertida em cada iteração. Esta informação é necessária para saber qual a probabilidade de um estado marcado ser observado. Depois, além da quantidade de estados marcados, é preciso saber quais são estes

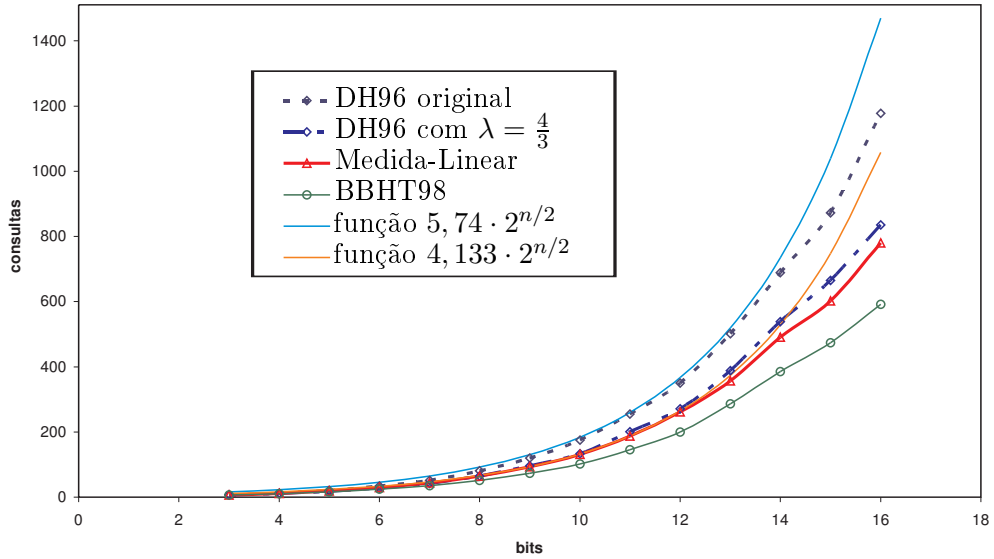

Figura 5.1: Gráfico com a quantidade de consultas ao oráculo dos algoritmos DH96 e Medida-Linear para o problema das Equações Binárias Quadráticas. As duas linhas sem marcadores se referem a estimativa teórica do número de chamadas dos dois algoritmos para otimização, sendo que a linha superior se refere ao DH96 enquanto que a outra linha se refere ao algoritmo Medida-Linear.

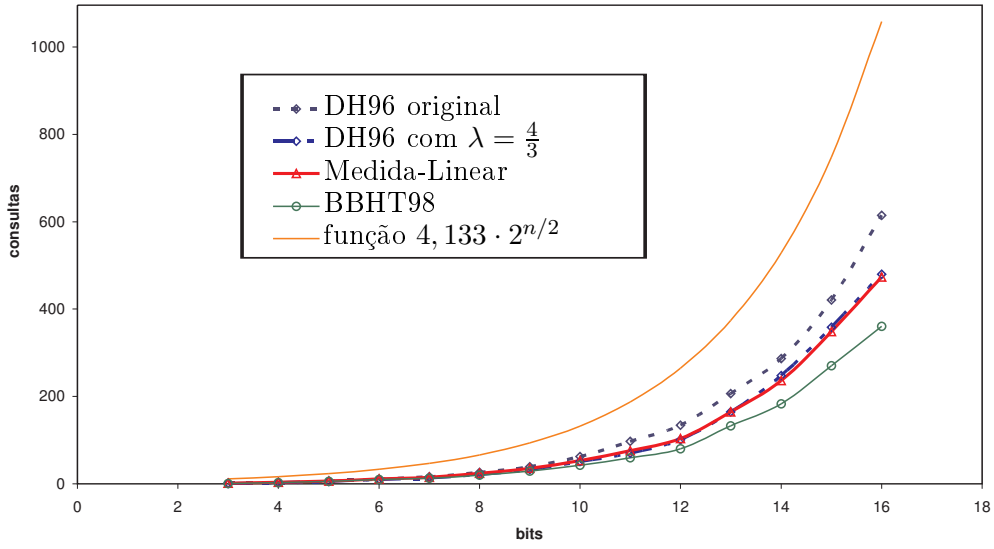

Figura 5.2: Gráfico com a quantidade de consultas ao oráculo dos algoritmos DH96 e Medida-Linear para o problema de otimização associado ao 3-Sat. A linha sem marcador se refere à estimativa teórica do número de chamadas do oráculo no algoritmo Medida-Linear.

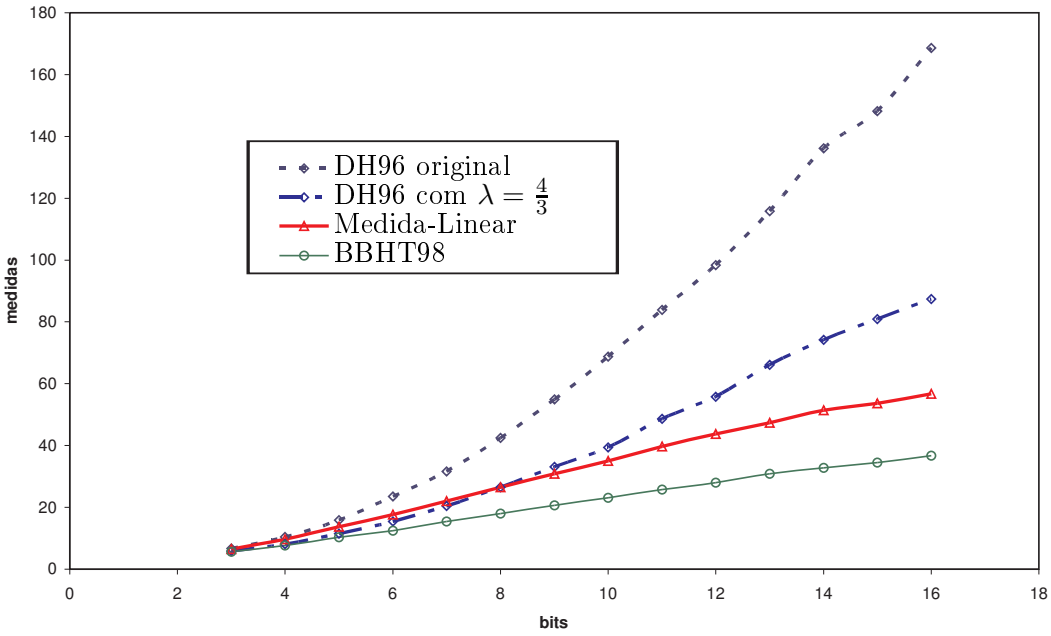

Figura 5.3: Gráfico com a quantidade de medições dos algoritmos DH96 e Medida-Linear (problema EBQ).

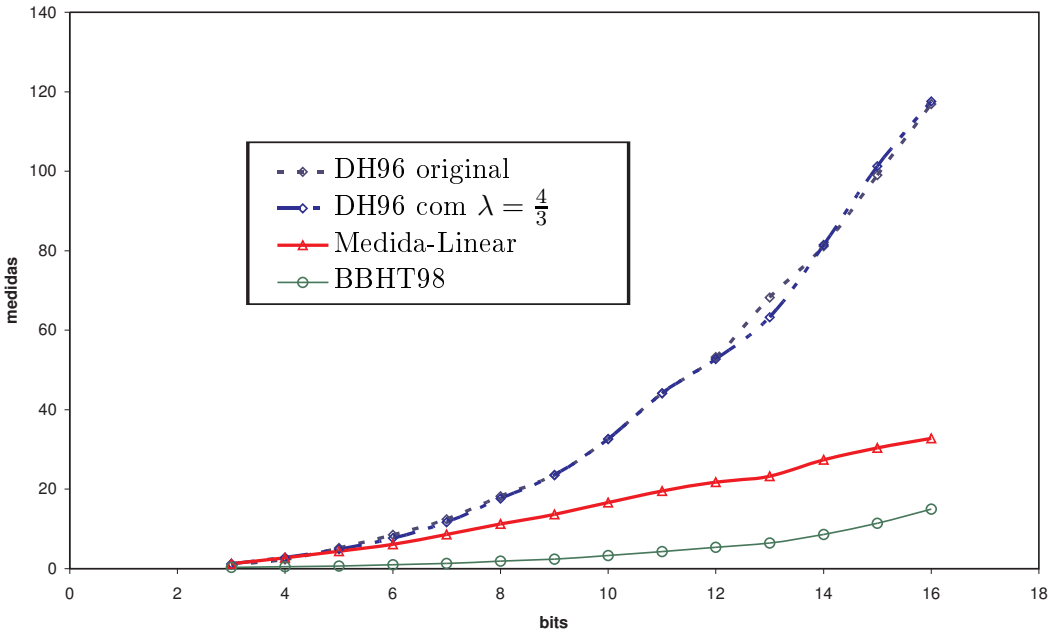

Figura 5.4: Gráfico com a quantidade de medições dos algoritmos DH96 e Medida-Linear (problema 3-Sat).

estados, pois cada valor medido é usado como cota para a rodada seguinte até chegar ao mínimo. Para facilitar a obtenção destas informações durante a simulação, o vetor com a seqüência de soluções é ordenado crescentemente de acordo com a imagem da função. Assim, as primeiras posições da lista contêm o menor valor da imagem. Se a quantidade de entradas possíveis N é muito grande, torna-se inviável a manipulação da lista: preenchimento, busca ou ordenação do mesmo. Uma solução alternativa é tomar uma função que já esteja ordenada. Optamos por tomar funções afim, ou seja, funções da forma: $f(x) = ax + b$ com x inteiro menor do que N, onde a e b são valores inteiros escolhidos aleatoriamente, com distribuição uniforme, dentro de um intervalo e N define o tamanho de entrada. Pelo fato de todos os elementos possuirem postos distintos, através da função afim, conseguimos simular o pior caso para ambos algoritmos, caso mais difícil inclusive do que os problemas NP-completos.

Para cada tamanho de entrada, foram escolhidas 50 funções distintas, sendo que para cada função destas, foram realizadas 50 execuções dos algoritmos 5.0.2 e 5.0.3. Foram tomadas as médias do número de iterações e da quantidade de medidas realizadas por cada algoritmo em cada execução.

O tamanho das entradas variou entre 2^3 e 2^{100}.

Como no caso dos problemas NP-difíceis, nas figuras 5.5 a 5.7, são mostrados os resultados das simulações para os algoritmos DH96 e Medida-Linear. São apresentadas duas versões para o DH96, uma com $\lambda = \frac{8}{7}$ e $\lambda = \frac{4}{3}$ como foi feito na seção anterior.

No gráfico 5.5, é feita uma comparação da quantidade de execuções do oráculo, que é o dobro do número de avaliações da função de otimização. Esta comparação é feita em uma escala logarítmica.

Como, teoricamente, o número de consultas é proporcional à raiz quadrada da dimensão da entrada (tamanho da lista ou tamanho do domínio da função), no gráfico 5.6, é mostrado o mesmo resultado do gráfico 5.5 dividido

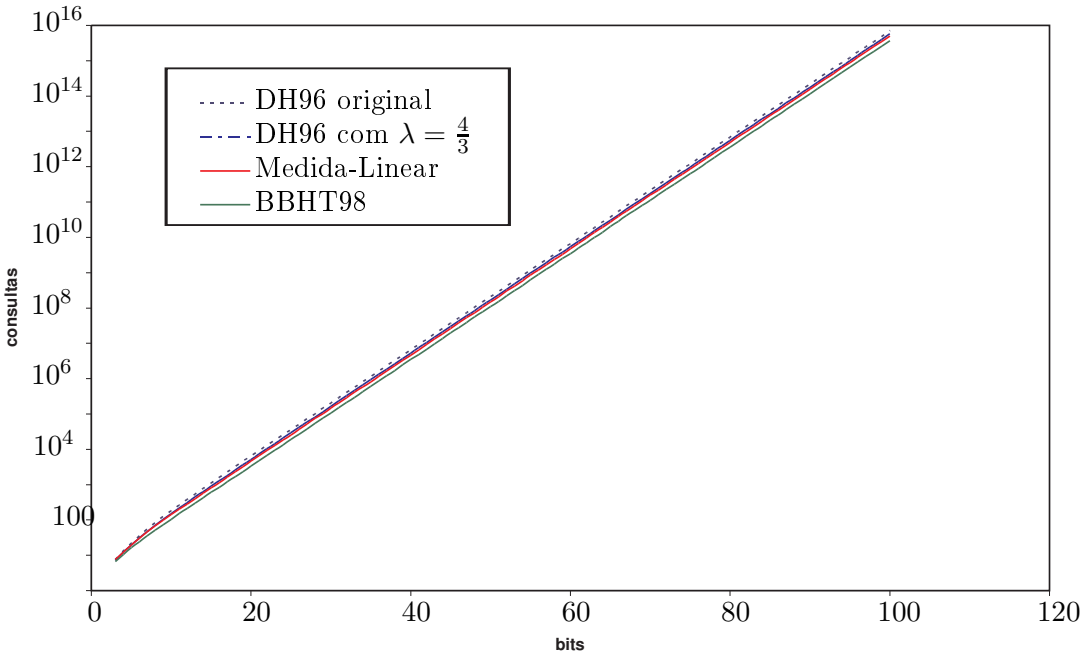

Figura 5.5: Gráfico com a quantidade de execuções do oráculo dos algoritmos DH96 e Medida-Linear para o caso em que todos os postos são distintos, em escala logarítmica.

pela raiz quadrada do tamanho da entrada, acrescentando também mais uma linha com a simulação do algoritmo Medida-Linear com $\lambda = \frac{8}{7}$. Isto ajuda a visualizar a diferença de performance dos algoritmos, visto que estes dados são apresentados em uma escala linear. Neste gráfico, pode-se observar que os dados, a partir de $n = 20$ bits, o número esperado de consultas ao oráculo é aproximadamente $6,25\sqrt{2^n}$ para o algoritmo DH96 original, $5,04\sqrt{2^n}$ para a versão DH96 com $\lambda = \frac{4}{3}$, $4,45\sqrt{2^n}$ para o algoritmo Medida-Linear com $\lambda = \frac{13}{12}$ e $4,05\sqrt{2^n}$ para $\lambda = \frac{8}{7}$.

No gráfico 5.7, é apresentada a quantidade de medições de ambos algoritmos para a simulação baseada na função afim.

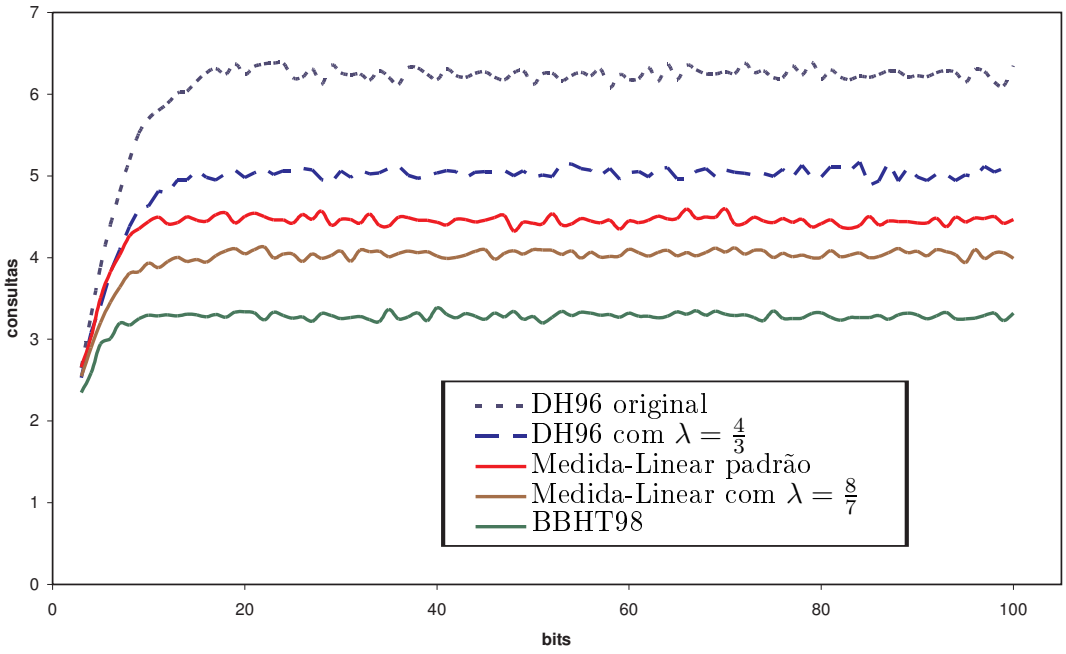

Figura 5.6: Gráfico com a quantidade de execuções do oráculo dos algoritmos DH96 e Medida-Linear dividido pela raiz quadrada do tamanho da entrada.

5.3.1 Simulação com λ variável

Como o desempenho do algoritmo DH96 foi diferente para valores distintos de λ, é natural nos perguntarmos qual é o melhor valor deste parâmetro e se há algum valor de λ que torne o algoritmo DH96 melhor do que o algoritmo Medida-Linear. Na descrição do algoritmo BBHT98 original, λ pode variar no intervalo $(1, \frac{4}{3})$, valores maiores podem diminuir o desempenho do algoritmo se houver múltiplas soluções. Mas no caso dos algoritmos para resolver problemas de otimização baseados no BBHT98, deve-se considerar sempre o pior caso (uma única solução) para finalizar o algoritmo, pois no caso da busca, ao obter o valor desejado, pode-se finalizar o algoritmo, mas no caso de otimização, não se pode ter certeza um mínimo (ou máximo) potencial encontrado é um mínimo (ou máximo absoluto).

O gráfico 5.8 mostra o desempenho de ambos algoritmos variando λ no intervalo $(1, 2)$.

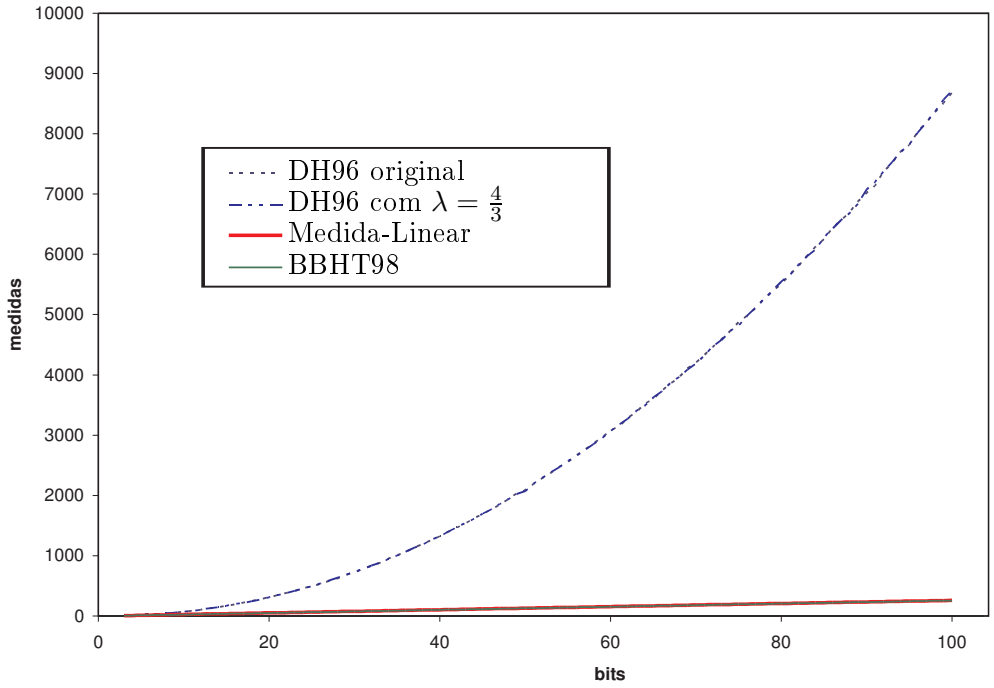

Figura 5.7: Gráfico com a quantidade de medições dos algoritmos DH96 e Medida-Linear para o caso em que todos os postos são distintos. As linhas referentes ao algoritmo DH96 com ambos valores de λ são quase coincidentes. O mesmo ocorre com as linhas referentes ao algoritmo Medida-Linear e o BBHT98.

Pelo gráfico 5.8, pode-se observar que para λ menor do que 1,41, o número de consultas do oráculo vai diminuindo para o algoritmo DH96, depois se mantém constante com pequenas oscilações, com o limite inferior igual a $4,75 \cdot 2^{20}$, e depois de 1,85, começa a subir lentamente. O gráfico do algoritmo Medida-Linear possui comportamento bem distinto, decrescendo até $\lambda = 8/7$, obtendo como limite inferior $3,98 \cdot 2^{20}$ e depois crescendo. O melhor desempenho para o algoritmo DH96 foi obtido na faixa em que ele foi simulado, já, no caso do algoritmo ML, não foi feita simulação com $\lambda = 8/7$, mas apenas com $\lambda = 13/12$, valor para o qual foi demonstrado, no capítulo anterior, o algoritmo funciona.

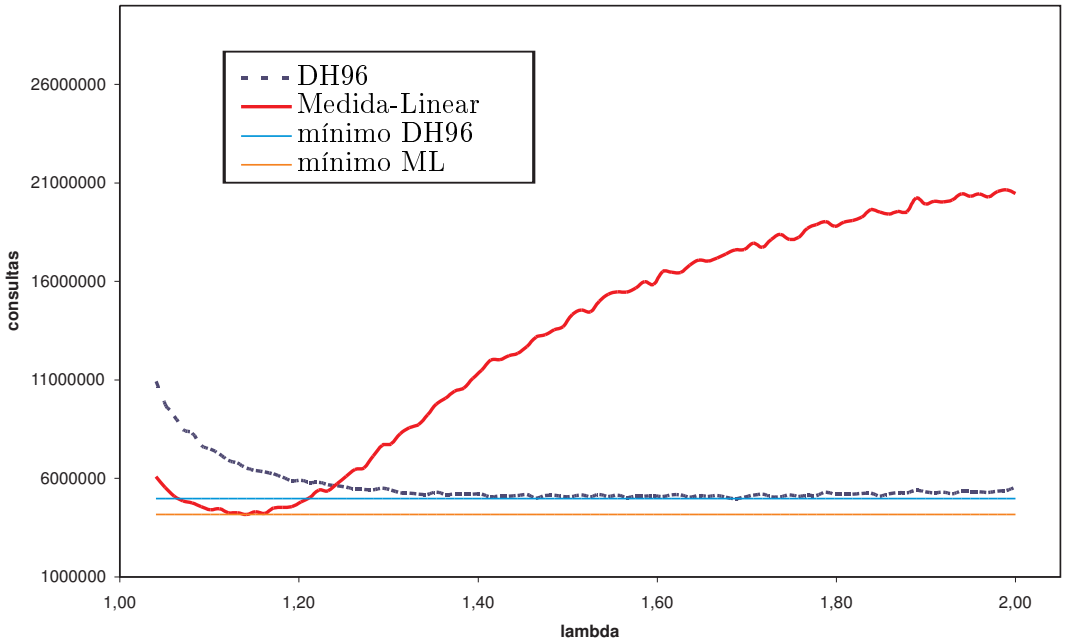

Figura 5.8: Gráfico com a variação de λ nos algoritmos DH96 e Medida-Linear para $N = 2^{40}$. As linhas paralelas indicam o menor número de operações em ambos algoritmos.

5.3.2 Simulação com múltiplas soluções

Até o momento, não foi considerado o caso de haver múltiplas soluções, ou seja, vários mínimos. No caso dos problemas tratados nas seções 5.1 e 5.2, em várias instâncias há mais de um mínimo, mas não foi feita uma análise destes casos.

É possível analisar o comportamento dos algoritmos DH96 e Medida-Linear, para problemas com várias soluções, modificando a função de otimização. Para as outras simulações, conforme foi comentado anteriormente, foram utilizadas funções da forma $f(x) = ax + b$, onde a e b eram escolhidos aleatoriamente, dentro de um intervalo, de acordo com uma distribuição uniforme. Para realizar simulação com múltiplas soluções, utilizamos a função

na forma

$$f(x) = \begin{cases} b & \text{se } x \leq t \\ ax + b & \text{se } t < x \leq N, \end{cases} \tag{5.1}$$

onde t é o número de mínimos e N é o tamanho do domínio da função.

As simulações foram feitas para $N = 2^{100}$ com t variando nas potências de 2, apartir de 1 até N. No caso do algoritmo DH96, é apresentada apenas a versão que apresenta melhor resultado: com $\lambda = \frac{4}{3}$.

No gráfico 5.9 é mostrada a quantidade de consultas ao oráculo para as simulações feitas.

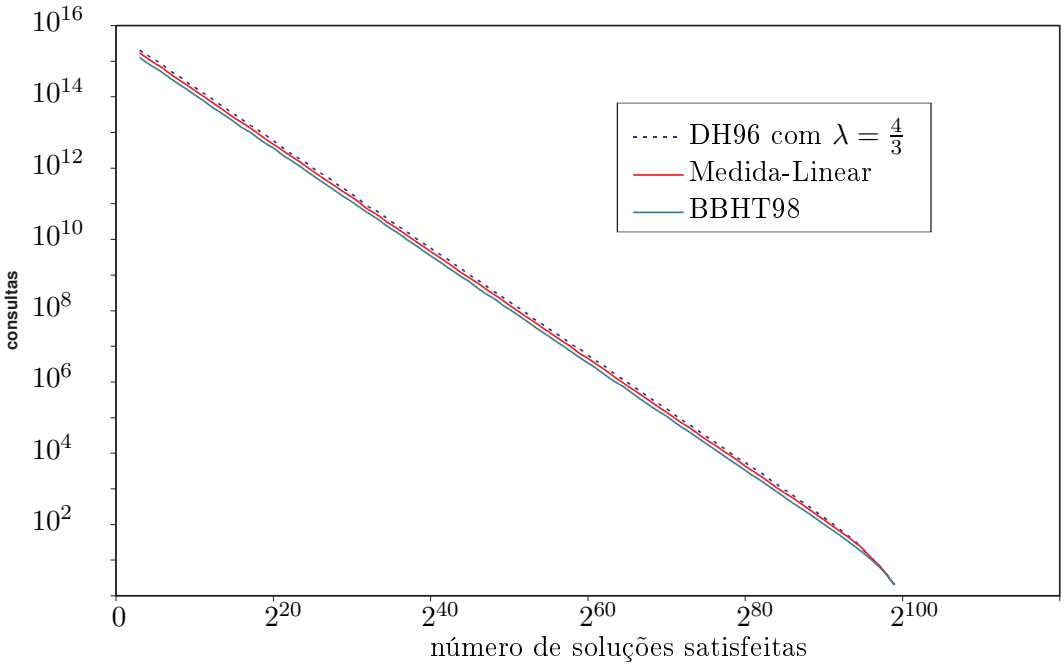

Figura 5.9: Gráfico com a quantidade de consultas ao oráculo dos algoritmos DH96 e Medida-Linear para o caso com vários mínimos. Ambas escalas são logarítmicas. Também é apresentado o resultado para o algoritmo BBHT98 para comparação.

No gráfico 5.10, também é mostrado o resultado da quantidade de execuções do oráculo de ambos algoritmos, mas multiplicado pela raiz quadrada de N/t.

O valor médio para o número de consultas ao oráculo para os algoritmos DH96, Medida-Linear e BBHT98 foram $5,16\sqrt{\frac{N}{t}}$, $4,27\sqrt{\frac{N}{t}}$ e $3,22\sqrt{\frac{N}{t}}$,

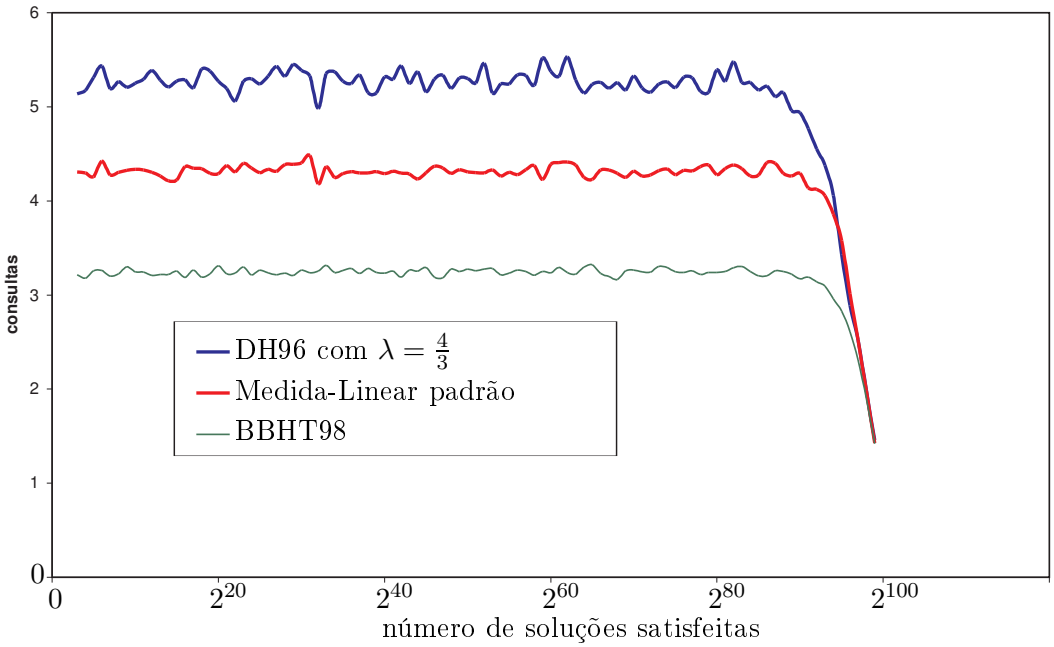

Figura 5.10: Gráfico com a quantidade de execuções do oráculo dos algoritmos DH96 e Medida-Linear dividido pela raiz quadrada do tamanho da entrada e multiplicado pela raiz quadrada do número de mínimos, para o caso de várias soluções para cada problema. O tamanho da entrada é 2^{100}.

respectivamente.

No gráfico 5.11, é mostrado o número de medições realizadas em relação ao número de mínimos, em escala logarítmica.

5.4 Análise dos resultados das simulações

Os gráficos para os três casos (os dois casos NP-difíceis e a função afim) foram bastante parecidos para as duas propriedades das simulações (número de execuções do oráculo e quantidade de medições).

A proporção de consultas à lista se manteve constante. Nos três casos, para todos tamanhos de entrada, o algoritmo Medida-Linear foi mais rápido do que o DH96.

O índice de falhas (caso em que o algoritmo não retorna o menor valor no

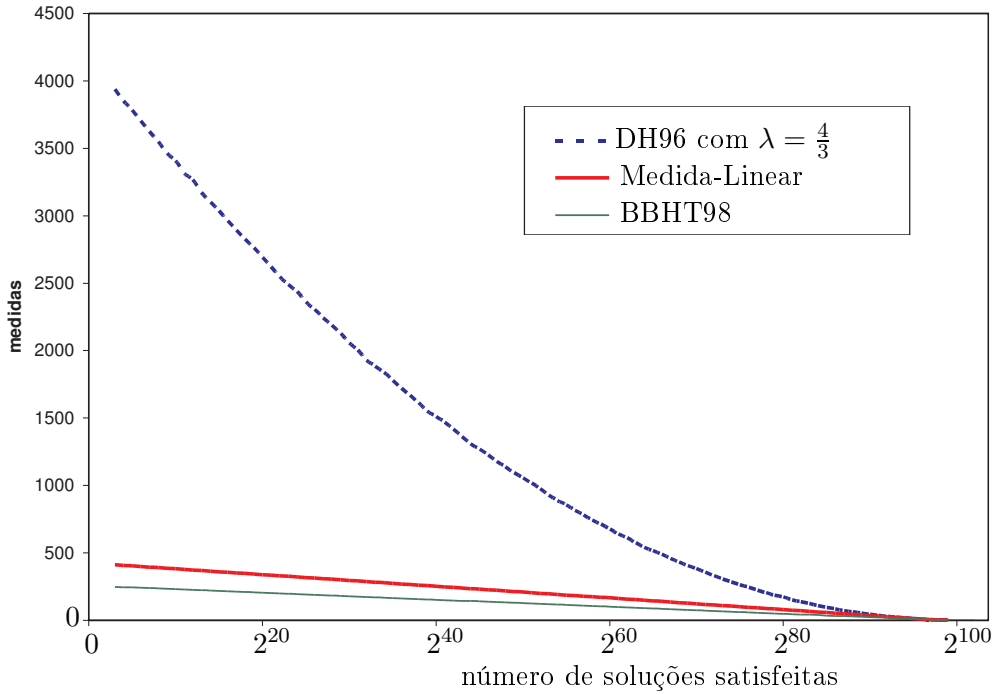

Figura 5.11: Gráfico com a quantidade de medições dos algoritmos DH96 e Medida-Linear para o caso em que todos os postos são distintos, com $N = 2^{100}$.

tempo esperado) esteve bem abaixo de 50%. Nos problemas 3-Sat e EQB, este índice foi quase nulo. Isto se deve a dois motivos: a quantidade de mínimos reais para cada problema t é maior do que um, e as instâncias das simulações são pequenas.

No caso das simulações com todos os elementos distintos, a taxa de erro no algoritmo DH96, foi de 19,44%, enquanto que no algoritmo Medida-Linear, o erro foi de 4,15%.

Em relação à quantidade de medições, pelo gráfico 5.7, pode-se confirmar a estimativa teórica que o algoritmo Medida-Linear faz $O(n)$ medições, enquanto o algoritmo DH96 realiza $O(n^2)$ medições, onde n é a quantidade de bits da entrada.

Pelo gráfico 5.11, pode se observar que os algoritmos para resolver problemas de otimização combinatória encontram uma solução em tempo $O(\sqrt{\frac{N}{t}})$, onde N é o tamanho do problema e t é o número de soluções. O problema é

que, em geral, não se conhece t de antemão, sendo necessário executar $\theta(N)$ iterações.

Capítulo 6

Conclusão

Neste trabalho, foram apresentados diversos circuitos úteis para a Computação Reversível Clássica e para a Computação Quântica, como é o caso dos circuitos para multiplicação e divisão de inteiros, que podem ser utilizados para a construção de diversos outros circuitos e algoritmos. Além do circuito convencional para multiplicação reversível, foram propostas versões reversíveis para o algoritmo Karatsuba. Em alguns destes circuitos foi utilizada uma forma de limpeza de lixo recursiva, que é distinta da encontrada na literatura.

Outra linha de algoritmos apresentados é relacionada ao problema de encontrar um valor ótimo em uma lista não ordenada ou retornado por uma função discreta. Propomos um algoritmo, chamado Medida-Linear, que se mostrou mais eficiente que o algoritmo DH96 [12], por fazer $8,27\sqrt{N}$ consultas a uma lista com N elementos em contraposição às $22,5\sqrt{N}$ consultas da descrição original do DH96. Além disso, chamando de n a quantidade de bits necessária para representar o tamanho da lista, o algoritmo ML executa $O(n)$ medições parciais enquanto que o algoritmo DH96 executa $\Omega(n^2)$ medições. Foi proposto também um operador, chamado Oráculo-Desigualdade, que permite transformar os algoritmos acima referidos em algoritmos de otimização combinatória.

Através de simulações, comparamos o desempenho dos algoritmos DH96 e ML, para encontrar um menor valor em uma lista gerada a partir de pequenas instâncias de problemas NP-difíceis e em problemas de otimização associados a funções injetoras, que é o pior caso para ambos algoritmos, com domínio de até 2^{100} elementos. Os resultados das simulações confirmaram o desempenho superior do algoritmo ML em relação ao algoritmo DH96.

Tanto em relação aos algoritmos aritméticos reversíveis, quanto em relação aos algoritmos quânticos para resolver o problema de otimização, há vários aspectos que podem ser estudados para tentar melhorar o desempenho.

O algoritmo Karatsuba reversível é mais eficiente do que o algoritmo convencional por fazer $\Theta(n^{\log 3})$ operações enquanto o algoritmo padrão é $\Theta(n^2)$, mas em compensação, ele utiliza um espaço extra maior. Pode-se tentar otimizá-lo de forma a gastar menos espaço, mas sem aumentar muito a complexidade do circuito. Mostramos como fazer a limpeza do lixo recursivo. Pode-se tentar fazer a limpeza de lixo para outros tipos de algoritmos nos quais a aplicação dos métodos de limpeza de Bennett seja pouco eficiente ou de difícil implementação.

Em relação ao problema de otimização combinatória, não é possível diminuir a ordem de complexidade do número de chamadas do oráculo para os algoritmos DH96 e Medida-Linear, visto que, para entrada de tamanho N, estes dois algoritmos são $\Theta(\sqrt{N})$, limite inferior do problema de busca quando há apenas um mínimo absoluto. E no caso de problemas de otimização mesmo tendo obtido o valor desejado, caso sejam executadas menos operações do que $\Theta(\sqrt{N})$, não se pode ter certeza que se chegou a um mínimo absluto. Neste caso, considerando que a complexidade do circuito é $\Theta(\sqrt{N}) \cdot \Theta(f(N))$, onde f é a função de otimização, a única maneira de otimizar o algoritmo é aplicá-lo a uma função de otimização que seja mais eficiente, como por exemplo, meta-heurísticas, que são estratégias gerais que guiam outras heurísticas em busca de soluções satisfatórias em problemas

difíceis. E como é possível fazer a simulação dos algoritmos ML e DH96 em computadores clássicos, pode-se compará-los com outros procedimentos que existem atualmente para resolver problemas de otimização combinatória.

Referências Bibliográficas

[1] AMBAINIS, A., "Quantum search algorithms". www.arxiv.org, april 2005. quant-ph/0504012.

[2] BECHMAN, D., CHARI, A. N., DEVABHAKTUNI, S., *et al.*, "Efficient networks for quantum factoring", *Phys. Rev. A*, v. 54, pp. 1034–1063, 1996.

[3] BENNETT, C. H., "The logical reversibility of computation", *IBM J. Res. Develop.*, v. 17, pp. 525–532, 1973.

[4] BENNETT, C. H., "Time/Space trade-offs for reversible computation", *SIAM J. Comput.*, v. 18, n. 4, pp. 766–776, 1989.

[5] BENNETT, C. H., BRASSARD, G., BERNSTEIN, E., *et al.*, "Strengths and weaknesses of quantum computing", *SIAM Journal on Computing*, v. 26, n. 5, pp. 1510–1523, 1997.

[6] BOYER, M., BRASSARD, G., HØYER, P., *et al.*, "Tight bounds on quantum searching", *Fortschritte Der Physik*, 1998.

[7] BRASSARD, G., HØYER, P., "An exact quantum polynomial algorithm for Simon's problem", In: *Proceedings of the Fifth Israeli Symposium on Theory of Computing and Systems(ISTCS'97)*, pp. 12–23, 1997.

[8] CHURCH, A., "An unsolvable problem of elementary number theory", *Am. J. Math.*, v. 58, 1936.

[9] CUCCARO, S. A., DRAPER, T. G., KUTIN, S. A., *et al.*, "A new quantum ripple-carry addition circuit". www.arxiv.org, 2004. quant-ph/0410184.

[10] DRAPER, T. G., "Addition on a quantum computer". www.arxiv.org, 2000. quant-ph/0008033.

[11] DRAPER, T. G., KUTIN, S. A., RAINS, E. M., *et al.*, "A logarithmic-depth quantum carry-lookahead". www.arxiv.org, 2004. quant-ph/0406142.

[12] DÜRR, C., HØYER, P., "A quantum algorithm for finding the minimum". www.arxiv.org, january 1999. quant-ph/9607014v2.

[13] FLORIO, G., PICCA, D., "Quantum implementation of elementary arithmetic operations". www.arxiv.org, 2004. quant-ph/0403048.

[14] GAREY, M. R., JOHNSON, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[15] GATHEN, J., GERHARD, J., *Modern Computer Algebra.* second ed. Cambridge, UK, Cambridge University Press, 2003.

[16] GOSSETT, P., "Quantum carry-save arithmetic". www.arxiv.org, 1998. quant-ph/9808061.

[17] GROVER, L. K., "Quantum mechanics helps in searching for a needle in a haystack", *Physical Review Letter*, v. 79, pp. 325–329, 1997.

[18] KARATSUBA, A., OFMAN, Y., "Multiplication of multidigit numbers on automata", *Soviet Physics – Doklady*, v. 7, n. 7, pp. 595–596, 1963.

[19] KAYE, P., ZALKA, C., "Optimized quantum implementation of elliptic curve arithmetic over binary fields". www.arxiv.org, 2004. quant-ph/0407095.

[20] KIMBLE, H., CALLAN JR., C., CASE, K., *et al.*, "Quantum Computing", jason report jsr-95-115, The MITRE Corporation, 1996.

[21] KLEENE, S., "A theory of positive integers in formal logic", *American Journal of Mathematics*, v. 57, pp. 153–173,219–244, 1935.

[22] KNUTH, D. E., *The Art of Computer Programming. vol 2.* 3a ed. ed. Addison-Wesley Publishing Company, 1997.

[23] KOWADA, L. A. B., PORTUGAL, R., DE FIGUEIREDO, C. M. H., "Reversible Karatsuba algorithm". submetido para Journal of Universal Computer Science.

[24] LANDAUER, R., "Irreversibility and heat generation in the computing process", *IBM J. Res. Develop.*, v. 5, pp. 183–191, 1961.

[25] LAVOR, C., MANSSUR, L. R. U., PORTUGAL, R., "Shor's algorithm for factoring large integer", 2003. quant-ph/0303175.

[26] LECERF, Y., "Machines de Turing réversibles. Récursive insolubilité en $n \in N$ de l'équation $u = \theta^n u$, où θ est un isomorphisme de codes", *C. R. Acad. Française Sci.*, v. 257, pp. 2597–2600, 1963.

[27] LEVINE, R., SHERMAN, A. T., "A note on Bennett's time-space tradeoff for reversible computation", *SIAM J. Comput.*, v. 19, n. 4, pp. 673–677, 1990.

[28] LI, M., TROMP, J., VITÁNYI, P., "Reversible simulation of irreversible computation", *Physica D*, v. 120, pp. 168–176, 1998.

[29] LOMONT, C., "A quantum fourier transform algorithm". www.arxiv.org, 2004. quant-ph/0404060.

[30] MANDERS, K. L., ADLEMAN, L., "NP-Complete decision problems for binary quadratics", *Journal of Computer and System Sciences*, v. 16, pp. 168–184, 1978.

[31] MANSSUR, L. R. U., PORTUGAL, R., "Stochastic simulation of quantum computation", *Europhysics Letters*, v. 63, n. 4, pp. 492–497, 2003.

[32] METER, R. V., ITOH, K. M., "Fast quantum modular exponentiation", *Physical Review A*, v. 71, n. 5, p. 052320, 2005.

[33] MIHARA, T., SUNG, S. C., "Deterministic polynomial-time quantum algorithms for simon's problem", In: *Computational Complexity*, v. 12, pp. 162–175, Birkhäuser Verlag AG, 2003.

[34] NIELSEN, M. A., CHUANG, I. L., *Quantum Computation and Quantum Information*. Cambrigde - UK, Cambridge Press, 2000.

[35] OLIVEIRA, I., SARTHOUR, R. S., BULNES, J. D., *et al.*, "Computação quântica", *Ciência Hoje*, v. 33, pp. 22–29, May 2003.

[36] PORTUGAL, R., LAVOR, C., CARVALHO, L. M., *et al.*, *Uma Introdução à Computação Quântica*, v. 8 of *Notas em Matemática Aplicada*. SBMAC, 2004.

[37] PRESKILL, J., *Lecture Notes for Physics 229: Quantum Information and Computation*. September 1998.

[38] SCHÖNHAGE, A., STRASSEN, V., "Schnelle multiplikation grosser zahlen", *Computing*, v. 7, pp. 281–292, 1971.

[39] SHENDE, V. V., PRASAD, A. K., MARKOV, I. L., *et al.*, "Synthesis of reversible logic circuits", *IEEE Trans. on CAD*, v. 22, n. 6, pp. 710–722, 2003.

[40] SHOR, P., "Algorithms of quantum computation: discrete logarithm and factoring", In: *Proceedings of 35th Annual Symposium on Foundations of Computer Science*, pp. 124–134, 1994.

[41] SHOR, P., "Quantum information theory: results and open problems", *Geom. Funct. Anal.*, v. Special Volume, pp. 816–838, 2000.

[42] SHOR, P., "Progress in quantum algoritms", *Quantum Information Processing*, v. 3, pp. 5–10, October 2004.

[43] SIMON, D., "On the power of quantum computation", *SIAM J. Comp.*, v. 26, n. 5, pp. 1474–1483, 1997.

[44] The PARI Group, Bordeaux, *PARI/GP, Version 2.2.10*, 2004. available from `http://pari.math.u-bordeaux.fr/`.

[45] TURING, A. M., "On computable numbers, with an application to Entscheidungs problem", In: *Proc. Lond. Math. Soc. 2*, v. 42, p. 230, 1936.

[46] VEDRAL, V., BARENCO, A., EKERT, A., "Quantum networks for elementary arithmetic operations", *Physical Review A*, v. 54, n. 1, pp. 147–153, 1996.

[47] ZALKA, C., "Fast version of Shor's quantum factoring algorithm", 1998. quant-ph/9806084.

[48] ZALKA, C., "Grover's quantum searching algorithm is optimal", *Physical Review A*, v. 60, pp. 2746–2751, 1999.