



**Universidade do Estado do Rio de Janeiro**

Centro de Tecnologia e Ciências

Faculdade de Engenharia

Marcos Paulo Mello Araujo

**Síntese evolucionária de circuitos seqüenciais  
inspirada nos princípios da computação quântica**

Rio de Janeiro  
2008

# **Livros Grátis**

<http://www.livrosgratis.com.br>

Milhares de livros grátis para download.

Marcos Paulo Mello Araujo

**Síntese evolucionária de circuitos seqüenciais  
inspirada nos princípios da computação quântica**



Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof<sup>a</sup>. Dr<sup>a</sup>. Nadia Nedjah

Co-orientadora: Prof<sup>a</sup>. Dr<sup>a</sup>. Luiza de Macedo Mourelle

Rio de Janeiro  
2008

CATALOGAÇÃO NA FONTE  
UERJ/REDE SIRIUS/CTC/B

A663 Araujo, Marcos Paulo Mello.

Síntese evolucionária de circuitos seqüenciais inspirada nos princípios da computação quântica/Marcos Paulo Mello Araujo. – 2008.

140 f. : il.

Orientadora: Nadia Nedjah.

Co-orientadora: Luiza de Macedo Mourelle.

Dissertação (mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

Bibliografia: f. 110 – 118.

1. Computação quântica – Teses. 2. Sistemas digitais. 3. Algoritmos. I. Nedjah, Nadia. II. Universidade do Estado do Rio de Janeiro. Faculdade de Engenharia. III. Título.

CDU 004:530.145

Autorizo, apenas para fins acadêmicos e científicos, a reprodução total ou parcial desta dissertação.

---

Assinatura

---

Data

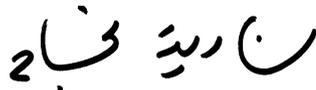
Marcos Paulo Mello Araujo

## Síntese evolucionária de circuitos seqüenciais inspirada nos princípios da computação quântica

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em 4 de Dezembro de 2008

Banca Examinadora:



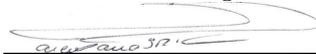
---

Prof<sup>a</sup>. Dr<sup>a</sup>. Nadia Nedjah (Orientadora)  
Faculdade de Engenharia, UERJ



---

Prof<sup>a</sup>. Dr<sup>a</sup>. Luiza de Macedo Mourelle (Co-orientadora)  
Faculdade de Engenharia, UERJ



---

Prof<sup>a</sup>. Dr<sup>a</sup>. Marley Maria Bernardes Rebuszi Vellasco  
Departamento de Engenharia Elétrica, PUC-Rio



---

Prof. Dr. Leandro dos Santos Coelho  
Programa de Pós-Graduação em Engenharia de Produção e Sistemas, PUC-PR

Rio de Janeiro  
2008

# DEDICATÓRIA

À minha amada esposa Christiane e a minha filha Luísa  
que me acompanharam e ajudaram a manter a motivação  
nos momentos mais difíceis desta longa jornada.

# AGRADECIMENTOS

A Deus por me permitir amar e sonhar;

Aos meus amados pais, Vicente e Suely, por tudo que proporcionaram na minha vida e por compreenderem a minha impossibilidade de visitá-los aos finais de semana;

À minha querida esposa Christiane pelo carinho, paciência e apoio incondicional;

À minha filha Luísa por bater na porta do quarto chamando o papai para um momento de distração;

À Universidade do Estado do Rio de Janeiro - UERJ por ter me recebido novamente, desta vez na qualidade de aluno de mestrado;

Aos funcionários e professores do Programa de Pós-graduação em Engenharia Eletrônica - PEL, pela ajuda e ensinamentos;

Às minhas orientadoras, professora Nadia Nedjah e professora Luiza de Macedo Mourelle, pelos ensinamentos, sugestões, correções, opiniões, conselhos, e principalmente pela paciência que permitiram a realização desta obra;

À Petróleo Brasileiro S.A. - PETROBRAS por ter permitido a realização desta pós-graduação com liberação parcial de expediente, particularmente ao gerente da Engenharia de Abastecimento - EAB, Vicente Gullo, por garantir esta liberação;

Ao gerente do setor Automação Industrial e Integração de Sistemas - AIIS, Gilberto Bartz, pela oportunidade, confiança, incentivo, apoio e compreensão;

Aos amigos do AIIS, em especial Saulo e Daniel, pelo apoio e pela ajuda nos momentos da minha ausência;

Aos colegas do mestrado, Joaquim, Daniel, Renato, André, Marcus Vinícius, Rodrigo e Ruben, pela força.

# RESUMO

ARAÚJO, Marcos Paulo Mello. *Síntese evolucionária de circuitos seqüenciais inspirada nos princípios da computação quântica*. 2008. 140f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2008.

Esta dissertação investiga a aplicação dos algoritmos evolucionários inspirados na computação quântica na síntese de circuitos seqüenciais. Os sistemas digitais seqüenciais representam uma classe de circuitos que é capaz de executar operações em uma determinada seqüência. Nos circuitos seqüenciais, os valores dos sinais de saída dependem não só dos valores dos sinais de entrada como também do estado atual do sistema. Os requisitos cada vez mais exigentes quanto à funcionalidade e ao desempenho dos sistemas digitais exigem projetos cada vez mais eficientes. O projeto destes circuitos, quando executado de forma manual, se tornou demorado e, com isso, a importância das ferramentas para a síntese automática de circuitos cresceu rapidamente. Estas ferramentas conhecidas como ECAD (*Electronic Computer-Aided Design*) são programas de computador normalmente baseados em heurísticas. Recentemente, os algoritmos evolucionários também começaram a ser utilizados como base para as ferramentas ECAD. Estas aplicações são referenciadas na literatura como eletrônica evolucionária. Os algoritmos mais comumente utilizados na eletrônica evolucionária são os algoritmos genéticos e a programação genética. Este trabalho apresenta um estudo da aplicação dos algoritmos evolucionários inspirados na computação quântica como uma ferramenta para a síntese automática de circuitos seqüenciais. Esta classe de algoritmos utiliza os princípios da computação quântica para melhorar o desempenho dos algoritmos evolucionários. Tradicionalmente, o projeto dos circuitos seqüenciais é dividido em cinco etapas principais: *(i)* Especificação da máquina de estados; *(ii)* Redução de estados; *(iii)* Atribuição de estados; *(iv)* Síntese da lógica de controle e *(v)* Implementação da máquina de estados. O Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ) proposto neste trabalho é utilizado na etapa de atribuição de estados. A escolha de uma atribuição de estados ótima é tratada na literatura como um problema ainda sem solução. A atribuição de estados escolhida para uma determinada máquina de estados tem um impacto direto na complexidade da sua lógica de controle. Os resultados mostram que as atribuições de estados obtidas pelo AEICQ de fato conduzem à implementação de circuitos de menor complexidade quando comparados com os circuitos gerados a partir de atribuições obtidas por outros métodos. O AEICQ é utilizado também na etapa de síntese da lógica de controle das máquinas de estados. Os circuitos evoluídos pelo AEICQ são otimizados segundo a área ocupada e o atraso de propagação. Estes circuitos são compatíveis com os circuitos obtidos por outros métodos e em alguns casos até mesmo superior em termos de área e de desempenho, sugerindo que existe um potencial de aplicação desta classe de algoritmos no projeto de circuitos eletrônicos.

**Palavras-chave:** computação quântica, computação evolucionária, sistemas digitais, sistemas seqüenciais, máquina de estados, atribuição de estados, eletrônica evolucionária.

# ABSTRACT

This thesis investigates the application of quantum inspired evolutionary algorithms in the synthesis of sequential circuits. Sequential digital systems represent a class of circuit that is able to execute operations in a particular sequence. In sequential circuits, the values of output signals not only depend on the values of input signals but also on the current state of the system. The increasingly high requirements regarding the functionality and performance of digital systems demand more efficient designs. The design of these circuits, when implemented manually, became slow and thus the importance of tools for automatic synthesis of circuits grew rapidly. These tools known as ECAD (Electronic Computer-Aided Design) are computer programs usually based on heuristics. Recently, evolutionary algorithms also began to be used as a basis in ECAD tools developing. These applications are referenced in literature as evolutionary electronics. The algorithms most commonly used in evolutionary electronics are genetic algorithms and genetic programming. This work presents a study of the application of quantum inspired evolutionary algorithms as a tool for automatic synthesis of sequential circuits. This class of algorithms uses the principles of quantum computing to improve the performance of evolutionary algorithms. Traditionally, the design of sequential circuits is divided into five main steps: *(i)* State machine specification; *(ii)* Reduction of states; *(iii)* State assignment; *(iv)* Control logic synthesis and *(v)* Implementation of the state machine. The proposed algorithm AEICQ is used in the state assignment design step. The choice of an optimal state assignment is treated in the literature as an issue still unresolved. The state assignment chosen for a particular state machine has a direct impact on the complexity of its control logic. The results show that the state assignment obtained by AEICQ in fact leads to the implementation of circuits of less complexity when compared with the ones generated from assignments obtained by other methods. The AEICQ is also used in the control logic synthesis of the state machine. The circuits evolved by AEICQ are optimized according to the area occupied and the propagation delay. These circuits are compatible with the circuits obtained by other methods and in some cases even higher in terms of area and performance, suggesting that there is a potential for application of this class of algorithms in the design of electronic circuits.

**Keywords:** quantum computation, evolutionary computing, digital systems, sequential systems, state machine, state assignment, evolutionary electronics.

## LISTA DE FIGURAS

|    |  |    |
|----|--|----|
| 1  | Arquitetura genérica de um sistema seqüencial . . . . .  | 11 |
| 2  | Exemplo de um diagrama de transição de estados . . . . .   | 12 |
| 3  | Mapa de <i>Karnaugh</i> da função de saída . . . . .   | 14 |
| 4  | Mapas de transição dos <i>flip-flops</i> $W$ e $X$ . . . . .   | 15 |
| 5  | Exemplo de um circuito esquemático de um sistema seqüencial . . . . .  | 15 |
| 6  | Metodologia para o projeto de máquinas de estados finitos . . . . .  | 17 |
|    |  |    |
| 7  | Dinâmica dos algoritmos evolucionários . . . . .   | 21 |
| 8  | Fluxograma básico de um algoritmo genético . . . . .   | 23 |
| 9  | Exemplo de cromossomos com codificação binária . . . . .   | 24 |
| 10 | Exemplo de cromossomos codificados com permutação . . . . .  | 24 |
| 11 | Seleção pelo giro da roleta . . . . .  | 25 |
| 12 | Seleção por torneio . . . . .  | 26 |
| 13 | Exemplo da operação de cruzamento simples . . . . .  | 27 |
| 14 | Tipos de operadores cruzamento . . . . .   | 28 |
| 15 | Operação de mutação . . . . .  | 29 |
| 16 | Uma estrutura em árvore simples da programação genética . . . . .  | 33 |
| 17 | Ilustração da operação de cruzamento . . . . .   | 34 |
| 18 | Ilustração da operação de mutação . . . . .  | 35 |
|    |  |    |
| 19 | Representação polar da porta rotação . . . . .   | 45 |
|    |  |    |
| 20 | Mapas de transição dos <i>flip-flops</i> $W$ e $X$ para a <i>Atribuição</i> <sub>1</sub> . . . . .   | 58 |
| 21 | Mapa de <i>Karnaugh</i> da função de saída para a <i>Atribuição</i> <sub>1</sub> . . . . .   | 58 |
| 22 | Esquemático da máquina de estados utilizando <i>Atribuição</i> <sub>1</sub> . . . . .  | 59 |
| 23 | Mapas de transições dos <i>flip-flops</i> $W$ e $X$ para <i>Atribuição</i> <sub>2</sub> . . . . .  | 59 |
| 24 | Mapa de <i>Karnaugh</i> da função de saída para <i>Atribuição</i> <sub>2</sub> . . . . .   | 60 |
| 25 | Esquemático da máquina de estados utilizando <i>Atribuição</i> <sub>2</sub> . . . . .  | 60 |
| 26 | Exemplo da codificação para o problema de atribuição de estados e duas possíveis observações dos estados dos $q$ -bits . . . . .           | 64 |
| 27 | Exemplo ilustrativo das simplificações permitidas pela aplicação da primeira regra . . . . .   | 65 |
| 28 | Exemplo ilustrativo das simplificações permitidas pela aplicação da segunda regra . . . . .  | 66 |
| 29 | Exemplo de uma matriz de adjacências . . . . .   | 67 |
| 30 | Exemplo de um $q$ -indivíduo na geração inicial . . . . .  | 68 |
| 31 | $q$ -indivíduo após a operação de atualização das probabilidades . . . . .   | 68 |
| 32 | Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados <i>bbara</i> e <i>bbsse</i> . . . . .   | 72 |
| 33 | Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados <i>dk14</i> e <i>dk16</i> . . . . .     | 72 |
| 34 | Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados <i>donfile</i> e <i>lion9</i> . . . . . | 73 |

---

|    |   |     |
|----|---|-----|
| 35 | Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados <i>modulo12</i> e <i>shiftreg</i> . . . . .                                  | 73  |
| 36 | Progresso da aptidão da melhor solução e da aptidão média da população para a máquina de estados <i>train11</i> . . . . .   | 74  |
| 37 | Comparação das lógicas de controle em termos da quantidade de portas . . . . .  | 74  |
| 38 | Comparação das lógicas de controle em termos de área consumida . . . . .  | 77  |
| 39 | Comparação das lógicas de controle em termos de atraso de propagação . . . . .  | 77  |
| 40 | Observação do efeito da etapa de limitação das amplitudes de probabilidade . . . . .  | 78  |
| 41 | Demonstração do efeito da etapa de migração global . . . . .  | 79  |
| 42 | Metodologia proposta para a síntese evolucionária de máquinas de estados . . . . .  | 88  |
| 43 | Exemplo de uma matriz de células . . . . .  | 88  |
| 44 | Exemplo da codificação de uma célula com 3 q-bits para a seleção das portas e 4 q-bits para a seleção dos sinais de cada entrada . . . . .                                      | 89  |
| 45 | Exemplo do circuito codificado por uma célula . . . . .   | 90  |
| 46 | Poder de representação do AEICQ . . . . .   | 91  |
| 47 | Exemplo de um circuito evoluído . . . . .   | 94  |
| 48 | Comparação das lógicas de controle em termos da quantidade de portas . . . . .  | 101 |
| 49 | Comparação das lógicas de controle em termos de área consumida . . . . .  | 101 |
| 50 | Comparação das lógicas de controle em termos de atraso de propagação . . . . .  | 102 |
| 51 | Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados <i>bbara</i> e <i>bbtas</i> . . . . .       | 103 |
| 52 | Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados <i>dk15</i> e <i>dk27</i> . . . . .         | 103 |
| 53 | Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados <i>dk512</i> e <i>lion9</i> . . . . .       | 104 |
| 54 | Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados <i>modulo12</i> e <i>shiftreg</i> . . . . . | 104 |
| 55 | Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados <i>tav</i> e <i>train11</i> . . . . .       | 105 |

## LISTA DE TABELAS

|    |   |     |
|----|---|-----|
| 1  | Exemplo de uma tabela de transição de estados . . . . .   | 13  |
| 2  | Tabela de transição de estados considerando a codificação binária . . . . .   | 13  |
| 3  | Definição do ângulo $\Delta\theta_i$ . . . . .  | 46  |
| 4  | Quantidade de atribuições de estados diferentes . . . . .   | 56  |
| 5  | Tabela de estados com duas atribuições de estados possíveis . . . . .   | 57  |
| 6  | Tabela de transição de estados para <i>Atribuição</i> <sub>01</sub> . . . . .   | 57  |
| 7  | Tabela de transição de estados para <i>Atribuição</i> <sub>02</sub> . . . . .   | 58  |
| 8  | Comparação do número de portas necessárias para várias atribuições, inclusive <i>Atribuição</i> <sub>01</sub> e <i>Atribuição</i> <sub>02</sub> . . . . .                             | 61  |
| 9  | Características das máquinas de estados . . . . .   | 69  |
| 10 | Melhor atribuição de estados obtida por cada um dos algoritmos . . . . .  | 71  |
| 11 | Aptidão das melhores atribuições obtidas pelos algoritmos . . . . .   | 71  |
| 12 | Melhor atribuição de estados obtida por cada um dos métodos . . . . .   | 75  |
| 13 | Comparação das lógicas de controle utilizando as atribuições obtidas pelo AG <sub>2</sub> , AG <sub>3</sub> e NOVA <sup>TM</sup> . . . . .  | 76  |
| 14 | Comparação das lógicas de controle utilizando as atribuições obtidas pelo AEICQ . . . . .   | 76  |
| 15 | Grau de liberdade, valores de $\chi^2$ calculado, valores críticos de $\chi^2$ para o nível de confiança 99,5% e níveis de confiança obtidos para as diferentes comparações . . . . . | 78  |
| 16 | Portas utilizadas pelo AEICQ e seus respectivos códigos . . . . .   | 90  |
| 17 | Portas utilizadas pelo AEICQ e seus respectivos valores de atraso de propagação, fator de carga e área . . . . .  | 95  |
| 18 | Valores de $\alpha$ e $\beta$ de cada uma das portas lógicas utilizadas pelo AEICQ . . . . .  | 96  |
| 19 | Atribuições de estados utilizadas para a síntese das lógicas de controle . . . . .  | 98  |
| 20 | Características dos circuitos gerados por PG, AG <sub>3</sub> e ABC . . . . .   | 100 |
| 21 | Resultados experimentais do AEICQ . . . . .   | 100 |
| 22 | Grau de liberdade, valores de $\chi^2$ calculado, valores críticos de $\chi^2$ para o nível de confiança 99,5% e níveis de confiança obtidos para as diferentes comparações . . . . . | 102 |

## LISTA DE ALGORITMOS

|   |   |    |
|---|---|----|
| 1 | Estrutura dos algoritmos evolucionários inspirados na computação quântica . . . | 47 |
| 2 | QEA . . . . .   | 49 |
| 3 | vQEA . . . . .  | 51 |
| 4 | NQGA . . . . .  | 52 |
| 5 | AEICQ - Algoritmo Evolucionário Inspirado na Computação Quântica . . . . .      | 63 |

# SUMÁRIO

|  |           |
|--|-----------|
| INTRODUÇÃO   | 1         |
| <b>1 PROJETO DE SISTEMAS DIGITAIS</b>                                  | <b>8</b>  |
| 1.1 <b>Introdução</b>  | 8         |
| 1.2 <b>Sistemas Seqüenciais</b>  | 9         |
| 1.2.1 <u>Descrição dos Sistemas Seqüenciais</u>                        | 10        |
| 1.2.2 <u>Sistemas Síncronos e Assíncronos</u>                          | 16        |
| 1.3 <b>Projeto de Sistemas Seqüenciais</b>                             | 16        |
| 1.4 <b>Resumo do Capítulo</b>  | 19        |
| <b>2 COMPUTAÇÃO EVOLUCIONÁRIA</b>                                      | <b>20</b> |
| 2.1 <b>Introdução</b>  | 20        |
| 2.2 <b>Algoritmos Genéticos</b>  | 23        |
| 2.2.1 <u>Codificação</u>   | 23        |
| 2.2.2 <u>Estratégias de Seleção</u>                                    | 25        |
| 2.2.2.1 Seleção pelo giro da roleta                                    | 25        |
| 2.2.2.2 Seleção por torneio  | 26        |
| 2.2.2.3 Elitismo   | 26        |
| 2.2.3 <u>Operadores de Reprodução</u>                                  | 27        |
| 2.2.3.1 Cruzamento   | 27        |
| 2.2.3.2 Mutação  | 28        |
| 2.3 <b>Estratégias Evolucionárias</b>                                  | 29        |
| 2.3.1 <u>Mutação nas Estratégias Evolucionárias</u>                    | 29        |
| 2.3.2 <u>Cruzamento ou Recombinação nas Estratégias Evolucionárias</u> | 30        |
| 2.3.3 <u>Tipos de Estratégias Evolucionárias</u>                       | 30        |
| 2.4 <b>Programação Evolucionária</b>                                   | 31        |
| 2.5 <b>Programação Genética</b>  | 32        |
| 2.5.1 <u>Codificação do Programa de Computador</u>                     | 33        |
| 2.5.2 <u>Reprodução de Programas de Computador</u>                     | 33        |
| 2.6 <b>Resumo do Capítulo</b>  | 36        |
| <b>3 COMPUTAÇÃO EVOLUCIONÁRIA INSPIRADA NA COMPUTAÇÃO QUÂNTICA</b>     | <b>37</b> |
| 3.1 <b>Introdução à Computação Quântica</b>                            | 37        |
| 3.1.1 <u>Conceitos da Computação Quântica</u>                          | 38        |
| 3.1.1.1 Bits quânticos   | 38        |
| 3.1.1.2 Sistemas quânticos   | 39        |
| 3.1.1.3 Portas quânticas   | 40        |
| 3.1.2 <u>Simuladores de Computadores Quânticos</u>                     | 42        |
| 3.2 <b>Algoritmos Evolucionários Inspirados na Computação Quântica</b> | 42        |
| 3.2.1 <u>Representação</u>   | 43        |
| 3.2.2 <u>Operadores de Variação</u>                                    | 44        |

|            |   |     |
|------------|---|-----|
| 3.2.3      | <u>Estrutura Básica</u> . . . . .   | 46  |
| 3.2.4      | <u>Exemplos de Aplicações</u> . . . . .   | 48  |
| <b>3.3</b> | <b>Resumo do Capítulo</b> . . . . .   | 53  |
| <b>4</b>   | <b>ATRIBUIÇÃO DE ESTADOS DE SISTEMAS SEQÜENCIAIS</b> . . . . .                  | 54  |
| <b>4.1</b> | <b>O Problema de Atribuição de Estados</b> . . . . .                            | 54  |
| 4.1.1      | <u>Impacto de Diferentes Atribuições de Estados</u> . . . . .                   | 56  |
| 4.1.2      | <u>Técnicas para Atribuição de Estados</u> . . . . .                            | 60  |
| <b>4.2</b> | <b>AEICQ Aplicado ao Problema de Atribuição de Estados</b> . . . . .            | 62  |
| 4.2.1      | <u>Estrutura do AEICQ</u> . . . . .   | 62  |
| 4.2.2      | <u>Codificação</u> . . . . .  | 63  |
| 4.2.3      | <u>Avaliação de Aptidão</u> . . . . .   | 64  |
| 4.2.4      | <u>Operador de Atualização</u> . . . . .  | 68  |
| <b>4.3</b> | <b>Resultados Experimentais</b> . . . . .                                       | 68  |
| 4.3.1      | <u>Comparação dos Resultados</u> . . . . .                                      | 69  |
| 4.3.2      | <u>Discussão dos Resultados</u> . . . . .                                       | 73  |
| <b>4.4</b> | <b>Resumo do Capítulo</b> . . . . .   | 79  |
| <b>5</b>   | <b>PROJETO EVOLUCIONÁRIO DE MÁQUINAS DE ESTADOS FINITOS</b> . . . . .           | 80  |
| <b>5.1</b> | <b>Eletrônica Evolucionária</b> . . . . .                                       | 80  |
| 5.1.1      | <u>Tipo de Projeto</u> . . . . .  | 81  |
| 5.1.2      | <u>Natureza do Projeto</u> . . . . .  | 81  |
| 5.1.3      | <u>Plataforma Evolucionária</u> . . . . .                                       | 82  |
| <b>5.2</b> | <b>Características da Eletrônica Evolucionária</b> . . . . .                    | 84  |
| 5.2.1      | <u>Espectro de Soluções</u> . . . . .   | 84  |
| 5.2.2      | <u>Conhecimento do Projeto</u> . . . . .  | 84  |
| 5.2.3      | <u>Restrições do Projeto</u> . . . . .  | 84  |
| 5.2.4      | <u>Escalabilidade</u> . . . . .   | 84  |
| 5.2.5      | <u>Avaliação das Soluções</u> . . . . .   | 85  |
| 5.2.6      | <u>Compreensão e Manutenção</u> . . . . .                                       | 85  |
| <b>5.3</b> | <b>Aplicações Envolvendo a Eletrônica Evolucionária</b> . . . . .               | 86  |
| <b>5.4</b> | <b>AEICQ Aplicado na Eletrônica Evolucionária</b> . . . . .                     | 87  |
| 5.4.1      | <u>Metodologia</u> . . . . .  | 87  |
| 5.4.2      | <u>Codificação</u> . . . . .  | 88  |
| 5.4.3      | <u>Avaliação da Aptidão</u> . . . . .   | 90  |
| <b>5.5</b> | <b>Resultados Experimentais</b> . . . . .                                       | 97  |
| 5.5.1      | <u>Comparação dos Resultados</u> . . . . .                                      | 98  |
| 5.5.2      | <u>Discussão dos Resultados</u> . . . . .                                       | 103 |
| <b>5.6</b> | <b>Resumo do Capítulo</b> . . . . .   | 104 |
| <b>6</b>   | <b>CONCLUSÕES E TRABALHOS FUTUROS</b> . . . . .                                 | 106 |
| <b>6.1</b> | <b>Conclusões</b> . . . . .   | 106 |
| <b>6.2</b> | <b>Trabalhos Futuros</b> . . . . .  | 108 |
|            | <b>REFERÊNCIAS</b> . . . . .  | 110 |
|            | <b>APÊNDICE A – Especificação das Máquinas de Estados Referência</b> . . . . .  | 119 |
|            | <b>APÊNDICE B – Lógicas Evoluídas para as Máquinas de Referências</b> . . . . . | 135 |

# INTRODUÇÃO

O PROJETO de circuitos eletrônicos ganhou um novo horizonte nos últimos anos com o surgimento de uma nova área de pesquisa denominada *eletrônica evolucionária* (ZEBULUM; PACHECO; VELLASCO, 2002). Esta área estuda a aplicação da computação evolucionária no domínio da eletrônica. A computação evolucionária é uma área de pesquisa que se inspira na natureza para a resolução de problemas de otimização e busca em diversas áreas do conhecimento através do desenvolvimento de novos modelos computacionais. Em geral, as técnicas agrupadas sob o termo computação evolucionária compartilham um conceito básico comum de simulação da evolução da estrutura de possíveis soluções através de transformações inspiradas nos processos naturais de seleção, mutação e reprodução.

Antes do surgimento da eletrônica evolucionária, o projeto de circuitos eletrônicos podia ser desenvolvido basicamente de duas maneiras: a primeira é a forma manual executada diretamente pelo projetista e a segunda é realizada com o auxílio das ferramentas do tipo ECAD (*Electronic Computer-Aided Design*). As ferramentas ECAD são programas de computador que utilizam heurísticas ou mesmo metaheurísticas para tratar de forma automatizada as etapas que envolvem o projeto dos circuitos eletrônicos. Estas ferramentas se tornaram importantes nos últimos anos devido às exigências cada vez maiores quanto à funcionalidade e ao desempenho dos circuitos eletrônicos. As exigências atualmente impostas ao projeto de circuitos colocam o auxílio das ferramentas ECAD praticamente como indispensável.

O principal desafio das pesquisas envolvendo a eletrônica evolucionária é a evolução de circuitos para aplicações industriais e também obter métodos para melhorar o desempenho dos próprios algoritmos evolucionários aplicados na síntese de circuitos eletrônicos. Os algoritmos evolucionários normalmente utilizados em aplicações envolvendo a eletrônica evolucionária são os algoritmos genéticos (ALI, 2003) e a programação genética (NEDJAH; MOURELLE, 2005a), os quais encontram-se agrupados sob o termo computação evolucionária. A ideia principal desta área do conhecimento é que cada circuito eletrônico possível pode ser representado como um indivíduo ou *cromossomo* de um processo evolucionário, que engloba operações genéticas sobre os circuitos. Esta área do conhecimento é muito vasta e, por isso, é possível encontrar

na literatura trabalhos com foco em diferentes problemas, tais como a otimização e síntese de circuitos digitais combinacionais e seqüenciais (MILLER; THOMSON, 1995) (ARAUJO; NEDJAH; MOURELLE, 2008b), síntese de circuitos analógicos (KOZA; KEANE; STREETER, 2005), síntese de amplificadores operacionais (KRUIKAMP; LEENAERTS, 1995), otimização do tamanho de transistores (ROGENMOSER; KAESLIN; BLICKLE, 1996), entre outros.

Dentre as vantagens da eletrônica evolucionária, pode-se destacar a capacidade de evoluir circuitos que dificilmente seriam obtidos em projetos desenvolvidos por seres humanos, mesmo quando aplicada por usuários que possuem apenas um conhecimento básico sobre o domínio do projeto. Em contrapartida, a propriedade da escalabilidade é descrita na literatura como um dos maiores desafios para a eletrônica evolucionária (YAO; HIGUCHI, 1999). Quanto mais complexo é um circuito, maior deve ser o tamanho do indivíduo que o codifica e quanto maior o indivíduo, maior fica o espaço de busca do algoritmo. Por consequência, quanto maior o espaço de busca, maior deve ser o tamanho da população utilizada para que uma área significativa do espaço de busca possa ser explorada.

Um caminho para aliviar o problema de escalabilidade pode ser a computação quântica. A computação quântica é uma área de pesquisa relativamente nova que explora as possibilidades de se aplicar as propriedades da mecânica quântica na computação. Os circuitos eletrônicos que implementam os bits de informação nos computadores atuais são objetos clássicos e, portanto, seguem as leis da física clássica. Como consequência, cada bit em um computador clássico só pode estar em um dos estados possíveis, 0 ou 1, que são, por sua vez, mutuamente exclusivos. Acontece que no mundo dos átomos, a mecânica quântica nos ensina que os bits, que no caso quântico são chamados de bits quânticos ou q-bits, podem simultaneamente estar nos estados 0 e 1. Esta propriedade é chamada de *superposição* de estados quânticos.

Para entender o seu alcance, pode-se considerar a seguinte analogia: supondo que uma pessoa jogue uma moeda para o alto, sabemos que ao cair no chão, o resultado será ou “cara” ou “coroa”, com probabilidade igual a 50% para cada lado. Se a moeda fosse um objeto quântico, o resultado poderia ser “cara”, “coroa”, ou qualquer superposição dos dois, como se a moeda pudesse cair com as duas faces para cima ao mesmo tempo. Atribuindo o estado lógico 0 para “cara” e 1 para “coroa”, pode-se então, com uma moeda quântica, superpor os estados lógicos que classicamente são exclusivos. Contudo, os estados quânticos superpostos não podem ser observados diretamente. Por exemplo, se uma moeda fosse colocada em uma superposição de “cara” e “coroa” e fosse feita uma tentativa de olhar para ela, não seria vista a superposição, mas apenas uma das possibilidades clássicas, ou seja, “cara” ou “coroa”. Em resumo, o ato

de tentar observar estados superpostos tem como consequência a destruição da superposição (OLIVEIRA *et al.*, 2003).

A propriedade de superposição de estados já foi demonstrada muitas vezes em laboratórios de física em várias partes do mundo e, hoje em dia, é uma verdade incontestável (OLIVEIRA *et al.*, 2003). Para a computação, esta propriedade representa um ganho inimaginável em termos de velocidade de processamento, pois todas as seqüências de bits possíveis em um computador poderiam ser manipuladas simultaneamente. Uma demonstração espetacular deste ganho de velocidade foi feita em 1994 por um cientista americano chamado Peter Shor. Ele desenvolveu um algoritmo quântico para fatorar números grandes, um problema difícil para computadores clássicos (SHOR, 1994). Peter Shor mostrou que através do uso da computação quântica seria possível a fatoração de um número grande em um tempo polinomial. A dificuldade na fatoração de números grandes é a base da segurança da informação criptografada que trafega todos os dias pela *Internet* levando dados secretos. O algoritmo de Shor mostra que no dia em que um computador quântico estiver disponível, nenhuma mensagem criptografada pelos sistemas existentes será secreta. Este dia parece não estar muito longe, pois já foram implementados sistemas quânticos capazes de processar alguns bits quânticos e acredita-se que até o ano de 2020 será possível a fabricação de computadores quânticos práticos (BALL, 2006).

Enquanto os computadores quânticos não se encontram disponíveis, pode-se fazer uso dos princípios da computação quântica para tentar melhorar o desempenho dos algoritmos evolucionários. Em (NARAYANAN; MOORE, 1996), foi proposta pela primeira vez a utilização dos princípios da computação quântica em conjunto com a teoria que envolve a computação evolucionária, dando origem a uma nova classe de algoritmos chamados de *algoritmos evolucionários inspirados na computação quântica*. Nesta classe de algoritmos, a representação dos indivíduos, que codificam as possíveis soluções de um determinado problema, é feita de forma probabilística usando o conceito de  $q$ -bits e formando indivíduos quânticos ( $q$ -indivíduo) que são compostos de uma série de  $q$ -bits. Este tipo de representação em conjunto com os operadores de atualização proporcionam aos algoritmos evolucionários inspirados na computação quântica uma grande capacidade de busca e rápida convergência, devido ao uso de populações de tamanhos reduzidos (ZHANG *et al.*, 2006).

Neste trabalho, foi desenvolvido um Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ) cuja estrutura básica é similar a apresentada por outros algoritmos evolucionários inspirados na computação quântica que encontram-se disponíveis na literatura, como por exemplo, (HAN; KIM, 2002), (ZHANG *et al.*, 2003), (PLATEL; SCHLIEBS; KASABOV, 2007). O

AEICQ foi aplicado no projeto evolucionário de circuitos seqüenciais síncronos.

Um circuito seqüencial é um sistema digital no qual a saída é determinada tanto pela entrada atual quanto pelo resultado de um evento anterior (UYEMURA, 2002). Em geral, um circuito seqüencial é composto de uma via de dados e de um controlador. A via de dados é formada no mínimo por um barramento e registradores. O controlador é uma máquina de estados finitos composta de um circuito lógico combinacional, o qual é responsável pelo comportamento do circuito seqüencial. Nos circuitos seqüenciais síncronos, as transições de um estado da máquina para outro ocorre em instantes discretos no tempo que são definidos pelos momentos de transição de um sinal astável, chamado de sinal de *relógio*.

Os circuitos seqüenciais são responsáveis por grande parte do *potencial* associado às técnicas digitais. Como as máquinas de estados podem ser projetadas para reagir a qualquer entrada, fornecendo a resposta apropriada, estas podem ser implementadas em situações de isolamento (*stand-alone*), nas quais não necessitam da ajuda de operadores humanos. As máquinas de estado são, realmente, a base da automação moderna (UYEMURA, 2002). A lista de aplicações de circuitos lógicos seqüenciais é limitada apenas pela imaginação do projetista. Devido à complexidade, normalmente o projeto dos sistemas seqüenciais é dividido em cinco etapas principais: (i) Especificação da máquina de estados; (ii) Redução de estados; (iii) Atribuição de estados; (iv) Síntese da lógica de controle e (v) Implementação da máquina de estados.

Na etapa de especificação, é definido o comportamento requerido para a máquina de estados através de diagramas e tabelas de transição de estados. Na etapa de redução de estados são identificados os estados que produzem o mesmo sinal de saída e apresentam o mesmo comportamento de próximo estado. Estes estados são ditos *equivalentes* e, portanto, são substituídos por um único estado. Estas duas etapas, assim como a etapa de implementação da máquina, na qual o circuito projetado é montado através da interconexão dos componentes físicos, não são o foco desta dissertação.

Na etapa da atribuição de estados, cada um dos estados, que normalmente são representados de forma simbólica nos primeiros estágios do projeto, recebe um código binário. A atribuição de estados escolhida para uma determinada máquina de estados pode afetar a complexidade da sua lógica de controle (ERCEGOVAC; LANG; MORENO, 1999). Portanto, uma *boa* atribuição de estados pode reduzir a quantidade de lógica combinacional requerida para a implementação da operação seqüencial do sistema. O problema de atribuição de estados é *NP-difícil* (PAPADIMITRIOU, 1994) e já vem sendo estudado desde o início da década de

60 (ARMSTRONG, 1962) e desde então vários trabalhos sobre o assunto foram desenvolvidos e publicados, como por exemplo (AMARAL; TUMER; GLOSH, 1995), (ALI, 2003) e (ARAUJO; NEDJAH; MOURELLE, 2008a). Uma máquina de estados com apenas nove estados possui aproximadamente 4 bilhões de diferentes possíveis atribuições. Portanto, é praticamente impossível testar todas as atribuições com o objetivo de determinar qual delas proporcionaria o circuito lógico mais simples. Por isso, ao longo dos anos foram desenvolvidas algumas técnicas para a busca de atribuições de estados otimizadas. As mais importantes são a *one-hot* (ERCEGOVAC; LANG; MORENO, 1999) e as técnicas baseadas em heurísticas, como por exemplo, KISS™ (MICHELI; BRAYTON; SANGIOVANNI-VINCENTELLI, 1984) e NOVA™ (VILLA; SANGIOVANNI-VINCENTELLI, 1990). Recentemente, os algoritmos evolucionários também passaram a ser utilizados na resolução do problema de atribuição de estados, como por exemplo, em (ALI, 2003) e (AMARAL; TUMER; GLOSH, 1995).

Na primeira parte desta dissertação, propõe-se a aplicação do AEICQ na solução do problema da atribuição de estados. O AEICQ utiliza as heurísticas de (ARMSTRONG, 1962) e (HUMPHREY, 1958) para avaliação das atribuições de estados geradas durante o processo evolucionário. Os resultados obtidos pelo AEICQ são comparados com os obtidos pelos algoritmos genéticos (NEDJAH; MOURELLE, 2005a), (AMARAL; TUMER; GLOSH, 1995) e (ALI, 2003) e pelo método não evolucionário NOVA™ (VILLA; SANGIOVANNI-VINCENTELLI, 1990).

Na segunda parte desta pesquisa, o AEICQ foi aplicado na síntese evolucionária da lógica de controle das máquinas de estados finitos. Após a etapa de atribuição de estados estar concluída, a tabela de transição de estados da máquina é utilizada como um dado de entrada para o AEICQ. Nesta aplicação, os circuitos são codificados pelo AEICQ na forma de uma matriz de células, onde cada célula codifica uma porta lógica e os seus respectivos sinais de entrada.

Segundo o tipo de plataforma evolucionária, as aplicações da eletrônica evolucionária podem ser classificadas em extrínsecas ou intrínsecas. Nas aplicações intrínsecas, cada uma das soluções geradas pelo algoritmo é implementada em circuitos semi-personalizados que podem ser reconfigurados. Já nas aplicações extrínsecas, os circuitos evoluídos são simulados e avaliados através de programas de computador. O AEICQ utiliza a evolução extrínseca e em nível de portas lógicas, ou seja, a unidade básica do circuito digital que é manipulada pelo algoritmo evolucionário é uma porta lógica. Cada circuito evoluído pelo AEICQ é avaliado segundo os seguintes critérios: (i) Deve ser totalmente *funcional*, ou seja, garantir o comportamento da entrada/saída; (ii) Deve ser otimizado segundo *área* ocupada e o *atraso* de propagação. Os

circuitos evoluídos pelo AEICQ são comparados com os circuitos obtidos em outros trabalhos que utilizam a programação genética (NEDJAH; MOURELLE, 2005a), algoritmos genéticos (ALI, 2003) e pelo método não evolucionário ABC (ABC, 2005).

Os resultados desta dissertação contribuem para a área da engenharia eletrônica através da apresentação de uma nova ferramenta para tratar o problema de atribuição de estados nas máquinas de estados finitos e para a síntese da lógica de controle destas máquinas. Os resultados obtidos são competitivos em termos de área e desempenho e, em alguns casos, superiores aos de técnicas usuais de projeto, dos algoritmos genéticos e da programação genética. Na área da computação evolucionária foram concebidas algumas alterações com relação aos algoritmos evolucionários inspirados na computação quântica que encontram-se disponíveis na literatura. O AEICQ utiliza uma limitação nas amplitudes de probabilidade dos q-bits, além de implementar a operação de migração global de uma maneira diferente da que é proposta no algoritmo apresentado em (HAN; KIM, 2002). O objetivo destas alterações é aprimorar o desempenho do AEICQ em problemas envolvendo a eletrônica evolucionária, ajudando o algoritmo a escapar de soluções locais. Porém, estas alterações possuem caráter genérico, podendo ser utilizadas em outras aplicações.

Esta dissertação possui mais seis capítulos e dois apêndices, cujos conteúdos são descritos a seguir.

O Capítulo 1 apresenta uma breve discussão das dificuldades e os desafios que envolvem o projeto de circuitos digitais, em especial dos sistemas seqüenciais. Os sistemas seqüenciais são formalmente definidos e, em seguida, são descritas as etapas que envolvem o projeto destes sistemas. São discutidos também os motivos que levaram ao surgimento de uma nova área de pesquisa que estuda a utilização de ferramentas automáticas no projeto dos circuitos eletrônicos.

O Capítulo 2 descreve as principais características dos algoritmos evolucionários que são agrupados sob a computação evolucionária. Os principais componentes dos algoritmos evolucionários, isto é, *representação*, *avaliação* e *operadores genéticos*, são descritos em detalhes. Algumas vantagens da computação evolucionária frente aos algoritmos de busca tradicional e as técnicas de otimização são demonstradas. Em seguida, é apresentada uma breve descrição dos algoritmos genéticos (HOLLAND, 1975), estratégias evolucionárias (RECHENBERG, 1973) (SCHWEFEL, 1977), programação evolucionária (FOGEL; OWENS; WALSH, 1966) e programação genética (KOZA, 1992).

O Capítulo 3 apresenta uma introdução aos princípios da computação quântica e des-

creve uma nova classe de algoritmos que surgiu recentemente, chamada de algoritmos evolucionários inspirados na computação quântica. As principais características relativas à representação das soluções e aos operadores de atualização por estes algoritmos são discutidas. Os principais algoritmos encontrados na literatura e algumas aplicações relevantes também são descritos.

O Capítulo 4 detalha o impacto da etapa de atribuição de estados na complexidade da lógica de controle das máquinas de estados finitos. Também são descritas algumas das técnicas aplicadas no problema de atribuição de estados. Neste capítulo se propõe a aplicação do Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ) na solução do problema de atribuição de estados de máquinas de estados finitos.

O Capítulo 5 introduz a área da eletrônica evolucionária e apresenta a classificação da mesma. Aqui são descritas as principais características dos projetos de circuitos que são desenvolvidos utilizando as técnicas evolucionárias. Alguns trabalhos relevantes divulgados nesta área até o presente momento são comentados. O AEICQ é aplicado como uma ferramenta automática para a síntese da lógica de controle das máquinas de estados finitos.

O Capítulo 6 apresenta uma síntese do trabalho desenvolvido e dos resultados obtidos, quando algumas conclusões são, então, tecidas. Além disso, uma discussão sobre a continuidade do trabalho de pesquisa também é elaborada, apontando para algumas direções pertinentes.

Essa dissertação possui dois apêndices. O Apêndice A apresenta as tabelas de transição de estados das máquinas de estados finitos utilizadas nos experimentos apresentados no Capítulo 4 e no Capítulo 5. Estas máquinas de estados finitos são utilizadas como referência ou *benchmarks* para o desenvolvimento de sistemas seqüenciais e encontram-se disponíveis em (ACM/SIGDA, 1989). No Apêndice B, são apresentadas as equações evoluídas para a excitação dos *flip-flops* que formam o registrador de estado da máquina, assim como as equações de controle dos sinais da saída primária da mesma.

# Capítulo 1

## PROJETO DE SISTEMAS DIGITAIS

**O**S requisitos de funcionalidade e desempenho que estão sendo impostos ultimamente aos sistemas digitais estão exigindo projetos de circuitos cada vez mais eficientes. Os projetistas ou as ferramentas ECAD (*Electronic Computer-Aided Design*) devem ser capazes de gerar projetos de circuitos que englobem cada vez mais funcionalidades em áreas ainda menores. Além disso, estes circuitos necessariamente devem apresentar desempenho superior. Neste capítulo será apresentada uma introdução do processo que envolve o projeto dos sistemas seqüenciais síncronos com o objetivo de permitir o entendimento do trabalho apresentado nesta dissertação.

### 1.1 Introdução

O projeto de circuitos digitais eficientes, realizado de forma manual, se tornou uma tarefa demorada devido ao aumento das exigências quanto à funcionalidade e ao desempenho dos circuitos digitais. Por isso, a importância das ferramentas de síntese automática de circuitos digitais vem crescendo rapidamente nos últimos anos (KHAN, 2005). Com o uso destas ferramentas, os projetistas podem trabalhar em níveis mais altos de abstração, deixando a ferramenta executar a tarefa de transição para os níveis mais baixos de abstração, assim como as tarefas de otimização.

O conceito de síntese de lógica de circuitos pode ser definido como um processo de transformação de uma descrição da funcionalidade requerida em um circuito. Tradicionalmente, este processo é dividido em síntese de circuitos combinacionais e seqüenciais.

Em geral, os processos automatizados envolvendo o projeto de circuitos lógicos buscam soluções otimizadas com relação a um ou mais requisitos do circuito. Os requisitos mais comuns nestes processos são: a *área* ocupada pelo circuito, o *atraso* de propagação imposto e o *consumo* de energia necessário para seu bom funcionamento.

A minimização de área e as restrições de atraso têm um efeito imediato no projeto de circuitos integrados. O projeto deve se limitar aos recursos disponíveis no circuito integrado. A minimização de área tem um impacto econômico importantíssimo no custo de implementação, pois este diminui exponencialmente em função do tamanho do circuito integrado (ALI, 2003). Um outro parâmetro que tem aumentado de importância ultimamente, especialmente para tecnologias móveis tais como computadores portáteis e telefones celulares, é a minimização da energia consumida pelo circuito.

O projeto de circuitos otimizados que atendam simultaneamente as restrições de área, atraso e consumo de energia é um problema de difícil solução devido ao grande número de soluções potenciais mesmo para um pequeno conjunto de equações lógicas. Neste caso, as ferramentas automatizadas utilizadas no projeto de circuitos desempenham um papel fundamental. As ferramentas do tipo ECAD (*Electronic Computer-Aided Design*) são programas de computador utilizados para o projeto automatizado de sistemas eletrônicos.

## 1.2 Sistemas Seqüenciais

Os sistemas digitais dividem-se em duas classes: *sistemas combinacionais* e *sistemas seqüenciais*. Para ser classificado como combinacional, um sistema deve atender às seguintes restrições (RHYNE, 1973):

1. Os valores 0/1 dos sinais de saída devem depender apenas dos valores atuais 0/1 dos sinais de entrada.
2. Não deve existir realimentação dos sinais de saída para os sinais de entrada.

Estas duas restrições fazem do projeto e análise de sistemas combinacionais uma tarefa direta. Cada sinal de saída pode ser expresso como uma função Booleana dos sinais de entrada:

$$s_i = \phi_i(e_1, e_2, \dots, e_n), i = 1, 2, \dots, m \quad (1)$$

onde  $e_1, e_2, \dots, e_n$  são os sinais de entrada e os  $\phi_i$ s são as funções Booleanas que geram os sinais de saída  $s_i$ .

Em muitos sistemas lógicos, as saídas não podem ser determinadas apenas conhecendo-se os valores atuais das entradas. Para tal, a história passada dos sinais de entrada e saída deve ser incluída na determinação destes sinais de saída. Os sistemas seqüenciais são fundamentalmente diferentes dos sistemas combinacionais, apesar destes poderem ser em parte

combinacionais. O termo *seqüencial* é comumente usado para descrever esta distinção. Os sistemas seqüenciais possuem duas características principais:

1. Existe pelo menos um caminho de realimentação entre a saída e a entrada do sistema.
2. O sistema tem a capacidade de memorizar informações passadas, de tal forma que os valores anteriores de entrada e de saída possam ser utilizados na determinação dos sinais de saída atuais e próximos.

A eliminação das restrições combinacionais possibilita uma aplicação maior dos sistemas digitais. A utilização de memória e realimentação permite que o tempo seja incluído como um parâmetro na determinação do comportamento do sistema. Assim, informações relacionadas a eventos passados podem ser usadas para determinar os sinais atuais de saída, e informações de ambos, do passado e do presente, podem ser usadas para especificar atividades futuras.

Uma vantagem que pode ser observada através da comparação entre sistemas seqüenciais e sistemas puramente combinacionais é a economia de *hardware* que é obtida através do uso seqüencial (repetitivo) do mesmo circuito lógico. A desvantagem neste caso reside no fato do sistema seqüencial demandar um tempo maior que um sistema combinacional para execução de uma mesma tarefa (RHYNE, 1973).

### 1.2.1 Descrição dos Sistemas Seqüenciais

Antes da abordagem de algumas questões relacionadas ao projeto de sistemas seqüenciais, uma descrição mais detalhada destes sistemas se faz necessária. A Figura 1 mostra a arquitetura genérica de um sistema seqüencial.

Os sinais de entrada podem ser divididos em dois conjuntos: *sinais da entrada primária* ( $e_1, e_2, \dots, e_n$ ) e *sinais da entrada secundária* ( $x_1, x_2, \dots, x_k$ ). Os valores dos sinais da entrada primária definem a entrada atual do sistema, que pode ser uma das  $2^n$  diferentes combinações possíveis. Os valores dos sinais da entrada secundária refletem a história passada da operação do sistema seqüencial. Estes sinais também são conhecidos como *sinais de estado atual* e cujos valores são lidos da memória do sistema seqüencial.

A capacidade de memorização do sistema pode ser provida através da utilização de registradores de estado na forma de *flip-flops* ou *latches* (ERCEGOVAC; LANG; MORENO, 1999). Os  $k$  valores dos sinais da entrada secundária formam o *estado atual* do sistema. Portanto, o sistema pode apresentar no máximo  $2^k$  estados diferentes. Por este motivo, os sistemas seqüenciais são comumente chamados de *sistemas de estados finitos* (ERCEGOVAC; LANG; MORENO,

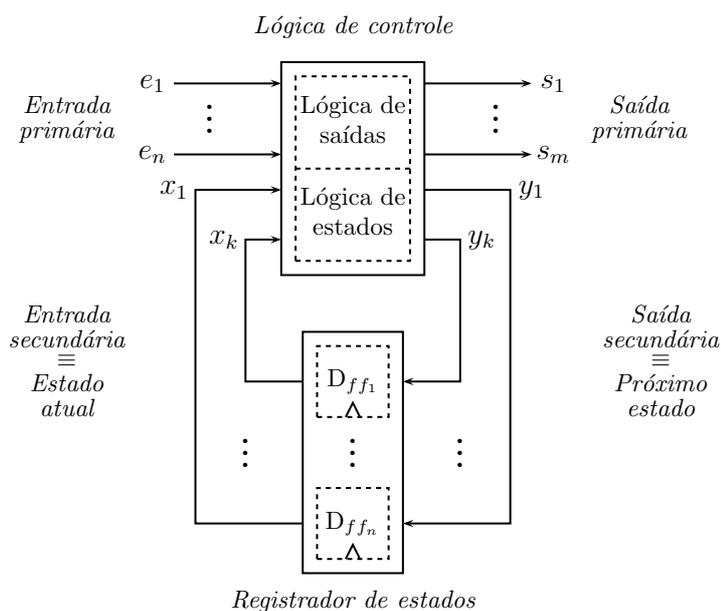


Figura 1: Arquitetura genérica de um sistema seqüencial

1999). O *estado atual total* do sistema é definido pelos dois conjuntos de entradas, sendo possível  $2^{n+k}$  estados atuais totais diferentes.

Os sinais de saída também podem ser divididos em dois conjuntos: *sinais de saída primárias* ( $s_1, s_2, \dots, s_m$ ) e *sinais da saída secundária* ( $y_1, y_2, \dots, y_k$ ). Os sinais da saída primária são os sinais que são enviados pelo sistema seqüencial para o ambiente no qual este se encontra inserido. Estes sinais podem, por exemplo, operar ou transmitir informações para outros sistemas, gerar sinais de indicação e alarme, entre outros. Os sinais da saída secundária formam a entrada da memória do sistema seqüencial. Estes sinais descrevem o novo valor que será armazenado na memória assim que começar o próximo ciclo de operação do sistema. Portanto, os sinais da saída secundária são descritos como o *próximo estado* do sistema seqüencial. No momento em que os novos valores dos sinais  $y_i$  são escritos na memória, através da substituição do valor dos sinais  $x_i$ , o sistema seqüencial passa então do estado atual para o próximo estado. Os valores dos sinais da saída primária e secundária são obtidos através de operações combinacionais entre os valores dos sinais da entrada primária e da entrada secundária. Portanto, conhecendo-se o estado atual do sistema definido por  $x_1, x_2, \dots, x_k$  e os valores dos sinais da entrada primária (uma das  $2^n$  combinações possíveis), a parte combinacional do sistema seqüencial gera os sinais da saída primária apropriados, e por meio da geração dos valores dos sinais da saída secundária ( $y_1, y_2, \dots, y_k$ ), seleciona o próximo estado para o qual o sistema mudará. Para cada estado atual podem existir até  $2^n$  combinações de próximos estados

e das saídas, correspondendo às  $2^n$  combinações diferentes que os  $n$  sinais da entrada primária podem assumir.

A relação entre de um lado os sinais de estado atual e os da entrada primária, e do outro lado os sinais de próximo estado e os da saída primária, relação esta que descreve o comportamento de um sistema seqüencial, pode ser representada de diversas maneiras. As representações mais utilizadas são: o *diagrama de transição de estados* e a *tabela de transição de estados*.

O diagrama de transição de estados representa um sistema seqüencial diretamente na forma de grafo com os seus nós representando os estados e os arcos representando as transições entre os estados. Cada estado possível é representado por um círculo com a sua designação dentro. As condições da entrada primária que correspondem a cada transição são representadas acima de cada arco, juntamente com os valores da saída primária. Esta representação é válida para máquinas cuja saída primária depende do estado atual e da entrada primária. Neste caso, a máquina de estados é dita *Mealy*. Caso a saída primária possa ser determinada baseando-se somente nos valores dos sinais do estado atual, a máquina de estado é dita *Moore*. Neste caso, a saída é indicada junto com a designação do estado, dentro do círculo. A Figura 2 mostra um exemplo de máquina de Mealy representada através de seu diagrama de transição de estados.

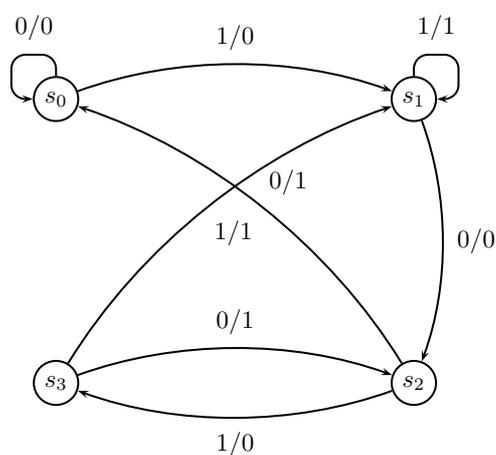


Figura 2: Exemplo de um diagrama de transição de estados

A tabela de transição de estados também é muito utilizada para a representação de uma máquina de estados. A Tabela 1 mostra o mesmo sistema da Figura 2 representado de forma tabular. Cada linha da tabela corresponde a um arco no diagrama de transição de estados. Por convenção, as colunas mais à esquerda da tabela correspondem aos sinais da entrada primária e as colunas mais à direita correspondentes aos sinais da saída primária. A coluna após a entrada

primária é a coluna que representa o estado atual do sistema e a seguinte é a que descreve o comportamento do próximo estado.

Tabela 1: Exemplo de uma tabela de transição de estados

| Entrada | Estado Atual | Próximo Estado | Saída |
|---------|--------------|----------------|-------|
| 0       | $s_0$        | $s_0$          | 0     |
| 0       | $s_1$        | $s_2$          | 0     |
| 0       | $s_2$        | $s_0$          | 1     |
| 0       | $s_3$        | $s_2$          | 1     |
| 1       | $s_0$        | $s_1$          | 0     |
| 1       | $s_1$        | $s_1$          | 1     |
| 1       | $s_2$        | $s_3$          | 0     |
| 1       | $s_3$        | $s_1$          | 1     |

A tabela e o diagrama de transição de estados descrevem claramente o comportamento de um sistema seqüencial digital, por isso são ferramentas essenciais na análise e projeto destes sistemas.

O sistema seqüencial simples representado na Tabela 1 possui um sinal de entrada primária e outro de saída primária. Como o sistema possui quatro estados ( $s_0 \dots s_3$ ), são necessários dois sinais de estado para a codificação dos mesmos. Quando combinados com os dois valores possíveis de entrada, isto representa oito estados *totais* possíveis. Os valores dos sinais de próximo estado e o sinal da saída primária devem ser especificados para cada estado total atual. Supondo que os estados do sistema sejam codificados como  $s_0 \equiv 00$ ,  $s_1 \equiv 10$ ,  $s_2 \equiv 01$ ,  $s_3 \equiv 11$ , então a tabela de transição de estados seria preenchida conforme indicado na Tabela 2. Como existem apenas quatro estados, dois *flip-flops*,  $W$  e  $X$  são suficientes para o armazenamento do estado do sistema.

Tabela 2: Tabela de transição de estados considerando a codificação binária

| Estado Presente Total |     |     | Próximo Estado |       | Saída | Comportamento |          |
|-----------------------|-----|-----|----------------|-------|-------|---------------|----------|
| $I$                   | $W$ | $X$ | $W^+$          | $X^+$ | $O$   | $W$           | $X$      |
| 0                     | 0   | 0   | 0              | 0     | 0     | 0             | 0        |
| 0                     | 0   | 1   | 0              | 0     | 1     | 0             | $\beta$  |
| 0                     | 1   | 0   | 0              | 1     | 0     | $\beta$       | $\alpha$ |
| 0                     | 1   | 1   | 0              | 1     | 1     | $\beta$       | 1        |
| 1                     | 0   | 0   | 1              | 0     | 0     | $\alpha$      | 0        |
| 1                     | 0   | 1   | 1              | 1     | 0     | $\alpha$      | 1        |
| 1                     | 1   | 0   | 1              | 0     | 1     | 1             | 0        |
| 1                     | 1   | 1   | 1              | 0     | 1     | 1             | $\beta$  |

A coluna *comportamento* na Tabela 2 resulta da comparação dos estados atuais e próximos estados dos *flip-flops*  $W$  e  $X$ . O comportamento de um *flip-flop* ao perceber uma transição

do sinal astável de *relógio* pode ser representado simbolicamente como uma das quatro seguintes situações:

1. Se o *flip-flop* está inicialmente no estado 0 (RESET) e continua neste estado após ter percebido a transição do sinal de relógio, o seu comportamento é *estático* e pode ser representado pelo símbolo 0, indicando que o *flip-flop* está em RESET tanto no estado atual quanto no próximo estado.
2. Se o *flip-flop* está no estado 0 e muda para o estado 1 (SET), o seu comportamento é *dinâmico* e pode ser representado pelo símbolo  $\alpha$ .
3. Se o *flip-flop* está no estado 1 e muda para o estado 0, o seu comportamento é também *dinâmico* e pode ser representado pelo símbolo  $\beta$ .
4. Se o *flip-flop* está inicialmente no estado 1 e continua neste estado após ter percebido a transição do sinal de relógio, o seu comportamento é também *estático* e o símbolo 1 pode ser usado.

O mapa de *Karnaugh* pode então ser utilizado para a obtenção da equação de controle do sinal de saída da máquina de estados. O mapa de *Karnaugh* é um método gráfico para a representação de funções Booleanas. Este método é similar a uma tabela verdade, onde um mapa fornece o valor 0, 1 ou  $X$  de uma função Booleana para todas as possíveis combinações dos seus argumentos lógicos. Os mapas de *Karnaugh* podem ser usados para representar funções com um número qualquer de variáveis, porém são mais usados para funções com seis ou menos argumentos (RHYNE, 1973).

O mapa de *Karnaugh* para a função de saída,  $O$ , é mostrada na Figura 3, resultando na seguinte equação de controle:

$$O = W \cdot I + \bar{I} \cdot X. \quad (2)$$

|   |   |   |   |   |
|---|---|---|---|---|
|   |   | W |   |   |
|   |   | X |   |   |
|   | 0 | 1 | 1 | 0 |
| I | 0 | 0 | 1 | 1 |

Figura 3: Mapa de *Karnaugh* da função de saída

Os mapas de transição podem ser utilizados para se obter as equações de controle dos *flip-flops*. Estes mapas estendem a técnica do mapa de *Karnaugh* para cobrir o comportamento dos *flip-flops* ao perceber uma transição do sinal de relógio. Os mapas de transição dos *flip-flops* são mostrados na Figura 4. Utilizando *flip-flops* tipo D, a lógica de controle se torna:

$$\begin{aligned} D_w &= I \\ D_X &= \bar{I} \cdot W + X \cdot I \cdot \bar{W}. \end{aligned} \tag{3}$$

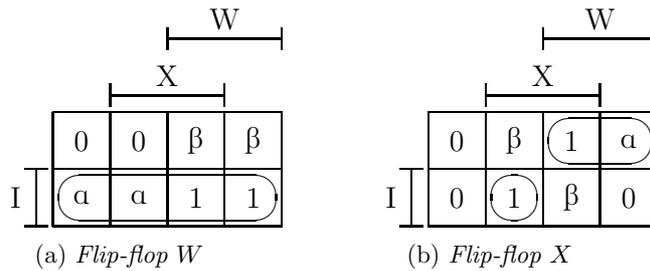


Figura 4: Mapas de transição dos *flip-flops* *W* e *X*

O esquemático do circuito que implementa a máquina de estados definida inicialmente pela tabela de transição de estados (Tabela 1) é mostrados na Figura 5.

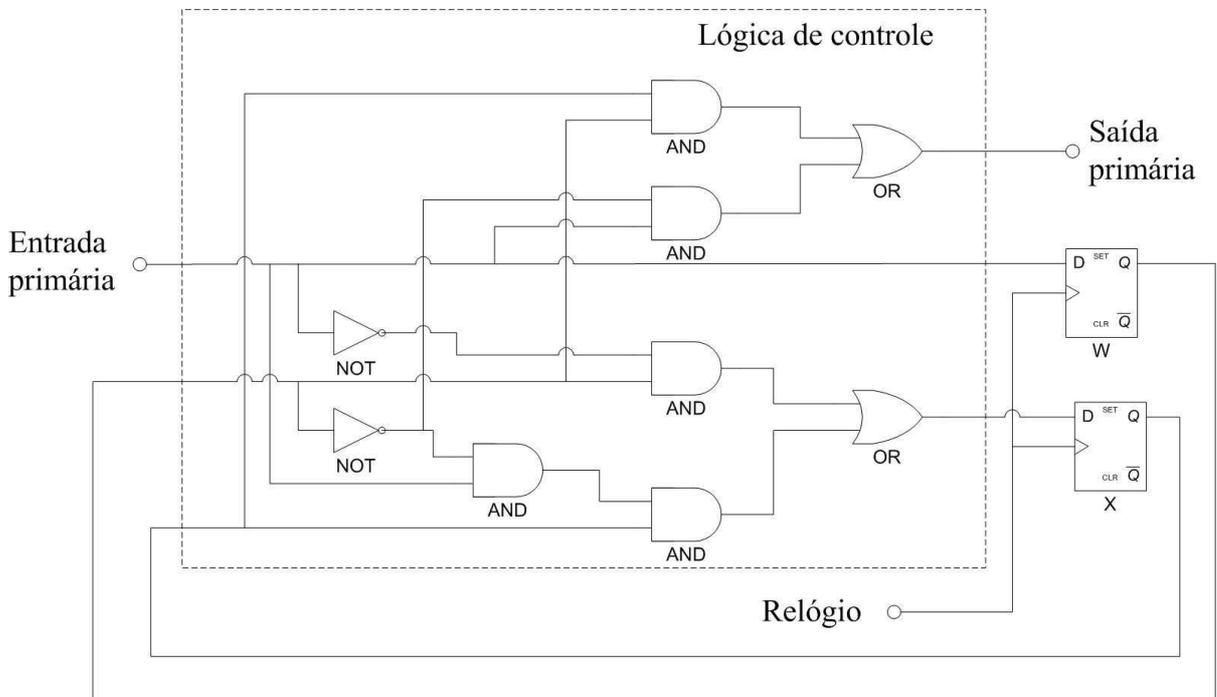


Figura 5: Exemplo de um circuito esquemático de um sistema seqüencial

### 1.2.2 Sistemas Síncronos e Assíncronos

De acordo com os instantes no tempo em que as entradas são consideradas, os sistemas seqüenciais são classificados em sistemas *síncronos* e *assíncronos* (ERCEGOVAC; LANG; MORENO, 1999). Nos sistemas síncronos, as entradas e saídas são consideradas em instantes discretos no tempo que são definidos pelas transições positivas ( $\Uparrow$ ) ou negativas ( $\Downarrow$ ) do sinal de relógio. Já nos sistemas assíncronos a variável tempo é contínua e os sinais de entrada e saída podem ser considerados em qualquer instante do tempo. Os sistemas assíncronos são mais difíceis de descrever, analisar e projetar do que os sistemas síncronos (ERCEGOVAC; LANG; MORENO, 1999). Uma vez que a maioria das aplicações usadas são síncronas, este trabalho se dedica a estes casos.

## 1.3 Projeto de Sistemas Seqüenciais

Um progresso considerável foi alcançado no entendimento da otimização de lógicas combinacionais num passado recente e conseqüentemente uma grande quantidade de ferramentas ECAD desenvolvidas industrialmente e em universidades encontram-se disponíveis para a síntese otimizada de circuitos combinacionais, como por exemplo, (GAJSKI; KUHN, 1983), (MAZUMDER; RUDNICK, 1998) e (ABC, 2005). Estes programas produzem resultados competitivos com relação aos circuitos lógicos projetados manualmente. Por outro lado, o projeto automático de circuitos seqüenciais otimizados ainda é considerado um assunto menos consolidado.

De uma forma genérica, a implementação de um sistema seqüencial digital consiste de uma *via de dados* e um *controlador*. Em geral, a via de dados inclui, entre outros, registradores, contadores, somadores, multiplexadores e barramentos. O controlador é uma máquina de estados finitos que implementa o controle requerido.

A síntese de sistemas seqüenciais é o processo de criação de uma implementação adequada de uma dada máquina de estados finitos. Devido à complexidade, a metodologia tradicional utilizada no projeto das máquinas de estados finitos é dividida em etapas. O processo passa por cinco etapas principais conforme ilustrado na Figura 6.

- *Especificação da máquina de estados*: O processo se inicia com a especificação do comportamento da máquina de estados de alguma forma, como por exemplo, através de uma tabela ou diagrama de transição de estados. Os estados internos da máquina de estados são normalmente identificados por símbolos, de tal forma que a especificação esteja o máximo possível focada no comportamento desejado e não em uma implementação específica.

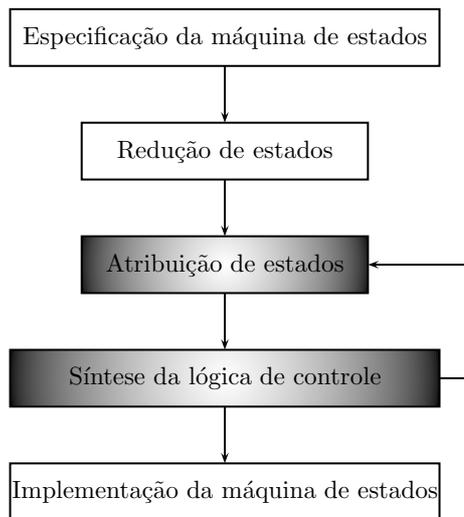


Figura 6: Metodologia para o projeto de máquinas de estados finitos

- *Redução de estados*: Os estados que produzem o mesmo sinal de saída e apresentam o mesmo comportamento de próximo estado, chamados de *estados equivalentes*, são identificados e combinados em um único estado que atue em substituição a todos os estados distintos anteriores. A quantidade mínima de sinais de estado de uma máquina é definida conforme mostrado na Equação 4,

$$K = \lceil \log_2(N) \rceil, \quad (4)$$

onde  $N$  representa a quantidade de estados necessários para a operação do sistema sequencial. Portanto, reduzindo-se o número de estados da máquina é possível diminuir o número de sinais de estado, e por consequência, reduzir a memória requerida para a implementação do sistema. A descrição de algumas técnicas tradicionais para a redução de estados podem ser encontradas em (BOOTH, 1967).

- *Atribuição de estados*: Durante os estágios iniciais do projeto das máquinas de estados finitos, normalmente são utilizadas especificações gerais dos sistemas. Nestas especificações, os diferentes estados são representados por símbolos distintos conforme indicado na Tabela 1. Porém, antes da implementação do sistema, é necessário atribuir para cada estado da máquina uma combinação distinta dos sinais de estado conforme mostrado na Tabela 2. O *problema de atribuição de estados* consiste em atribuir a cada estado da máquina um código binário distinto com o objetivo de minimizar alguma função objetivo relacionada com a realização do circuito. A relação entre os estados e uma combinação específica dos sinais de estado pode ser definida como desejado. Entretanto, a complexidade

da lógica de controle da máquina de estados é diretamente afetada por esta escolha, uma vez que a lógica combinacional deve interpretar a informação de estado atual para gerar os sinais de próximo estado. Uma *boa* atribuição de estados é aquela que reduz a quantidade de lógica combinacional requerida para a implementação da operação seqüencial (RHYNE, 1973). A atribuição de estados pode ser realizada de forma tradicional, como por exemplo, utilizando-se a técnica *One-hot* (ERCEGOVAC; LANG; MORENO, 1999).

- *Síntese da lógica de controle*: A lógica combinacional que gera os sinais de saída primários e secundários é obtida durante esta etapa. Os sinais que definem o estado presente total são usados como entradas para esta lógica. A síntese tradicional da lógica de controle é realizada utilizando-se os mapas de *Karnaugh* e de transições. Caso algum dos parâmetros do circuito como, por exemplo, área consumida ou atraso de propagação, não atendam às restrições do projeto, uma nova atribuição de estados pode ser testada.
- *Implementação da máquina de estados*: O projeto elaborado nas etapas anteriores é então implementado através da interconexão de componentes digitais físicos e da conexão destes com o ambiente no qual o sistema está inserido.

Atualmente, encontram-se disponíveis algumas ferramentas baseadas em heurísticas do tipo ECAD para a síntese de sistemas seqüenciais. Destas, algumas são produtos comerciais e outras são ferramentas desenvolvidas em universidades e que são de domínio público. A ferramenta ABC (ABC, 2005), por exemplo, foi desenvolvida por um grupo de pesquisadores da Universidade da Califórnia. O ABC incorpora diferentes algoritmos para cada uma das etapas que envolvem o projeto de sistemas seqüenciais, tais como, a redução de estados e atribuição de estados.

Recentemente, os algoritmos evolucionários surgiram como um conceito geral para o desenvolvimento de novos modelos computacionais para projeto e otimização. Os algoritmos evolucionários possuem uma base conceitual que é a simulação de dois processos fundamentais. Estes processos são a variação aleatória e a seleção dentro de uma população, descrito por Charles Darwin (1859) como o princípio da evolução. A variação e seleção conduzem a população gradualmente para um alvo que na computação evolucionária é simplesmente a solução de um problema computacional. Em muitos problemas, os algoritmos evolucionários se mostraram capazes de produzirem soluções competitivas com aquelas produzidas por projetos tradicionais ou outras técnicas de busca. As soluções obtidas através da evolução são freqüentemente incomuns, uma vez que são geradas de uma maneira completamente diferente

dos métodos convencionais para projeto e otimização. Este conceito de projeto evolucionário de soluções novas e eficientes também vem sendo adotado no projeto e otimização de circuitos eletrônicos.

Esta dissertação estuda a aplicação dos algoritmos evolucionários inspirados na computação quântica como uma ferramenta para a síntese automática de sistemas sequenciais e em particular no problema de atribuição de estados devido ao impacto deste na complexidade do circuito resultante. Os detalhes do algoritmo implementado para o problema de atribuição de estados e para a síntese automática de circuitos são apresentados no Capítulo 4 e no Capítulo 5, respectivamente. Este algoritmo foi avaliado através da comparação com os resultados apresentados em outros trabalhos nesta área e também com uma ferramenta de síntese do tipo ECAD.

## 1.4 Resumo do Capítulo

Este capítulo apresentou uma visão geral sobre o projeto de sistemas digitais sequenciais, introduzindo as definições básicas para um entendimento do trabalho apresentado nesta dissertação. Na Seção 1.2 foi apresentada uma introdução sobre os sistemas sequenciais. Na Seção 1.3 foi descrita a metodologia tradicional utilizada no projeto de máquinas de estados finitos, as quais são responsáveis pelo controle do comportamento dos sistemas sequenciais. No Capítulo 2 são descritas as principais características de alguns algoritmos evolucionários que foram desenvolvidos antes dos algoritmos evolucionários inspirados na computação quântica. Alguns dos algoritmos descritos, como por exemplo, os algoritmos genéticos e a programação genética, são muito utilizados na síntese evolucionária de sistemas eletrônicos. O entendimento da teoria que define estes algoritmos também ajuda no entendimento dos algoritmos evolucionários inspirados na computação quântica que são introduzidos no Capítulo 3.

## Capítulo 2

# COMPUTAÇÃO EVOLUCIONÁRIA

**E**STE capítulo faz uma apresentação da computação evolucionária. Seu objetivo é destacar os principais conceitos relacionados aos algoritmos evolucionários. Na Seção 2.1 é apresentada uma visão geral da computação evolucionária e algumas das vantagens quando comparada com as técnicas de otimização e busca tradicionais. Na Seção 2.2 os algoritmos genéticos são descritos. Em seguida, na Seção 2.3 é apresentada uma descrição das estratégias evolucionárias, enquanto na Seção 2.4 e na Seção 2.5 são descritas a programação evolucionária e a programação genética, respectivamente.

### 2.1 Introdução

A computação evolucionária se baseia nos princípios da evolução biológica natural, tendo surgido como um conceito geral para o desenvolvimento de novos modelos computacionais para a busca, otimização e projeto em diversas áreas do conhecimento. Os modelos desenvolvidos a partir deste conceito são utilizados na busca de soluções de problemas complexos ou com um espaço de soluções grande (espaço de busca), o que os tornam problemas de difícil modelagem e solução quando se aplicam métodos de otimização convencionais.

Usualmente, agrupados sob o termo computação evolucionária ou algoritmos evolucionários, encontram-se o domínio dos algoritmos genéticos (HOLLAND, 1975), estratégias evolucionárias (RECHENBERG, 1973) (SCHWEFEL, 1977), programação evolucionária (FOGEL; OWENS; WALSH, 1966) e programação genética (KOZA, 1992). Todas estas técnicas compartilham um conceito básico comum de simulação da evolução da estrutura do indivíduo através do processo de seleção, mutação e reprodução.

Os algoritmos evolucionários operam em uma população de soluções potenciais, também chamadas de indivíduos, aplicando o princípio da sobrevivência dos mais aptos para produzir sucessivamente melhores aproximações para uma solução. Em cada iteração do algoritmo evo-

lucionário, chamada de *geração*, um novo conjunto de aproximações é criado pelo processo de seleção de indivíduos de acordo com o seu nível de aptidão no domínio do problema e a reprodução dos mesmos usando operadores de variação. Este processo pode levar para a evolução de populações de indivíduos que são mais bem adaptados ao ambiente do que os indivíduos a partir dos quais foram criados, tal como em uma adaptação natural. Segundo (AKBARZADEH-T; KHORSAND, 2005), existem pelo menos dois aspectos importantes no sucesso de projetos evolucionários: o primeiro é a necessidade de se manter uma diversidade da população para permitir a exploração de novas soluções, e o segundo é a aplicação de operadores capazes de guiarem cada geração de soluções potenciais na direção de soluções melhores, através da investigação das soluções existentes. O procedimento iterativo dos algoritmos evolucionários é ilustrado na Figura 7.

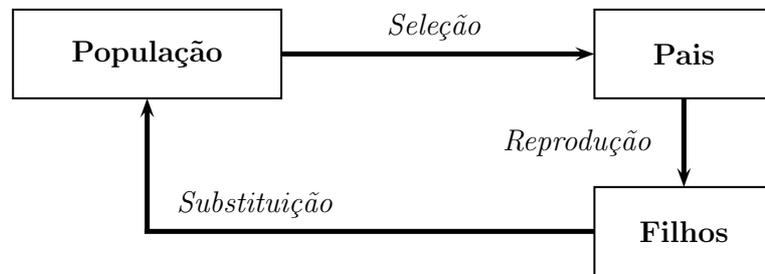


Figura 7: Dinâmica dos algoritmos evolucionários

Uma vantagem da computação evolucionária é que esta é conceitualmente simples. O procedimento pode ser escrito por uma equação de diferenças, conforme apresentado na Equação 5,

$$x[t + 1] = s(v(x[t])), \quad (5)$$

onde  $x[t]$  é a população no tempo  $t$  sob uma representação  $x$ ,  $v$  é o operador de variação aleatório, e  $s$  é o operador de seleção (FOGEL, 1999). Outras vantagens frente às técnicas tradicionais de otimização e busca, como por exemplo, *simulated annealing* (KIRKPATRICK; GELATT; VECCHI, 1983) e a busca tabular (GLOVER, 1990), podem ser listadas como seguem:

- A maioria dos métodos tradicionais de otimização se move de um ponto do espaço de busca para outro usando somente regras determinísticas. O problema com esta estratégia é que existe uma grande chance do algoritmo ficar preso em uma solução local. Os métodos baseados em algoritmos evolucionários começam com um conjunto diverso de soluções potenciais (população) e, tipicamente, uma nova população do mesmo tamanho

é gerada em cada geração subsequente. Isto permite uma exploração paralela, diminuindo a probabilidade do algoritmo ficar preso em uma solução local.

- O desempenho dos algoritmos evolucionários independe da representação, em contraste com outras técnicas numéricas, que podem ser aplicadas apenas para valores contínuos ou outros conjuntos restritos.
- Os algoritmos evolucionários oferecem uma estrutura tal que torna fácil a incorporação do conhecimento prévio sobre o problema. Através da incorporação destas informações, a busca evolucionária pode ser direcionada para um determinado espaço de busca, resultando numa exploração mais eficiente.
- Utilizam a probabilidade, ao contrário das regras de transição determinísticas. Apesar dos algoritmos evolucionários serem métodos probabilísticos, eles não são estritamente uma busca aleatória. Os operadores estocásticos utilizados nas operações sobre a população direcionam a busca para a região do espaço de busca onde se acredita ter soluções com melhores aptidões.
- Os algoritmos evolucionários podem ser combinados com técnicas de otimização tradicionais. Isto pode ser tão simples quanto a utilização de uma minimização de gradiente utilizada após uma busca primária com um algoritmo evolucionário, ou pode envolver a aplicação simultânea de outros algoritmos.
- A avaliação de cada solução pode ser manipulada em paralelo e somente a seleção, que requer pelo menos a competição entre pares, necessita de algum processamento serial. O paralelismo implícito não é possível na maioria dos algoritmos de otimização global.
- Os métodos tradicionais de otimização não são robustos com relação às mudanças dinâmicas no ambiente do problema e freqüentemente requerem um reinício completo para prover uma solução. Ao contrário, os algoritmos evolucionários podem ser usados para adaptarem as soluções às mudanças de circunstância.
- Os algoritmos evolucionários possuem a habilidade de lidar com problemas para os quais não existem pessoas especialistas, embora o conhecimento humano possa ser usado quando está disponível.

## 2.2 Algoritmos Genéticos

Um fluxograma típico de um algoritmo genético é mostrado na Figura 8. O algoritmo genético básico é muito genérico e existem muitos aspectos que podem ser implementados de forma diferente em função do tipo de problema, como por exemplo, o tipo de codificação das soluções, a estratégia de seleção e os tipos de operadores genéticos para a reprodução. Em geral, os algoritmos genéticos utilizam uma seqüência de bits para a representação das soluções. Esta seqüência de bits é chamada indivíduos ou cromossomos. Os indivíduos da população participam de um processo que simula a evolução. A simples operação de manipulação dos bits, também chamados de genes, permite a implementação dos operadores genéticos responsáveis pela reprodução dos indivíduos.

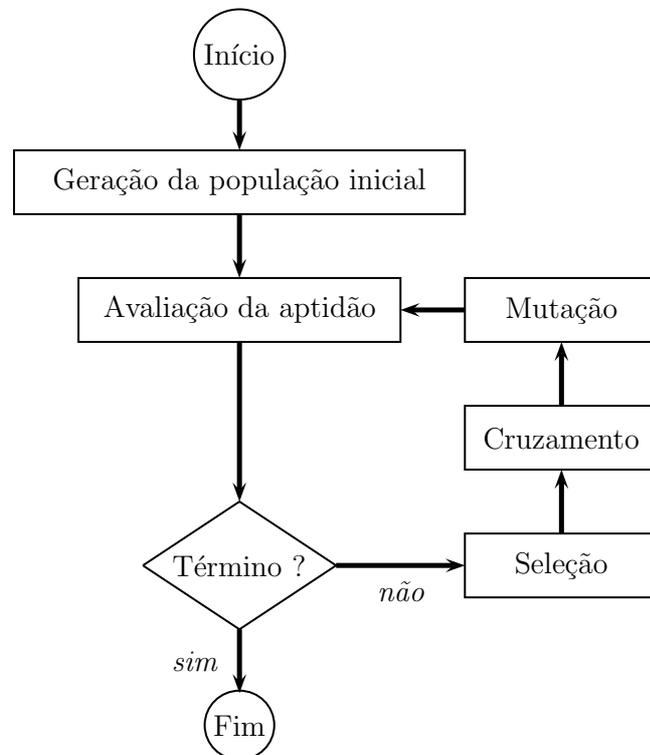


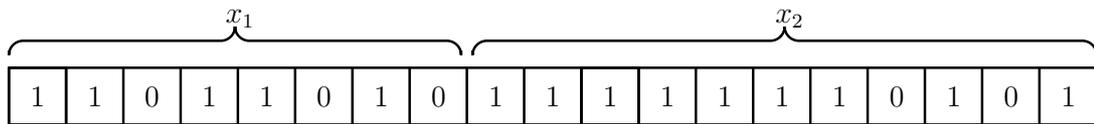
Figura 8: Fluxograma básico de um algoritmo genético

### 2.2.1 Codificação

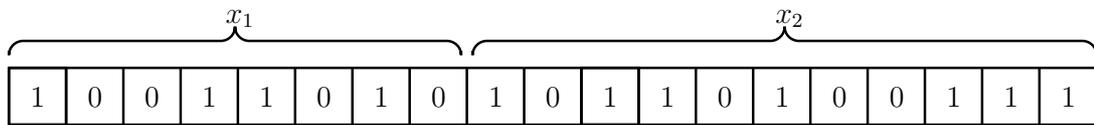
Em uma aplicação típica dos algoritmos genéticos, o problema é transformado em um conjunto de características genéticas que representam os parâmetros a serem otimizados. O conjunto ordenado destas características genéticas caracteriza um único indivíduo ou cromossomo. Por exemplo, se a tarefa é a minimização da função dada na Equação 6,

$$\min f(x_1, x_2) = (x_1 - 5)^2 + (x_2 - 2)^2, -3 \leq x_1 \leq 3, -8 \leq x_2 \leq 8, \quad (6)$$

então, conhecendo-se o espaço de busca de  $x_1$  e  $x_2$ , cada variável é representada utilizando-se cadeias binárias adequadas, as quais são concatenadas formando então um cromossomo. A codificação binária é a mais utilizada principalmente devido à sua relativa simplicidade. A Figura 9 demonstra uma possível representação das soluções para a otimização da função dada na Equação 6.



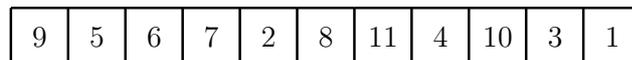
(a) Cromossomo A



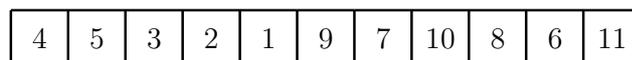
(b) Cromossomo B

Figura 9: Exemplo de cromossomos com codificação binária

Além da codificação binária, existem outros tipos de codificação, como por exemplo, a codificação diretamente com inteiros ou com números reais e a codificação para problemas de permutação e controle (DEB, 2000). A escolha da codificação mais adequada depende diretamente do tipo de problema. No caso de problemas de ordenamento, como por exemplo, o Problema do Caixeiro Viajante (do inglês, *Travelling Salesman Problem - TSP*) (GAREY; JOHNSON, 1979), o tipo de codificação mais utilizado é a codificação com permutação. Neste tipo de codificação, cada cromossomo é uma cadeia de números que representa números em uma seqüência. Um cromossomo que use a codificação com permutação para o problema TSP com 11 cidades irá parecer como mostrado na Figura 10.



(a) Cromossomo A



(b) Cromossomo B

Figura 10: Exemplo de cromossomos codificados com permutação

Os cromossomos representam a ordem das cidades em que o caixeiro irá visitá-las. Um

cuidado especial deve ser tomado para assegurar que o cromossomo represente uma seqüência real após ter sido submetido às operações de cruzamento e mutação. A representação em ponto flutuante (MICHALEWICZ, 1992) é muito utilizada em otimização numérica, por exemplo, para a codificação dos pesos de uma rede neural (KOEHN, 1994).

### 2.2.2 Estratégias de Seleção

O operador de seleção baseia-se no princípio da sobrevivência dos mais aptos que é inspirado na seleção natural observada na evolução biológica. A aptidão de um indivíduo em relação a uma determinada especificação determina a probabilidade deste indivíduo ser selecionado e contribuir para a criação de indivíduos em uma próxima geração. Algumas das formas de seleção mais populares são discutidos a seguir.

#### 2.2.2.1 Seleção pelo giro da roleta

A forma de seleção mais simples é a seleção pelo giro da roleta, também chamada de *amostragem estocástica com substituição*. Nesta forma, os indivíduos são selecionados de acordo com as suas aptidões. Quanto maior a aptidão de um determinado indivíduo, maior a sua chance de ser selecionado. Esta técnica é análoga a uma roleta circular onde os indivíduos são mapeados em segmentos contíguos de uma linha, de tal forma que estes segmentos sejam proporcionais às aptidões dos indivíduos. Através do giro da roleta ilustrada na Figura 11, um indivíduo é selecionado. Os indivíduos com as maiores porções da roleta (mais aptos) têm uma probabilidade maior de serem selecionados.

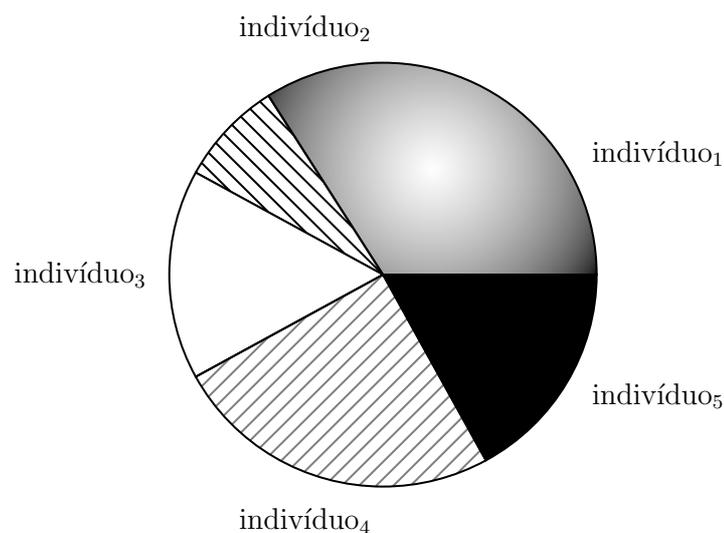


Figura 11: Seleção pelo giro da roleta

O processo de seleção pelo giro da roleta pode ser simulado através das seguintes etapas:

1. as aptidões de todos os indivíduos da população devem ser somadas e o valor obtido armazenado em  $S$ ;
2. um número aleatório  $f$ , selecionado do intervalo  $[0, S]$  deve ser gerado;
3. em seguida as aptidões de cada um dos indivíduos da população devem ser somadas, sendo  $\sigma$  o valor da soma parcial;
4. se  $\sigma \geq f$ , então o indivíduo corrente é selecionado;
5. se o número desejado de indivíduos ainda não foi obtido então retorne para a segunda etapa.

### 2.2.2.2 Seleção por torneio

Nesta forma, um número de indivíduos é escolhido aleatoriamente da população para participarem de um torneio e os vencedores deste torneio são selecionados. O parâmetro para a seleção por torneio é o tamanho do torneio, o qual depende de outros aspectos, como por exemplo, o tipo de problema e o tamanho da população. O tamanho do torneio pode variar de 2 até o número de indivíduos da população. O processo é iniciado com a escolha aleatória dos indivíduos que irão participar do torneio. A aptidão dos indivíduos não é considerada nesta etapa e todos os indivíduos possuem as mesmas chances de serem escolhidos. O torneio é vencido pelo indivíduo com a maior aptidão. Este processo é repetido tantas vezes quanto forem os indivíduos necessários. A Figura 12 ilustra o mecanismo da seleção por torneio, para um torneio com tamanho igual a 2.

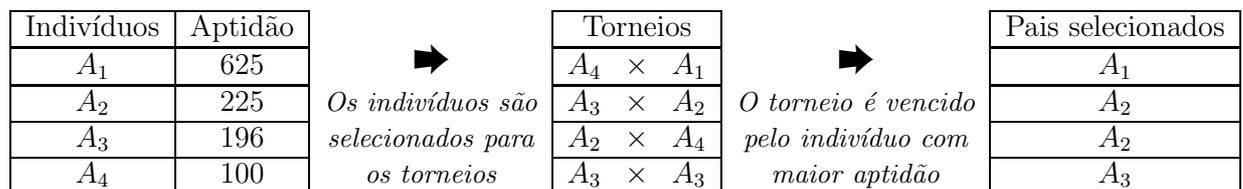


Figura 12: Seleção por torneio

### 2.2.2.3 Elitismo

Para não permitir a perda do melhor indivíduo que pode acontecer durante a etapa de reprodução, a técnica do elitismo é usada para preservar o indivíduo mais apto. Com o elitismo, o melhor indivíduo, ou alguns dos melhores, são simplesmente copiados para a nova população.

O elitismo pode melhorar o desempenho do algoritmo genético, pois previne a perda da melhor solução encontrada.

### 2.2.3 Operadores de Reprodução

O *cruzamento* e a *mutação* são dois operadores básicos dos algoritmos genéticos. O desempenho dos algoritmos genéticos depende fortemente destes operadores. O tipo e implementação destes operadores dependem da codificação e também do problema, por isso existem várias maneiras diferentes de se implementar o cruzamento e a mutação. A seguir são demonstrados alguns dos métodos mais populares da implementação destes operadores para diferentes esquemas de codificação.

#### 2.2.3.1 Cruzamento

O cruzamento seleciona genes de dois indivíduos “pais”, criando novos indivíduos “filhos” conforme pode ser visto na Figura 13. A maneira mais simples de se fazer isso é escolhendo aleatoriamente algum ponto de corte, sendo que os segmentos à direita deste ponto são permutáveis.

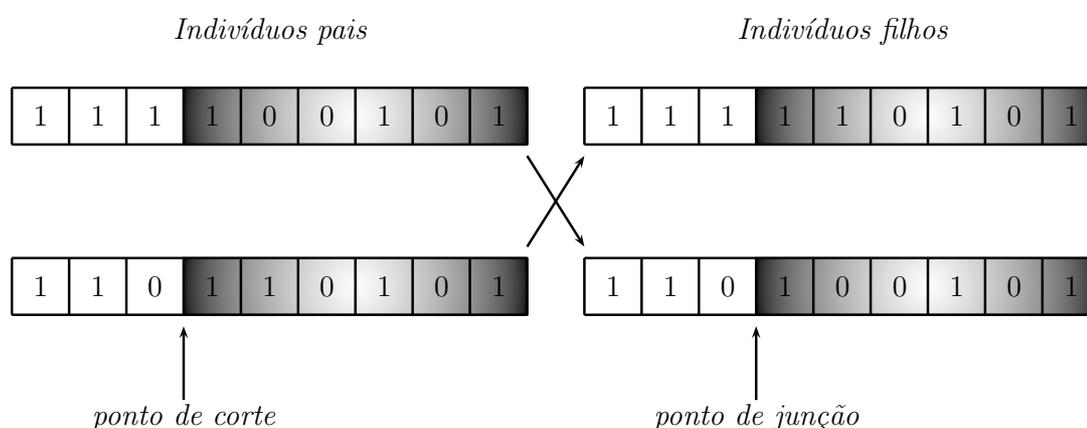


Figura 13: Exemplo da operação de cruzamento simples

A Figura 14 ilustra algumas das técnicas de cruzamento. O cruzamento de dois pontos consiste na seleção aleatória de dois pontos de corte, a parte do início do indivíduo “filho” até o primeiro ponto de corte é copiada de um indivíduo “pai”, a parte do primeiro até o segundo ponto de corte é copiada do segundo indivíduo “pai” e o restante é copiado do primeiro indivíduo “pai”. No cruzamento uniforme os bits são copiados aleatoriamente de um dos dois indivíduos “pais”. A utilização de um cruzamento elaborado especificamente para um deter-

minado problema pode melhorar de forma significativa o desempenho do algoritmo genético (ABRAHAM; NEDJAH; MOURELLE, 2005).

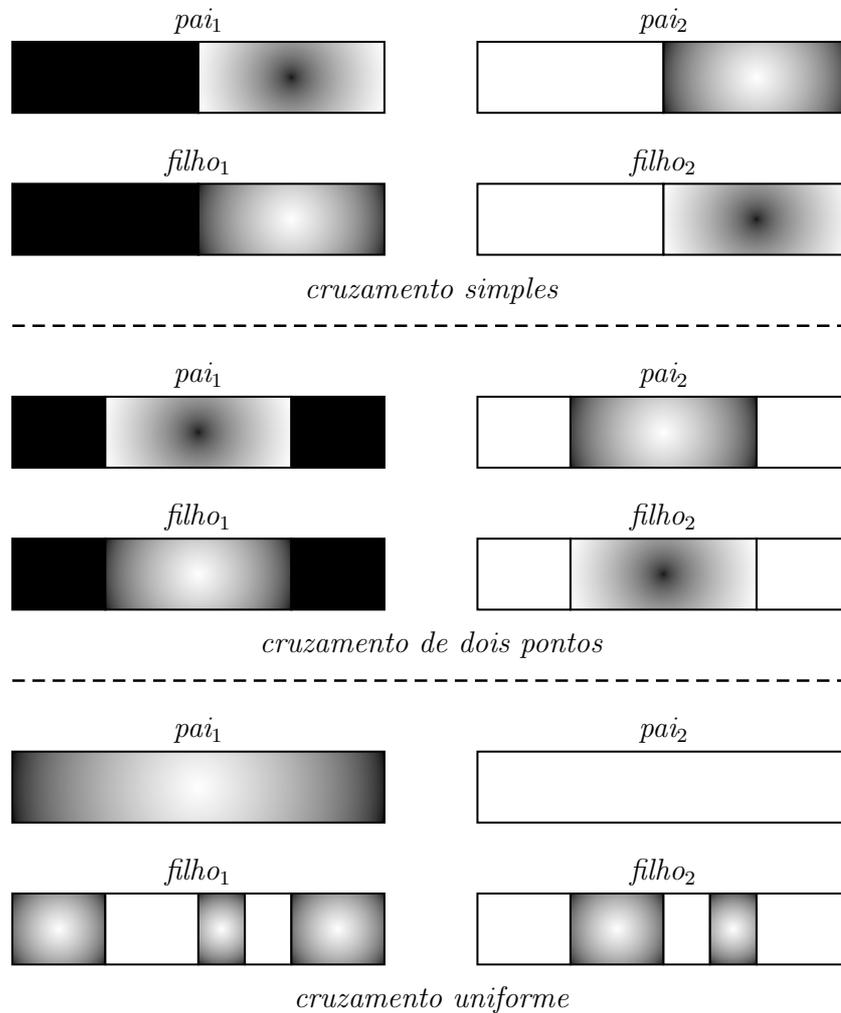


Figura 14: Tipos de operadores cruzamento

### 2.2.3.2 Mutaç o

A muta o consiste na altera o de alguns genes de alguns indiv duos da popula o obtidos ap s o cruzamento. A opera o de cruzamento tem uma caracter stica desfavor vel a longo prazo na busca do  timo global por ser um operador que torna a popula o mais homog nea. Ap s algumas gera es, a popula o ter  indiv duos semelhantes, que habitam uma regi o do espa o de busca, onde existe um  timo que pode n o ser necessariamente um  timo global, levando a uma converg ncia prematura. Para contornar o problema,   necess rio incluir no algoritmo gen tico um operador que torne a popula o mais heterog nea. Esse operador  

a *mutação*. A mutação muda aleatoriamente alguns dos novos indivíduos criados pelo cruzamento. Para a codificação binária a mutação é realizada através da troca de alguns bits selecionados aleatoriamente de 1 para 0 ou de 0 para 1. Assim como o cruzamento, a mutação também depende da codificação. Uma operação de mutação simples é mostrada na Figura 15.

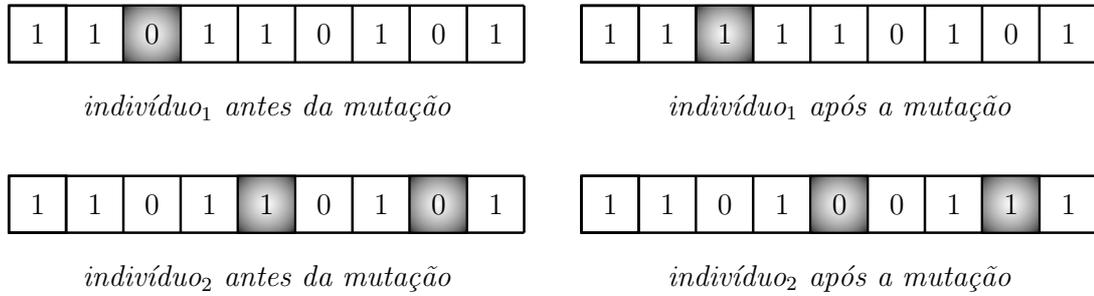


Figura 15: Operação de mutação

## 2.3 Estratégias Evolucionárias

A primeira estratégia evolucionária foi desenvolvida por Rechenberg (RECHENBERG, 1973). As estratégias evolucionárias tendem a ser usadas em experimentos empíricos que são difíceis de se modelar matematicamente. Uma vez construído o sistema a ser otimizado, a estratégia evolucionária é usada para encontrar os parâmetros ótimos de configuração. As estratégias evolucionárias se concentram na tradução do mecanismo fundamental da evolução biológica para problemas de otimização. Os indivíduos manipulados nas estratégias evolucionárias são compostos minimamente por dois vetores de números reais: *parâmetros objetivos* -  $po$  e *parâmetros estratégicos* -  $pe$ . Os parâmetros objetivos (genes) definem as características dos indivíduos, as quais são manipuladas através da aplicação dos operadores evolucionários: *mutação* e *cruzamento*. Os parâmetros estratégicos controlam a mutação dos parâmetros objetivos. Ambos os parâmetros, objetivos e estratégicos, formam a estrutura de dados de um único indivíduo. Uma população  $P$  de  $n$  cromossomos pode ser descrita como  $P = (c_1, c_2, \dots, c_{n-1}, c_n)$ , onde o  $i$ -ésimo cromossomo  $c_i$  é definido como  $c_i = (po, pe)$  com  $po = (o_1, o_2, \dots, o_{n-1}, o_n)$  e  $pe = (e_1, e_2, \dots, e_{n-1}, e_n)$ .

### 2.3.1 Mutação nas Estratégias Evolucionárias

O operador de mutação é definido como um componente de adição de números gerados aleatoriamente de distribuição normal. Tanto os parâmetros objetivos quanto os parâmetros estratégicos do indivíduo podem sofrer mutação. Um vetor de parâmetros objetivos que sofre

mutação é calculado como  $po(mut) = po + N_0(pe)$ , onde  $N_0$  é um vetor aleatório com distribuição normal com média zero e desvio padrão  $pe_i (i = 1, 2, \dots, n)$ . Normalmente, o valor do desvio padrão  $pe_i$  é definido de forma adaptativa de uma população para outra. Esta adaptação pode ser implementada, por exemplo, por  $pe(mut) = (e_1 \times A_1, e_2 \times A_2, \dots, e_{n-1} \times A_{n-1}, e_n \times A_n)$ , onde  $A_i$  é escolhido aleatoriamente como  $\alpha$  ou  $1/\alpha$ . Dependendo do valor sorteado para a variável  $M$  no intervalo  $[0, 1]$ , o valor de  $A_i = \alpha$  se  $M < 0,5$  e  $A_i = 1/\alpha$  se  $M \geq 0,5$ . O parâmetro  $\alpha$  é usualmente referenciado como *valor de adaptação* dos parâmetros estratégicos.

### 2.3.2 Cruzamento ou Recombinação nas Estratégias Evolucionárias

Para dois cromossomos  $c_1 = (po^1, pe^1)$  e  $c_2 = (po^2, pe^2)$  o operador *cruzamento* ou *recombinação* é definido como  $R(c_1, c_2) = c = (po, pe)$  com  $po = (o_1, o_2, \dots, o_{n-1}, o_n)$  e  $pe = (e_1, e_2, \dots, e_{n-1}, e_n)$ , onde

$$\begin{aligned} o_i &= \begin{cases} o_i^1, & \chi \leq \frac{1}{2} \\ o_i^2, & \chi > \frac{1}{2} \end{cases} \quad \forall i \in \{1, \dots, n\} \\ e_i &= \begin{cases} e_i^1, & \chi \leq \frac{1}{2} \\ e_i^2, & \chi > \frac{1}{2} \end{cases} \quad \forall i \in \{1, \dots, n\} \end{aligned} \quad (7)$$

sendo que  $\chi$  denota uma variável aleatória gerada no intervalo  $[0, 1]$  que é escolhida para cada componente dos vetores  $po$  e  $pe$ .

### 2.3.3 Tipos de Estratégias Evolucionárias

Seja  $P$  o número de indivíduos “pais” na geração 1 e seja  $C$  o número de indivíduos “filhos” na geração  $i$ . Existem basicamente quatro tipos diferentes de estratégias evolucionárias:  $P$ ,  $C$ ,  $P + C$ ,  $P/R, C$  e  $P/R + C$  conforme discutido a seguir. Estas estratégias são diferentes principalmente na maneira como são selecionados os “pais” para as próximas gerações e o uso do operador de cruzamento.

**Estratégia  $P, C$ :** Os  $P$  “pais” produzem  $C$  “filhos” usando a mutação. Os valores das aptidões são calculados para cada um dos  $C$  “filhos” e os melhores  $P$  “filhos” se tornam os “pais” na próxima geração. Note que  $(C \geq P)$ .

**Estratégia  $P + C$ :** Os  $P$  “pais” produzem  $C$  “filhos” usando a mutação. Os valores das aptidões são calculados para cada um dos  $C$  “filhos” e os melhores  $P$  indivíduos dentre os “pais” e os “filhos” se tornam os “pais” na próxima geração.

**Estratégia  $P/R, C$ :** Os  $P$  “pais” produzem  $C$  “filhos” usando a mutação e o cruzamento. Os valores das aptidões são calculados para cada um dos  $C$  “filhos” e os melhores  $P$  “filhos” se tornam os “pais” na próxima geração. Note que ( $C \geq P$ ). Exceto pelo uso do operador cruzamento, esta é exatamente a mesma que a estratégia  $P, C$ .

**Estratégia  $P/R + C$ :** Os  $P$  “pais” produzem  $C$  “filhos” usando a mutação e o cruzamento. Os valores das aptidões são calculados para cada um dos  $C$  “filhos” e os melhores  $P$  indivíduos dentre os “pais” e os “filhos” se tornam os “pais” na próxima geração. Exceto pelo uso do operador cruzamento, esta é exatamente a mesma que a estratégia  $P + C$ .

## 2.4 Programação Evolucionária

O livro de Fogel, Owens e Walsh (FOGEL; OWENS; WALSH, 1966) foi uma das primeiras publicações sobre a programação evolucionária. No livro, automata de estado finito são evoluídos para predizerem cadeias de símbolos geradas a partir de processos de Markov e séries temporais não estacionárias.

O método básico da programação evolucionária envolve as seguintes etapas:

1. Escolha de uma população inicial composta de possíveis soluções escolhidas de forma aleatória. O número de soluções em uma população é relevante para a velocidade de otimização, mas nenhuma resposta definitiva está disponível sobre quantas soluções são apropriadas.
2. Novos indivíduos são criados pela mutação. Cada novo indivíduo é avaliado através do cálculo da sua aptidão. Tipicamente, um torneio estocástico é realizado para determinar  $N$  soluções para serem retidas para a população de soluções. O método da programação evolucionária tipicamente não usa nenhum tipo de cruzamento como operador genético.

Quando efetua-se uma comparação entre a programação evolucionária e o algoritmo genético, pode-se identificar as seguintes diferenças:

1. O algoritmo genético é implementado com cadeias de bits ou caracteres para a representação do cromossomo. Na programação evolucionária não existe nenhuma restrição para a representação. Em muitos casos, a representação decorre do problema.
2. A programação evolucionária tipicamente utiliza um operador de mutação adaptativo no qual a severidade da mutação é freqüentemente reduzida à medida que o ótimo global vai

sendo atingido, enquanto o algoritmo genético utiliza um operador de mutação pré-fixado. Dentre as formas para adaptação do tamanho do passo da mutação, o mais estudado tem sido a técnica “meta-evolucionária” na qual a variância da distribuição da mutação está sujeita a mutação através de um operador de mutação de variância fixa que evolui junto com a solução.

Por outro lado, quando se compara a programação evolucionária com as estratégias evolucionárias, pode-se identificar as seguintes diferenças:

1. Quando implementados para resolverem problemas de otimização de funções de valores reais, ambos tipicamente operam com valores reais e utilizam operadores de reprodução adaptativos.
2. A programação evolucionária tipicamente utiliza um torneio estocástico para seleção, enquanto as estratégias evolucionárias tipicamente utilizam uma seleção determinística.
3. A programação evolucionária não utiliza o operador de cruzamento, enquanto as estratégias evolucionárias (estratégias  $P/R, C$  e  $P/R + C$ ) podem utilizar o operador de cruzamento. Porém, a eficácia do uso deste operador depende do problema em que está sendo aplicado.

## 2.5 Programação Genética

A técnica da programação genética proporciona uma maneira para a criação automática de programas de computador a partir de uma especificação do problema em questão em alto nível (KOZA, 1992). A programação genética atinge este objetivo da programação automática através da criação genética de uma população de programas de computador inspirada nos princípios de Darwin da seleção natural e operações inspiradas na biologia. As operações incluem a maioria das técnicas discutidas nas seções anteriores. A principal diferença entre a programação genética e os algoritmos genéticos é a representação da solução.

A programação genética é uma extensão do aprendizado evolucionário no espaço dos programas de computador. Na programação genética os membros da população de indivíduos não são cadeias de caracteres com comprimento fixo que codificam as possíveis soluções do problema, mas sim programas que, quando executados, são as potenciais soluções do problema. Estes programas são representados na programação genética como árvores, em vez de linha de códigos. Por exemplo, o programa “ $a + b * f(4, a, c)$ ” seria representado como mostrado na

Figura 16. Os conjuntos de terminais e de funções também são componentes importantes da programação genética. Eles são o alfabeto dos programas a serem construídos. O conjunto de terminais consiste de variáveis (por exemplo,  $a$ ,  $b$  e  $c$  na Figura 16) e constantes (por exemplo, 4 na Figura 16).

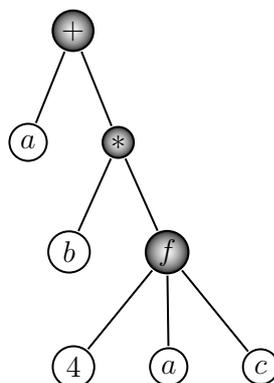


Figura 16: Uma estrutura em árvore simples da programação genética

Colocando de uma forma generalizada, o procedimento da programação genética pode ser resumido como segue:

1. Uma população inicial de composições aleatórias das funções e terminais é gerada;
2. O valor da aptidão de cada indivíduo da população é calculado;
3. Usando uma estratégia de seleção e operadores de reprodução, são gerados dois “filhos”;
4. O procedimento é repetido até que a solução requerida seja encontrada ou a condição de término seja atingida.

### 2.5.1 Codificação do Programa de Computador

Segundo (KOZA, 1992), uma árvore de análise sintática é uma estrutura adequada para a interpretação por um programa de computador. Nesta estrutura, as funções são escritas como nós e seus argumentos como folhas. Uma sub-árvore é a parte de uma árvore que está sob um nó interno desta árvore. Se esta sub-árvore é cortada, o nó interno se torna um nó raiz e esta sub-árvore se torna uma árvore válida por si só.

### 2.5.2 Reprodução de Programas de Computador

A criação de um indivíduo “filho” através da operação de cruzamento é realizada removendo-se um fragmento de um dos indivíduos “pais” e depois inserindo um fragmento do outro indivíduo

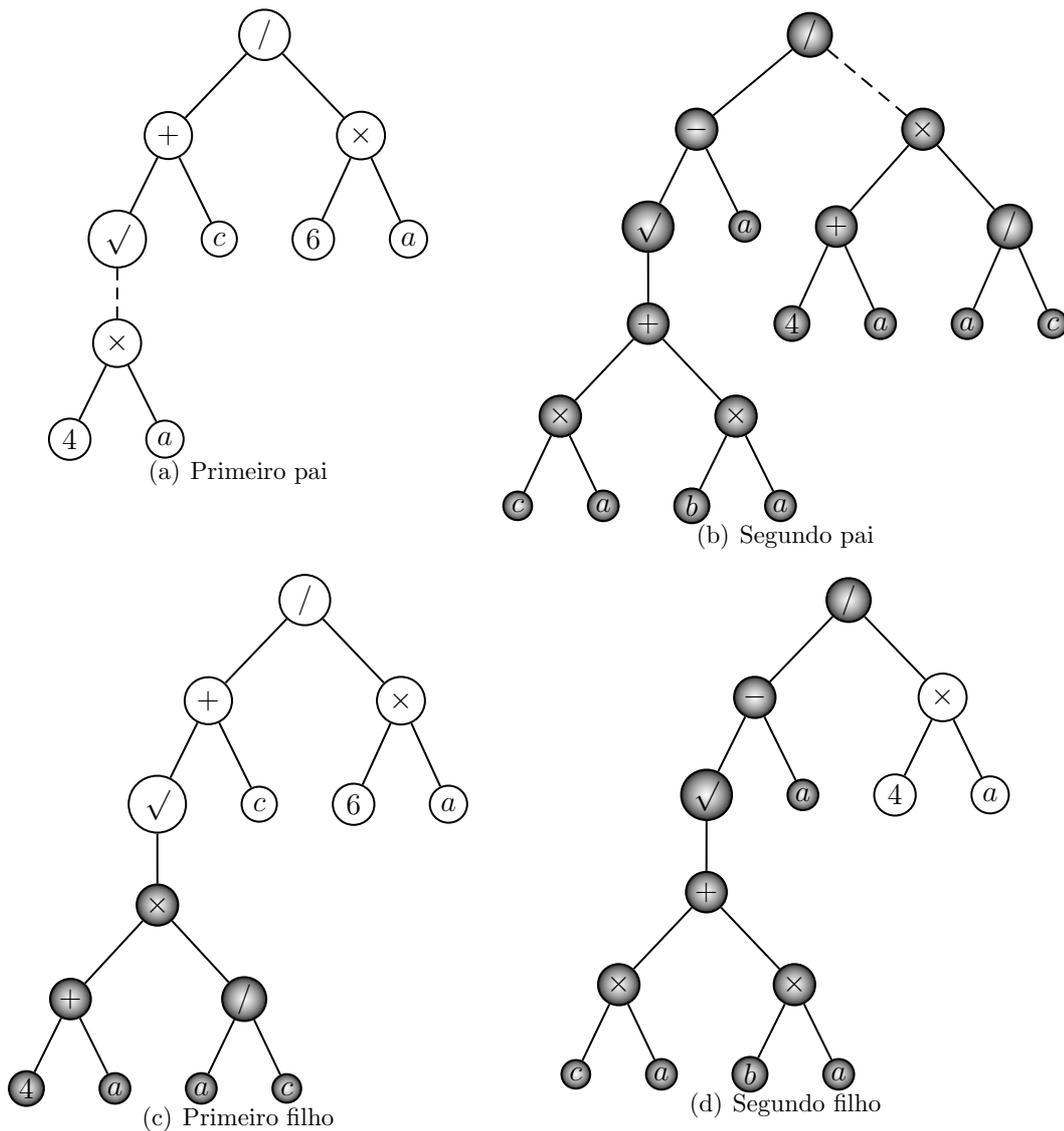


Figura 17: Ilustração da operação de cruzamento

“pai”. O segundo “filho” é produzido de uma maneira simétrica. Uma operação de cruzamento simples é mostrada na Figura 17. Na programação genética a operação de cruzamento é implementada tomando-se sub-árvores selecionadas aleatoriamente nos indivíduos e realizando-se a troca das mesmas.

A *mutação* é uma outra característica importante da programação genética. Dois tipos de mutação são normalmente utilizados. O tipo mais simples é a substituição de uma função ou de um terminal por outra função ou terminal, respectivamente. No segundo tipo uma sub-árvore inteira pode ser substituída por outra sub-árvore. A Figura 18 apresenta o conceito da mutação.

A programação genética utiliza uma estrutura de dados fácil de manipular e avaliar e que

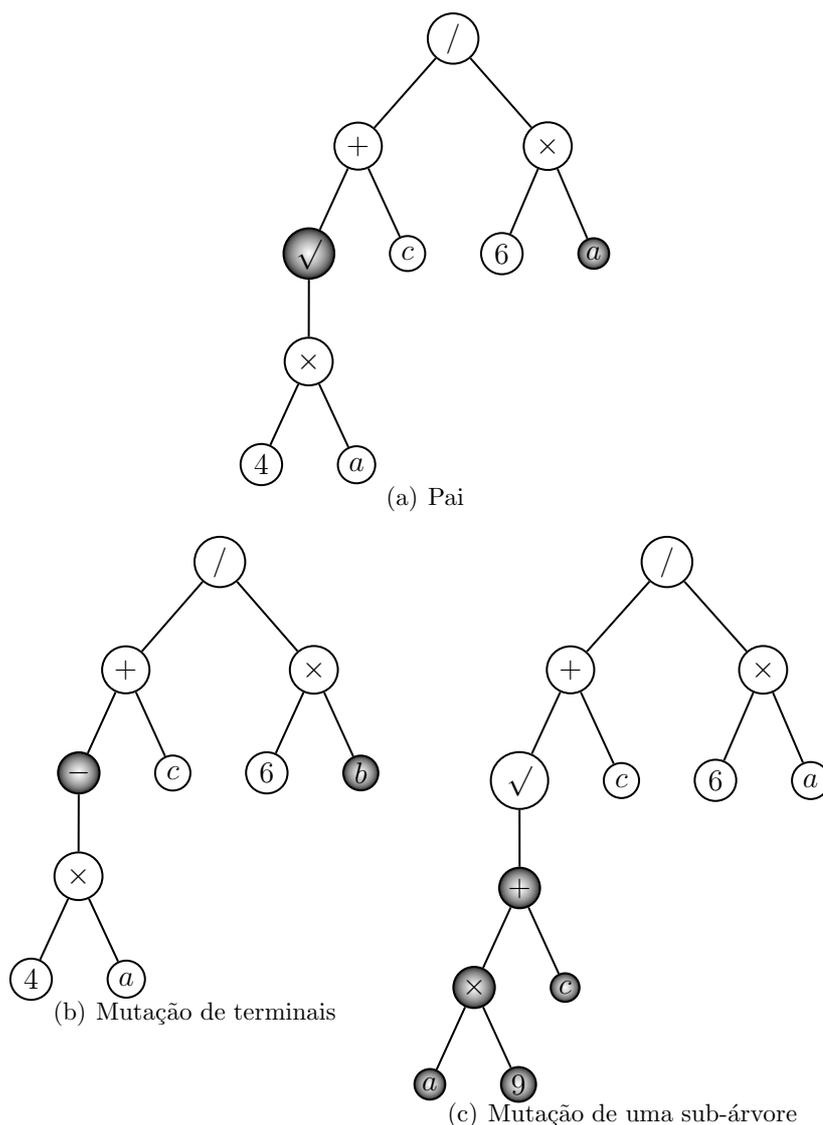


Figura 18: Ilustraç o da operaç o de mutaç o

  robusta para manipulaç es estruturais. O conjunto de funç es e terminais que ser o usados em um problema espec fico deve ser cuidadosamente escolhido. Se o conjunto de funç es n o   poderoso o suficiente, a soluç o pode ser complexa ou pode at  mesmo n o ser encontrada. Assim como em outras t cnicas evolucion ria, a geraç o da primeira populaç o de indiv duos   importante para o sucesso da implementaç o da programaç o gen tica. Alguns fatores que tamb m influenciam o desempenho do algoritmo   o tamanho da populaç o, a porcentagem de indiv duos que participam das operaç es de cruzamento e mutaç o, a profundidade m xima para os indiv duos iniciais e a profundidade m xima permitida para os “filhos” gerados.

---

## 2.6 Resumo do Capítulo

Este capítulo apresentou uma breve introdução dos métodos evolucionários clássicos que fazem parte da computação evolucionária. Foram apresentados os principais conceitos que envolvem os algoritmos genéticos, a estratégia evolucionária, a programação evolucionária e a programação genética. No capítulo seguinte será descrita uma nova classe de algoritmos evolucionários que busca inspiração nos princípios da computação quântica.

## Capítulo 3

# COMPUTAÇÃO EVOLUCIONÁRIA INSPIRADA NA COMPUTAÇÃO QUÂNTICA

**N**ESTE capítulo será apresentada uma classe de algoritmos que surgiu recentemente, denominada de algoritmos evolucionários inspirados na computação quântica. Estes algoritmos utilizam alguns conceitos da computação quântica, os quais são descritos na Seção 3.1. Na Seção 3.2 é apresentada a estrutura dos algoritmos evolucionários inspirados na computação quântica.

### 3.1 Introdução à Computação Quântica

A computação quântica é uma área de pesquisa recente que explora as possibilidades de se aplicar as propriedades da mecânica quântica na computação. O principal ganho dos computadores quânticos é a possibilidade de se realizar certas tarefas em um tempo eficiente, as quais levariam um tempo impraticável quando realizadas em computadores tradicionais.

A primeira proposta de aplicação de uma propriedade quântica na execução de rotinas computacionais foi apresentada por Richard Feynman, em 1982. Ele mostrou que um computador tradicional levaria um tempo longo para simular certos fenômenos quânticos e que apenas um *simulador quântico universal*, isto é, uma máquina que fizesse uso de propriedades quânticas, seria capaz de realizá-lo de forma eficiente (FEYNMAN, 1982). Em 1985, David Deutsch descreveu pela primeira vez um computador quântico universal (DEUTSCH, 1985). Foi ele também quem propôs, em 1989, um modelo de circuitos quânticos similar aos modelos de circuitos lógicos/digitais, baseado em bits quânticos (análogo quântico ao bit clássico) e portas lógicas quânticas (DEUTSCH, 1989).

Em 1994, Peter Shor publicou um algoritmo para computadores quânticos capaz de

fatorar números grandes muito mais rápido do que com máquinas clássicas (SHOR, 1994). A fatoração de números grandes é a base de alguns sistemas de criptografia, como, por exemplo, o sistema RSA. O algoritmo de Shor despertou um enorme interesse nos computadores quânticos, inclusive fora da comunidade científica. A partir deste interesse, surgiram outros algoritmos quânticos importantes, tais como os algoritmos para busca em banco de dados de Lov Grover (GROVER, 1996) e para solução de problemas da área de teoria dos números de Sean Hallgren (HALLGREN, 2002).

Atualmente, as pesquisas na área da computação quântica estão voltadas principalmente para a construção do computador quântico. Os algoritmos de Shor e Grover já puderam ser testados com sucesso em laboratórios através de sistemas com alguns bits quânticos, porém estes métodos ainda são difíceis de serem implementados em grande escala. Alguns pesquisadores acreditam que até o ano de 2020 será possível a fabricação de computadores quânticos práticos (BALL, 2006).

### 3.1.1 Conceitos da Computação Quântica

Nesta seção são descritos alguns dos conceitos da computação quântica que são importantes para o entendimento dos algoritmos evolucionários inspirados na computação quântica.

#### 3.1.1.1 Bits quânticos

A menor quantidade de informação que pode ser representada na computação quântica é o bit quântico ou q-bit. Em princípio, os q-bits podem ser representados por qualquer sistema quântico de dois estados (HEY, 1999). Os exemplos mais comuns são: estados de polarização de um fóton (horizontal ou vertical), elétrons em poços quânticos e *spins* nucleares.

Na computação quântica, os valores 0 e 1 de um bit clássico são substituídos pelos vetores de estado  $|0\rangle$  e  $|1\rangle$  de um q-bit. Estes vetores são usualmente escritos usando uma notação especial introduzida pelo físico britânico Paul Dirac, a notação *braket* (DIRAC, 1958).

Os vetores de estado de um q-bit são representados por:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{e} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \quad (8)$$

Enquanto, o bit clássico pode estar somente em um de dois estados básicos mutuamente exclusivos, o estado genérico de um q-bit pode ser representado pela combinação linear dos vetores de estado  $|0\rangle$  e  $|1\rangle$ , ou seja,

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle, \quad (9)$$

onde  $\alpha$  e  $\beta$  são números complexos. Os vetores de estado  $|0\rangle$  e  $|1\rangle$  formam uma base canônica e o vetor  $|\psi\rangle$  representa a superposição destes vetores, com amplitudes  $\alpha$  e  $\beta$ . A normalização unitária do estado do  $q$ -bit garante que:

$$|\alpha|^2 + |\beta|^2 = 1. \quad (10)$$

A interpretação física do  $q$ -bit é que ele pode estar simultaneamente nos estados  $|0\rangle$  e  $|1\rangle$ , o que permite que uma quantidade de informação infinita possa ser armazenada no estado  $|\psi\rangle$ . Entretanto, o ato de se fazer uma medição de um estado quântico faz com que este entre em colapso, assumindo um estado único (NARAYANAN, 1999). O  $q$ -bit assume o estado  $|0\rangle$ , com probabilidade  $|\alpha|^2$ , ou o estado  $|1\rangle$ , com probabilidade  $|\beta|^2$ .

### 3.1.1.2 Sistemas quânticos

O estado de sistemas quânticos com mais de um  $q$ -bit é definido pelo produto tensorial dos estados de cada um dos seus componentes (PORTUGAL *et al.*, 2004). Portanto, os sistemas quânticos com  $m$ - $q$ -bits podem representar  $2^m$  estados ao mesmo tempo. O produto tensorial de dois estados

$$|\psi\rangle = \begin{bmatrix} \psi_1 \\ \psi_2 \\ \vdots \\ \psi_m \end{bmatrix} \quad \text{e} \quad |\varphi\rangle = \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \vdots \\ \varphi_p \end{bmatrix},$$

denotado por  $|\psi\rangle \otimes |\varphi\rangle$  ou  $|\psi\varphi\rangle$ , tem como resultado o estado  $|\chi\rangle$  com  $mp$ -linhas, representado por

$$|\chi\rangle = \begin{bmatrix} \psi_1\varphi_1 \\ \psi_1\varphi_2 \\ \vdots \\ \psi_1\varphi_p \\ \psi_2\varphi_1 \\ \psi_2\varphi_2 \\ \vdots \\ \psi_2\varphi_p \\ \vdots \\ \psi_m\varphi_1 \\ \psi_m\varphi_2 \\ \vdots \\ \psi_m\varphi_p \end{bmatrix}, \quad (11)$$

onde  $\psi_i\varphi_j$  é o produto usual dos números complexos.

Um estado genérico  $|\psi\rangle$  de dois  $q$ -bits é descrito pela superposição dos estados  $|00\rangle$ ,  $|01\rangle$ ,  $|10\rangle$  e  $|11\rangle$ , ou seja,

$$|\psi\rangle = \alpha_1 |00\rangle + \alpha_2 |01\rangle + \alpha_3 |10\rangle + \alpha_4 |11\rangle, \quad (12)$$

onde

$$\sum_{i=1}^4 |\alpha_i|^2 = 1. \quad (13)$$

Considerando, para efeito de notação,  $x_1 = |00\rangle$ ,  $x_2 = |01\rangle$ ,  $x_3 = |10\rangle$  e  $x_4 = |11\rangle$ , um sistema quântico de  $m$ -q-bits pode ser representado por

$$|\psi\rangle = \sum_{i=1}^{2^m} \alpha_i x_i,$$

com as amplitudes  $\alpha_i$  atendendo a

$$\sum_{i=1}^{2^m} |\alpha_i|^2 = 1.$$

Além da superposição de estados, outra propriedade fundamental na computação quântica é o emaranhamento ou entrelaçamento quântico, fenômeno no qual estados quânticos permanecem correlacionados mesmo estando espacialmente separados. Quando dois ou mais q-bits estão emaranhados, o estado de cada um deles é indefinido. É possível, em princípio, conhecer as características globais do sistema com precisão, mas não de cada uma das partes. Um estado de dois q-bits pode ou não ser o resultado do produto tensorial de estados de um q-bit. Por exemplo, o estado  $|01\rangle$  pode ser descrito como produto tensorial dos estados  $|0\rangle$  e  $|1\rangle$ , isto é,

$$|01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

No entanto, o estado

$$\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

é definido como um estado emaranhado, pois não pode ser descrito como produto tensorial de estados de um q-bit.

### 3.1.1.3 Portas quânticas

Uma porta quântica aplica uma operação unitária  $U$  sobre um q-bit no estado  $|\psi\rangle$  fazendo-o evoluir para o estado  $U|\psi\rangle$ , o qual mantém a interpretação de probabilidade definida na Equação 10.

Existem vários tipos de portas quânticas, como, por exemplo: a porta NOT, a porta NOT-Controlada ou porta CNOT, a porta *Hadamard* ou porta H, etc. A porta NOT é descrita pela matriz

$$U_{NOT} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

e sua operação é mostrada a seguir:

$$U_{NOT} |0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} |0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \quad (14)$$

$$U_{NOT} |1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} |1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = |0\rangle. \quad (15)$$

A porta CNOT define uma operação sobre dois q-bits, chamados q-bit de *controle* e q-bit *alvo*,  $i$  e  $j$  respectivamente. Apenas quando o q-bit de controle está no estado  $|1\rangle$  é que o estado do q-bit alvo é alterado. Para um sistema de dois q-bits, a porta CNOT é descrita pela matriz

$$U_{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

e sua operação, representada de forma esquemática, é mostrada a seguir:

$$|ij\rangle \longrightarrow |i\ i \oplus j\rangle,$$

onde  $i, j \in \{0, 1\}$  e  $\oplus$  é a operação OR-exclusivo. A operação da porta CNOT é mostrada a seguir:

$$\begin{aligned} U_{CNOT} |00\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} |00\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = |00\rangle \\ U_{CNOT} |01\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} |01\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} = |01\rangle \\ U_{CNOT} |10\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} |10\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = |11\rangle \\ U_{CNOT} |11\rangle &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} |11\rangle = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = |10\rangle. \end{aligned} \quad (16)$$

A porta  $H$  é descrita pela matriz

$$U_H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

e sua operação é mostrada a seguir:

$$U_H |0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \quad (17)$$

$$U_H |1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}. \quad (18)$$

Os resultados da Equação 17 e Equação 18 representam uma superposição dos estados  $|0\rangle$  e  $|1\rangle$ , onde a probabilidade de se obter um dos estados, ao se fazer uma medição do estado  $U_H |0\rangle$  ou  $U_H |1\rangle$ , é a mesma: 50%.

### 3.1.2 Simuladores de Computadores Quânticos

Enquanto não é possível a fabricação de computadores quânticos práticos, estão sendo desenvolvidos vários simuladores de computadores quânticos em máquinas convencionais. O objetivo principal destes simuladores é obter uma ideia das dificuldades a serem enfrentadas no desenvolvimento de novos algoritmos destinados a computadores quânticos. Portanto, antes mesmo que o primeiro computador quântico prático esteja disponível, já tem-se diversos algoritmos específicos para computadores quânticos rodando em simulação. Além disso, através de simulações, tem-se uma ideia aproximada do que pode-se esperar ao construir computadores quânticos. O QCS (*Quantum Computing Simulator*) é um simulador de um computador quântico desenvolvido pela *Senko Corporation*, o qual se apresenta como um programa de computador comercial.

## 3.2 Algoritmos Evolucionários Inspirados na Computação Quântica

Os algoritmos evolucionários inspirados na computação quântica são algoritmos que utilizam os princípios da computação quântica para melhorar o desempenho dos algoritmos evolucionários. Estes algoritmos são caracterizados basicamente por uma população inicial composta de indivíduos que representam as possíveis soluções para um determinado problema, uma função capaz de avaliar a aptidão de cada um dos indivíduos e operadores quânticos que evoluem a população com o objetivo de se obter uma população final que contenha uma solução boa ou ótima para o problema.

Os algoritmos evolucionários inspirados na computação quântica vêm recebendo uma grande atenção nos últimos anos e já demonstraram a sua superioridade na solução de diversos tipos de problemas quando comparados com os algoritmos evolucionários. É importante ressaltar que apesar destes algoritmos serem inspirados na computação quântica, os mesmos não

são algoritmos quânticos, e sim algoritmos evolucionários desenvolvidos para serem executados em computadores digitais.

### 3.2.1 Representação

As principais características dos algoritmos evolucionários inspirados na computação quântica são a grande capacidade de busca, a rápida convergência, o pequeno tempo computacional e populações com tamanhos reduzidos (ZHANG *et al.*, 2006). Estas características se devem principalmente ao tipo de codificação da solução e aos operadores de atualização utilizados pelo algoritmo.

Na computação evolucionária, a codificação das soluções pelos indivíduos pode ser feita utilizando-se uma série de representações diferentes. Em termos gerais, esta representação pode ser classificada em binária, numérica e simbólica (HINTERDING, 1999). No caso dos algoritmos evolucionários inspirados na computação quântica, a representação é feita de forma probabilística através dos conceitos dos bits quânticos (q-bits), e do indivíduo quântico (q-indivíduo) que é composto de uma cadeia de q-bits.

O q-bit é definido por um par de números complexos  $\alpha$  e  $\beta$ , os quais indicam as amplitudes dos estados ‘0’ e ‘1’,

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

onde

$$|\alpha|^2 + |\beta|^2 = 1.$$

O valor de  $|\alpha|^2$  representa a probabilidade do q-bit estar no estado ‘0’, enquanto o valor de  $|\beta|^2$  representa a probabilidade do q-bit estar no estado ‘1’. Além dos estados ‘0’ e ‘1’, o q-bit também pode estar na superposição linear destes estados.

O q-indivíduo pode ser representado como um conjunto de q-bits definido como

$$\left[ \begin{array}{c|c|c|c|c} \alpha_1 & \alpha_2 & \alpha_3 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \beta_3 & \cdots & \beta_m \end{array} \right], \quad (19)$$

onde  $|\alpha_i|^2 + |\beta_i|^2 = 1$ , para  $i = 1, 2, 3, \dots, m$ .

A vantagem da representação do indivíduo através dos q-bits frente à representação clássica (binária, numérica e simbólica) é a capacidade de se representar a superposição linear dos estados. No caso de um indivíduo com três q-bits ( $m = 3$ ) tem-se, por exemplo:

$$\left[ \begin{array}{c|c|c} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & \sqrt{\frac{2}{3}} & \frac{\sqrt{3}}{2} \end{array} \right], \quad (20)$$

ou representado de forma alternativa, como

$$\begin{aligned} & \frac{1}{2\sqrt{6}} |000\rangle + \frac{1}{2\sqrt{2}} |001\rangle + \frac{1}{2\sqrt{3}} |010\rangle + \frac{1}{2} |011\rangle \\ & \frac{1}{2\sqrt{6}} |100\rangle + \frac{1}{2\sqrt{2}} |101\rangle + \frac{1}{2\sqrt{3}} |110\rangle + \frac{1}{2} |111\rangle. \end{aligned} \quad (21)$$

As frações indicadas na Equação 21 representam as amplitudes e o quadrado destas indicam que as probabilidades de se representar os estados  $|000\rangle$ ,  $|001\rangle$ ,  $|010\rangle$ ,  $|011\rangle$ ,  $|100\rangle$ ,  $|101\rangle$ ,  $|110\rangle$  e  $|111\rangle$  são  $1/24$ ,  $1/8$ ,  $1/12$ ,  $1/4$ ,  $1/24$ ,  $1/8$ ,  $1/12$  e  $1/4$ , respectivamente.

Os algoritmos evolucionários com a representação do indivíduo através de  $q$ -bits apresentam uma população com maior diversidade do que qualquer outra representação, uma vez que eles podem representar a superposição linear dos estados no espaço de busca de forma probabilística (HAN; KIM, 2002) (HAN, 2003) (AKBARZADEH-T; KHORSAND, 2005). Apenas um  $q$ -indivíduo, tal como o indicado na Equação 20, é suficiente para representar oito estados, onde na representação clássica seriam necessários oito indivíduos.

### 3.2.2 Operadores de Variação

A  $q$ -porta é definida como um operador de variação nos algoritmos evolucionários inspirados na computação quântica. Cada operação de atualização de um  $q$ -bit realizada por uma  $q$ -porta deve satisfazer a condição de normalização  $|\alpha'|^2 + |\beta'|^2 = 1$ , onde  $|\alpha'|^2$  e  $|\beta'|^2$  são os valores do  $q$ -bit atualizado.

As amplitudes de probabilidade dos  $q$ -bits de todos os  $q$ -indivíduos são normalmente inicializadas como o valor  $1/\sqrt{2}$ , o que implica na representação de todos os possíveis estados com a mesma probabilidade. À medida que a probabilidade de cada  $q$ -bit se aproxima de 1 ou 0 através das operações da  $q$ -porta, o  $q$ -indivíduo converge para um estado único e a diversidade da população desaparece gradualmente. Através deste mecanismo, os algoritmos evolucionários inspirados na computação quântica podem tratar o balanceamento entre a exploração e exploração (HAN; KIM, 2002).

As portas quânticas definidas na Seção 3.1 podem ser utilizadas como  $q$ -portas, porém a mais comumente utilizada para atualização dos  $q$ -bits é a porta *rotação*  $U(\Delta\theta_i)$ . A porta rotação é definida como

$$U(\Delta\theta_i) = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix}, \quad (22)$$

onde  $\Delta\theta_i$ ,  $i = 1, 2, \dots, m$ , é o ângulo de rotação de cada  $q$ -bit no sentido do estado 0 ou 1 dependendo do sinal de  $\Delta\theta_i$ .

As amplitudes dos estados de cada  $q$ -bit,  $(\alpha_i, \beta_i)$ , são atualizadas conforme segue:

$$\begin{bmatrix} \alpha'_i \\ \beta'_i \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}. \quad (23)$$

A Figura 19 demonstra a representação polar da porta rotação para os q-bits. Os valores do ângulo  $\Delta\theta_i$  são parâmetros da porta rotação e dependem do problema em que o algoritmo está sendo aplicado.

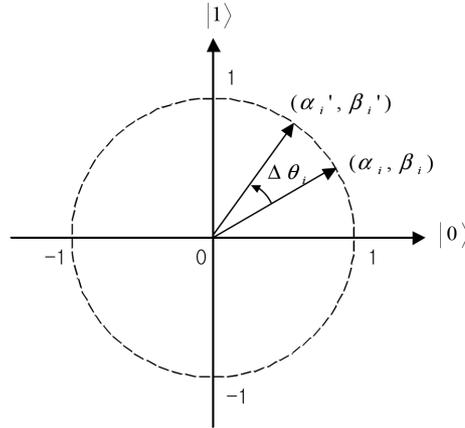


Figura 19: Representação polar da porta rotação

A definição dos valores do ângulo  $\Delta\theta_i$  da porta rotação pode ser realizada através de uma consulta à Tabela 3, conforme proposto por (HAN; KIM, 2002). Han e Kim definem um vetor de ângulos  $\Theta = [\theta_1\theta_2 \cdots \theta_8]$ , onde  $\theta_1\theta_2 \cdots \theta_8$  podem ser selecionados de maneira intuitiva. Por exemplo, se o  $j$ -ésimo bit da solução atual  $x_j$  e o bit correspondente da melhor solução  $b_j$  são 0 e 1, respectivamente, e se a aptidão da solução atual é pior que a da melhor solução, ou seja,  $f(\mathbf{x}) < f(\mathbf{b})$  é verdadeira, então:

1. se o q-bit estiver localizado no primeiro ou terceiro quadrante na Figura 19,  $\theta_3$ , o valor de  $\Delta\theta_i$ , é então definido como um valor positivo para aumentar a probabilidade do estado  $|1\rangle$ ;
2. se o q-bit estiver localizado no segundo ou quarto quadrante,  $-\theta_3$  deve ser usado para aumentar a probabilidade do estado  $|1\rangle$ .

Se  $x_j$  e  $b_j$  são 1 e 0, respectivamente, e se a condição  $f(\mathbf{x}) < f(\mathbf{b})$  é verdadeira, então:

1. se o q-bit estiver localizado no primeiro ou terceiro quadrante,  $\theta_5$  é definido como um valor negativo para aumentar a probabilidade do estado  $|0\rangle$ ;
2. se o q-bit estiver localizado no segundo ou quarto quadrante,  $-\theta_5$  deve ser usado para aumentar a probabilidade do estado  $|0\rangle$ .

Quando ocorre uma ambigüidade na seleção de números negativos ou positivos para os valores dos parâmetros dos ângulos é recomendado definir os valores  $\Delta\theta_i$  como 0 (HAN; KIM, 2002). A magnitude de  $\Delta\theta_i$  tem um efeito na velocidade da convergência, mas se os valores não forem escolhidos de forma adequada o algoritmo pode divergir ou convergir prematuramente para um ótimo local. Os autores recomendam também que os valores da magnitude de  $\Delta\theta_i$  sejam escolhidos dentro de uma faixa que varia de  $0,001\pi$  até  $0,05\pi$  em função do tipo de problema. O sinal de  $\Delta\theta_i$  determina a direção da convergência.

Tabela 3: Definição do ângulo  $\Delta\theta_i$ 

| $x_j$ | $b_j$ | $f(\mathbf{x}) < f(\mathbf{b})$ | $\Delta\theta_i$ |
|-------|-------|---------------------------------|------------------|
| 0     | 0     | verdadeiro                      | $\theta_1$       |
| 0     | 0     | falso                           | $\theta_2$       |
| 0     | 1     | verdadeiro                      | $\theta_3$       |
| 0     | 1     | falso                           | $\theta_4$       |
| 1     | 0     | verdadeiro                      | $\theta_5$       |
| 1     | 0     | falso                           | $\theta_6$       |
| 1     | 1     | verdadeiro                      | $\theta_7$       |
| 1     | 1     | falso                           | $\theta_8$       |

### 3.2.3 Estrutura Básica

Os algoritmos evolucionários inspirados na computação quântica mantêm uma população de  $q$ -indivíduos,  $P(g) = \{\mathbf{p}_1^g, \mathbf{p}_2^g, \dots, \mathbf{p}_n^g\}$  na geração  $g$ , onde  $n$  é o tamanho da população e  $\mathbf{p}_j^g$  é um  $q$ -indivíduo definido na Equação 24, tal que

$$\mathbf{p}_j^g = \left[ \begin{array}{c|c|c|c|c} \alpha_{j1}^t & \alpha_{j2}^t & \alpha_{j3}^t & \cdots & \alpha_{jm}^t \\ \beta_{j1}^t & \beta_{j2}^t & \beta_{j3}^t & \cdots & \beta_{jm}^t \end{array} \right], \quad (24)$$

onde  $m$  é o número de  $q$ -bits, ou seja, o comprimento do  $q$ -indivíduo, e  $j = 1, 2, \dots, n$ .

O processo sequencial simplificado dos algoritmos evolucionários inspirados na computação quântica é mostrado no Algoritmo 1. Esta é a estrutura básica encontrada na maioria dos algoritmos que foram pesquisados, porém algumas variações em torno deste algoritmo são implementadas dependendo do autor. Algumas destas variações são destacadas na Seção 3.2.4.

Primeiramente, a população inicial  $P(0)$  composta de  $n$   $q$ -indivíduos é gerada. Os valores de  $\alpha_j^0$  e  $\beta_j^0$ ,  $j = 1, 2, \dots, m$  de todos os  $q$ -indivíduos  $\mathbf{p}_j^0 = \mathbf{p}_j^g |_{g=0}$ ,  $j = 1, 2, \dots, n$ , são inicializados com  $1/\sqrt{2}$ , fazendo com que cada um dos  $q$ -indivíduos represente a superposição

**Algoritmo 1** Estrutura dos algoritmos evolucionários inspirados na computação quântica

- 
- 1:  $g := 0$ ;
  - 2: **Gere** a população inicial  $P(0)$  com  $n$   $q$ -indivíduos;
  - 3: **Observe**  $P(0)$  em  $S(0)$ ;
  - 4: **Calcule** a aptidão de cada solução em  $S(0)$ ;
  - 5: **Armazene** em  $\mathbf{b}$  a melhor solução em  $S(0)$ ;
  - 6: **Enquanto** (*a condição de término não estiver satisfeita*) **Faça**
  - 7:    $g := g + 1$ ;
  - 8:   **Observe**  $P(g - 1)$  em  $S(g)$ ;
  - 9:   **Calcule** a aptidão de cada solução em  $S(g)$ ;
  - 10:   **Atualize**  $P(g)$  utilizando a  $q$ -porta;
  - 11:   **Armazene** em  $\mathbf{b}$  a melhor solução dentre  $\mathbf{b}$  e  $S(g)$ ;
  - 12: **Fim Enquanto**
- 

linear de todas as soluções possíveis com a mesma probabilidade definida na Equação 25

$$|\psi_{\mathbf{p}_j^0}\rangle = \sum_{k=1}^{2^m} \frac{1}{\sqrt{2}} |X_k\rangle, \quad (25)$$

onde  $X_k$  é o  $k$ -ésimo estado representado pela série binária  $(x_1 x_2 \cdots x_m)$ , com  $x_i, i = 1, 2, \dots, m$ , podendo ser 0 ou 1 com probabilidade  $|\alpha_i^0|^2$  ou  $|\beta_i^0|^2$ , respectivamente.

Em seguida, são obtidas as soluções binárias em  $S(0)$  através da observação dos estados de  $P(0)$ . Seja  $S(0) = \{\mathbf{x}_1^0, \mathbf{x}_2^0, \dots, \mathbf{x}_n^0\}$  na geração  $g = 0$ , cada solução binária  $\mathbf{x}_j^0, j = 1, 2, \dots, n$ , é uma sequência binária de comprimento  $m$ , que é formada através da seleção de 0 ou 1 para cada bit usando a probabilidade,  $|\alpha_i^0|^2$  ou  $|\beta_i^0|^2, i = 1, 2, \dots, m$  de  $\mathbf{p}_j^0$ , respectivamente. Diferentemente de um sistema quântico puro, a observação aqui não destrói a superposição dos estados. Como os algoritmos evolucionários inspirados na computação quântica operam em computadores clássicos e não requerem a presença de um computador quântico, é possível e de interesse que todas as possíveis soluções presentes na superposição estejam disponíveis nas próximas gerações.

O processo de observação pode ser implementado usando probabilidades aleatórias: para cada par de amplitudes  $[\alpha_k, \beta_k]^T (k = 1, 2, \dots, n \times m)$  de cada  $q$ -bit da população  $P_g$ , um número aleatório  $r$  é gerado no intervalo  $[0, 1]$ . Se  $r < |\beta_k|^2$ , então o  $q$ -bit observado é 1; caso contrário, o  $q$ -bit observado é 0.

Cada uma das soluções binárias  $\mathbf{x}_j^0$  em  $S(0)$  é então avaliada para obtenção de uma medida de sua aptidão. A melhor solução em  $S(0)$  é armazenada em  $\mathbf{b}$ .

Em cada iteração, são obtidas novas soluções binárias em  $S(g)$  através da observação de  $P(g-1)$ . Em seguida, cada solução binária é avaliada para obtenção do valor da sua aptidão. Os  $q$ -indivíduos em  $P(g)$  são atualizados utilizando-se portas quânticas. A porta quântica rotação, definida na Seção 3.2, é a mais utilizada como operador de variação.

Caso alguma das soluções em  $S(g)$  seja superior à solução obtida até a geração atual, então a solução armazenada em  $\mathbf{b}$  é substituída por esta nova melhor solução.

Enquanto a condição de término não é alcançada, as etapas mencionadas são repetidas. Normalmente, a condição de término é definida como um número máximo de gerações.

### 3.2.4 Exemplos de Aplicações

Em (NARAYANAN; MOORE, 1996) foi apresentado pela primeira vez um algoritmo que introduziu os conceitos e a teoria da computação quântica nos algoritmos genéticos. Desde então, foram publicados vários trabalhos sobre algoritmos que possuem alguma de suas etapas inspiradas na computação quântica. Dentre estes, pode-se destacar o algoritmo chamado de QEA (*Quantum-inspired Evolutionary Algorithm*) que foi apresentado pelos autores (HAN; KIM, 2002). O processo do QEA é mostrado no Algoritmo 2.

O QEA é muito parecido com o Algoritmo 1. No entanto, para melhorar a eficiência de busca, além de armazenar a melhor solução encontrada até a geração atual  $\mathbf{b}$ , o QEA armazena também a melhor solução encontrada por cada  $q$ -indivíduo até a geração atual, ou seja, as soluções em  $S(0)$  são copiadas em  $B(0)$ , onde  $B(0) = \{\mathbf{b}_1^0, \mathbf{b}_2^0, \dots, \mathbf{b}_n^0\}$ . Neste caso, na etapa de atualização descrita na Seção 3.2.2, a melhor solução encontrada por cada  $q$ -indivíduo até a geração atual é então utilizada na atualização das suas respectivas probabilidades. No caso do Algoritmo 1 apenas a melhor solução encontrada até a geração atual  $\mathbf{b}$  é utilizada na atualização de todos os  $q$ -indivíduos. Para evitar a convergência prematura do algoritmo os autores introduziram duas operações no algoritmo: a *migração global* e a *migração local*. Quando a condição de migração global é satisfeita, então a melhor solução  $\mathbf{b}$  sofre uma migração para  $B(g)$  de forma global, ou seja, todas as soluções em  $B(g)$  são substituídas por  $\mathbf{b}$ . Quando a condição de migração local é satisfeita, então a melhor solução em um grupo local de  $B(g)$  sofre uma migração para as outras soluções em um mesmo grupo local, ou seja, todas as soluções de um mesmo grupo são substituídas pela melhor delas. No QEA, um grupo local é definido como uma sub-população afetada por uma migração local, e o seu tamanho é o número de indivíduos do grupo local.

O QEA foi explorado em diversos trabalhos, sendo aplicado em problemas clássicos de

---

**Algoritmo 2** QEA

---

- 1:  $g := 0$ ;
  - 2: **Gere** a população inicial  $P(0)$  com  $n$   $q$ -indivíduos;
  - 3: **Observe**  $P(0)$  em  $S(0)$ ;
  - 4: **Calcule** a aptidão de cada solução em  $S(0)$ ;
  - 5: **Armazene** em  $B(0)$  as soluções de  $S(0)$ ;
  - 6: **Enquanto** (a condição de término não estiver satisfeita) **Faça**
  - 7:    $g := g + 1$ ;
  - 8:   **Observe**  $P(g - 1)$  em  $S(g)$ ;
  - 9:   **Calcule** a aptidão de cada solução em  $S(g)$ ;
  - 10:   **Atualize**  $P(g)$  utilizando a  $q$ -porta;
  - 11:   **Armazene** em  $B(g)$  as melhores soluções dentre  $B(g - 1)$  e  $S(g)$ ;
  - 12:   **Armazene** em  $\mathbf{b}$  a melhor solução que encontra-se em  $B(g)$ ;
  - 13:   **Se** (a condição de migração global estiver satisfeita) **Então**
  - 14:     **Faça** uma migração global de  $\mathbf{b}$  para  $B(g)$
  - 15:   **Senão** (a condição de migração local estiver satisfeita) **Então**
  - 16:     **Faça** uma migração local de  $\mathbf{b}_j^g$  em  $B(g)$  para  $B(g)$
  - 17:   **Fim Se**
  - 18: **Fim Enquanto**
- 

otimização combinacional e numérica do tipo  $NP$ -completo (HAN; KIM, 2002) (HAN, 2003) (HAN; KIM, 2004) (HAN; KIM, 2006) e também em problemas do mundo real, tais como, um método eficiente para alocação de espaço em disco rígido (HAN; KIM *et al.*, 2003) e um novo sistema de detecção de face (JANG; HAN; KIM, 2004).

Além do QEA, alguns algoritmos desenvolvidos por outros autores também podem ser destacados. (PLATEL; SCHLIEBS; KASABOV, 2007) apresentaram o algoritmo chamado de QEA versátil ou vQEA. Neste algoritmo, as soluções armazenadas em  $B(g)$  são chamadas pelos autores de *atratores*. No vQEA, os atratores são substituídos a cada geração sem levar em consideração as suas aptidões. No QEA, cada  $q$ -indivíduo explora uma dada região do espaço de busca. Caso uma boa solução seja encontrada nesta região, a mesma é então escolhida como um atrator e a exploração irá se concentrar nesta nova região. De modo geral, existem duas maneiras para que um atrator  $\mathbf{b}_j^g$  seja atualizado: quando uma solução melhor  $\mathbf{x}_j^g$  é encontrada ou quando a migração é aplicada. Quando um novo atrator  $\mathbf{b}_j^g$  é escolhido no espaço de busca,

o  $q$ -indivíduo correspondente será suavemente movido na direção deste ponto até que uma solução melhor  $x_j^g$  seja encontrada. Quando nenhuma solução melhor é encontrada, durante este movimento, o QEA fica preso e converge prematuramente. A única oportunidade para um  $q$ -indivíduo escapar deste atrator é que a etapa de migração o substitua por um atrator melhor que tenha sido produzido em outro grupo local. Caso contrário, é possível que a escolha deste atrator, que não é o ótimo, seja irreversível.

No vQEA, com o objetivo de se evitar o caso de uma escolha irreversível, a estratégia para atualização dos atratores é modificada em relação ao QEA. No QEA, o processo de atualização dos atratores, etapa na qual são selecionadas as melhores soluções entre  $B(g - 1)$  e  $S(g)$ , é aplicado o princípio do elitismo, uma vez que os atratores em  $B(g - 1)$  só são substituídos se as soluções em  $S(g)$  forem melhores. No vQEA, esta etapa é simplesmente desabilitada. Portanto, os atratores são substituídos em todas as gerações sem considerar as suas aptidões o que faz com que estes apresentem uma volatilidade muito grande. Mais ainda, para garantir a convergência do vQEA, a migração global é realizada em todas as gerações, de tal forma que todos os atratores sejam idênticos e que estes correspondam na geração  $g + 1$  à melhor solução encontrada na geração  $g$ . Com estas alterações, pode-se notar que o tamanho do grupo local e a migração local definidos no QEA não afetam o algoritmo.

Com o vQEA a informação sobre o espaço de busca coletada ao longo do processo evolucionário é continuamente atualizada e compartilhada por todos os  $q$ -indivíduos da população. O processo seqüencial simplificado do vQEA é mostrado no Algoritmo 3.

O vQEA utiliza a porta de rotação como operador de atualização dos  $q$ -indivíduos e o número máximo de gerações como critério de parada. O vQEA foi aplicado na solução do Problema da Mochila (do inglês, *Knapsack Problem*) (GAREY; JOHNSON, 1979) e obteve resultados melhores que o QEA (PLATEL; SCHLIEBS; KASABOV, 2007).

O NQGA (*Novel Quantum Genetic Algorithm*) (ZHANG *et al.*, 2003) também é muito semelhante ao Algoritmo 1. A principal diferença diz respeito ao acréscimo das etapas chamadas pelos autores de *imigração* e *catástrofe*. Estas operações, assim como as etapas de migração local e global do QEA, têm o objetivo de ajudar o algoritmo a escapar das soluções locais. A cada número pré-determinado de gerações a operação de imigração é realizada. Nesta operação, as amplitudes de probabilidade dos  $q$ -bits são substituídas por novos valores escolhidos de forma aleatória ou por  $q$ -bits da melhor solução  $\mathbf{b}$ . Já na operação de catástrofe, caso a melhor solução armazenada em  $\mathbf{b}$  permaneça sem ser atualizada por um número pré-determinado de gerações, então  $\mathbf{b}$  é substituída pela melhor solução da geração atual. O NQGA utiliza a porta

---

**Algoritmo 3** vQEA

---

- 1:  $g := 0$ ;
  - 2: **Gere** a população inicial  $P(0)$  com  $n$   $q$ -indivíduos;
  - 3: **Observe**  $P(0)$  em  $S(0)$ ;
  - 4: **Calcule** a aptidão de cada solução em  $S(0)$ ;
  - 5: **Armazene** em  $B(0)$  as soluções em  $S(0)$ ;
  - 6: **Enquanto** (*a condição de término não estiver satisfeita*) **Faça**
  - 7:    $g := g + 1$ ;
  - 8:   **Observe**  $P(g - 1)$  em  $S(g)$ ;
  - 9:   **Calcule** a aptidão de cada solução em  $S(g)$ ;
  - 10:   **Atualize**  $P(g)$  em função de  $B(g - 1)$  e  $S(g)$  utilizando a  $q$ -porta;
  - 11:   **Armazene** em  $\mathbf{b}$  a melhor solução que encontra-se em  $S(g)$ ;
  - 12:   **Faça** a migração global de  $\mathbf{b}$  para  $B(g)$
  - 13: **Fim Enquanto**
- 

rotação como operador de atualização dos  $q$ -indivíduos e o número máximo de gerações como critério de parada. O NQGA foi aplicado na seleção do melhor subconjunto de características dentre um enorme número de características envolvidas no reconhecimento de sinais emitidos por radares (ZHANG *et al.*, 2003). O processo seqüencial simplificado do NQGA é mostrado no Algoritmo 4.

O algoritmo multiobjetivo chamado de MOQEA (*Multiobjective Quantum Inspired Evolutionary Algorithm*) foi apresentado em (TALBI; BATOUCHE; DRAA, 2007). O MOQEA utiliza os resultados obtidos pelo algoritmo *K-means* (PAPPAS, 1992) como entrada para estabelecer um conjunto de soluções não-dominadas em problemas de otimização multiobjetivo de segmentação de imagens. As soluções obtidas pelo MOQEA foram superiores às soluções obtidas por um algoritmo genético multiobjetivo.

O algoritmo apresentado em (CRUZ; VELLASCO; PACHECO, 2006) e (CRUZ; VELLASCO; PACHECO, 2008), chamado de QIEA<sub>R</sub> (*Quantum-Inspired Evolutionary Algorithm*), diferentemente dos algoritmos citados anteriormente, utiliza a codificação real ao invés da binária. Apesar de não fazer uso dos  $q$ -bits na representação das soluções, o algoritmo faz observações periódicas de uma distribuição que pode ser considerada como um estado quântico que entra em colapso para um estado clássico após uma observação. O algoritmo foi aplicado com sucesso na otimização de funções numéricas selecionadas de (TU; LU, 2004).

---

**Algoritmo 4** NQGA

---

- 1:  $g := 0$ ;
  - 2: **Gere** a população inicial  $P(0)$  com  $n$   $q$ -indivíduos;
  - 3: **Observe**  $P(0)$  em  $S(0)$ ;
  - 4: **Calcule** a aptidão de cada solução em  $S(0)$ ;
  - 5: **Armazene** em  $\mathbf{b}$  a melhor solução em  $S(0)$ ;
  - 6: **Enquanto** (a condição de término não estiver satisfeita) **Faça**
  - 7:    $g := g + 1$ ;
  - 8:   **Observe**  $P(g - 1)$  em  $S(g)$ ;
  - 9:   **Calcule** a aptidão de cada solução em  $S(g)$ ;
  - 10:   **Atualize**  $P(g)$  em função de  $\mathbf{b}$  e  $S(g)$  utilizando a  $q$ -porta;
  - 11:   **Armazene** em  $\mathbf{b}$  a melhor solução entre  $\mathbf{b}$  e  $S(g)$ ;
  - 12:   **Se** (a condição de imigração estiver satisfeita) **Então**
  - 13:     **Altere** as amplitudes de probabilidade de alguns  $q$ -indivíduos;
  - 14:   **Fim Se**
  - 15:   **Se** (a condição de catástrofe estiver satisfeita) **Então**
  - 16:     **Substitua** a melhor solução em  $\mathbf{b}$ ;
  - 17:   **Fim Se**
  - 18: **Fim Enquanto**
- 

(ALFARES; ESAT, 2007) também desenvolveram um algoritmo que trabalha com a codificação real das variáveis. O algoritmo chamado de RQIEA (*Real-Coded Quantum Inspired Evolution Algorithm*) foi aplicado na otimização de projetos de vasos de pressão e de projetos de tensão/compressão de molas, obtendo resultados superiores aos obtidos por outros métodos bem conhecidos, como, por exemplo, o algoritmo genético.

Em (COELHO; NEDJAH; MOURELLE, 2008) os autores apresentam uma abordagem inspirada na computação quântica para o método de otimização de enxame de partículas usando a distribuição gaussiana. O método chamado de G-QPSO (*Gaussian Quantum-Behaved Particle Swarm Optimization*) foi utilizado com sucesso na sintonia dos parâmetros de um sistema de controle baseado na lógica *fuzzy*.

O algoritmo chamado de RQEA (*Real-parameter quantum evolutionary algorithm*) foi apresentado em (BABU; DAS; PATVARDHAN, 2008). O RQEA, diferentemente dos algoritmos descritos anteriormente, não utiliza o processo de observação para obter as possíveis soluções

---

do problema no qual está sendo aplicado. As possíveis soluções, compostas de parâmetros reais, são obtidas através de operadores evolucionários quânticos especiais. Os resultados obtidos na aplicação do RQEA no problema de despacho de cargas em sistemas elétricos foram superiores aos obtidos por outros métodos em termos da qualidade das soluções e da consistência na obtenção do mínimo global.

### 3.3 Resumo do Capítulo

Neste capítulo foram apresentados de forma resumida alguns dos conceitos que envolvem a computação quântica. Também foi apresentada a estrutura básica dos algoritmos evolucionários inspirados na computação quântica e algumas variações em torno desta estrutura básica. No próximo capítulo será investigada a aplicação destes algoritmos no projeto de circuitos digitais. O algoritmo será aplicado na busca de uma atribuição de estados otimizada, a qual representa uma das etapas importantes no projeto de circuitos seqüenciais síncronos, conforme mencionado no Capítulo 1.

## Capítulo 4

# ATRIBUIÇÃO DE ESTADOS DE SISTEMAS SEQUENCIAIS

**M**ÁQUINAS de estados finitos síncronas são importantes para um projeto de sistemas digitais sequenciais. Entre outros aspectos importantes, representam uma maneira poderosa de sincronização de componentes eletrônicos de tal forma que estes possam cooperar adequadamente no cumprimento dos objetivos finais do projeto do circuito. Neste capítulo é proposta uma metodologia evolucionária baseada nos algoritmos evolucionários inspirados na computação quântica para resolver um dos problemas relacionados ao projeto de máquinas de estados finitos. O problema *NP*-difícil da atribuição de estados é abordado utilizando-se um algoritmo evolucionário inspirado na computação quântica. A motivação advém do fato de que com uma boa atribuição de estados, a máquina de estados pode ser fisicamente implementada utilizando-se um circuito otimizado, o qual irá consumir uma área mínima e/ou terá um tempo de resposta mínimo.

Na Seção 4.1, o problema de atribuição de estados é descrito, sendo apresentado um exemplo de como a atribuição de estados pode impactar na complexidade da lógica de controle da máquina de estados. Ainda nesta Seção, algumas técnicas utilizadas para resolver o problema de atribuição de estados são apresentadas. Na Seção 4.2, um algoritmo evolucionário inspirado na computação quântica é aplicado na solução do problema de atribuição de estados. Na Seção 4.3 os resultados são comparados com aqueles obtidos por outros métodos que foram aplicados na solução do mesmo problema. Em seguida, na Seção 4.3.2, o mérito dos resultados obtidos é discutido.

### 4.1 O Problema de Atribuição de Estados

A síntese de um circuito digital envolve a transformação da especificação comportamental do circuito em uma arquitetura de componentes eletrônicos que possa ser implementada utilizando-

se, por exemplo, um processo de fabricação de circuitos integrados VLSI (*Very Large Scale Integration*) personalizados ou configuração de dispositivos semi-personalizados, tais como as FPGAs (*Field-Programmable Gate Array*). Em geral, os projetistas ou as ferramentas ECAD (*Electronic Computer-Aided Design*) devem atender algumas restrições de projeto relacionadas com a otimização de parâmetros e/ou a limitação de recursos disponíveis. Os parâmetros que normalmente busca-se otimizar durante o projeto dos circuitos digitais são: a área ocupada, o tempo de resposta e a dissipação de energia dos circuitos (ZEBULUM, 1999), (ALI, 2003).

Uma vez que as etapas de especificação e redução de estados descritas no Capítulo 1 tenham sido concluídas, o passo seguinte no projeto das máquinas de estados consiste da atribuição de um código binário para cada um dos seus estados simbólicos. Estes códigos são utilizados para criar uma implementação da lógica de controle da máquina de estados. Os códigos binários correspondentes aos estados internos da máquina ficam armazenado em uma memória formada por um conjunto de *flip-flops*, sendo que o *flip-flops* tipo D é o mais comumente utilizado nestas implementações. A codificação escolhida tem um efeito direto na complexidade e na estrutura da lógica de controle da máquina de estados, conforme será demonstrado na Seção 4.1. O problema de atribuição de estados vem sendo estudado desde o início dos anos 60 (ARMSTRONG, 1962), e representa um dos problemas importantes no projeto automático de sistemas seqüenciais.

A Figura 1 do Capítulo 1 mostra o diagrama em blocos de uma máquina de estados. Para a obtenção do circuito lógico responsável pela ativação dos *flip-flops* na seqüência desejada deve ser atribuído um código binário para cada estado. A única restrição para que uma codificação de estados seja válida é que para cada estado deve ser atribuído um valor binário distinto.

Seja  $n$  o número de estados em uma máquina de estados finitos. O número mínimo de sinais de estado é definido por  $b = \lceil \log_2 n \rceil$ . Uma atribuição que utilize um menor número de sinais de estado tem a vantagem de utilizar um número menor de dispositivos de memória. O processo de atribuição consiste na decisão de qual dos  $2^b$  códigos fornecidos pelos  $b$  sinais de estado deve ser atribuído para cada estado particular da máquina de estados. Considerando o número mínimo de sinais de estado, a quantidade de possíveis codificações (HARTMANIS, 1961) é dada por:

$$N(n, b) = \frac{2^b!}{(2^b - n)!} \quad (26)$$

A Tabela 4 mostra os valores de  $N(n, b)$  para vários valores de  $n$  e  $b$ . Como pode ser visto, a quantidade de atribuições de estado possíveis para uma máquina com 8 estados é igual

a 40320. Para uma máquina com 9 ou mais estados, este valor se torna extremamente elevado. Uma vez que o número de possíveis atribuições de estados cresce bastante com o número de estados, é praticamente impossível testar todas as atribuições com o intuito de selecionar a que proporciona o circuito lógico mais simples. Por exemplo, se a avaliação de cada atribuição de estados consumir um tempo de  $100 \mu s$ , então levaria 66 anos para testar todas as possibilidades de uma máquina de 16 estados. Por isso, a avaliação exaustiva não é apropriada para máquinas de estados que não sejam muito pequenas (até 4 estados). Portanto, o uso de heurísticas se faz necessário para lidar com o problema de atribuição de estados. Estas heurísticas, encapsuladas em uma função objetivo, tentam otimizar os objetivos do problema definidos pelos usuários. A qualidade da solução depende, portanto, da fidelidade e precisão da modelagem do problema pela função objetivo.

Tabela 4: Quantidade de atribuições de estados diferentes

| $n$ | $b$ | $N(n, b)$                 |
|-----|-----|---------------------------|
| 2   | 1   | 2                         |
| 3   | 2   | 24                        |
| 4   | 2   | 24                        |
| 5   | 3   | 6720                      |
| 6   | 3   | 20160                     |
| 7   | 3   | 40320                     |
| 8   | 3   | 40320                     |
| 9   | 4   | $\approx 4 \cdot 10^9$    |
| 10  | 4   | $\approx 3 \cdot 10^{10}$ |
| 11  | 4   | $\approx 2 \cdot 10^{11}$ |
| 12  | 4   | $\approx 9 \cdot 10^{11}$ |
| 13  | 4   | $\approx 3 \cdot 10^{12}$ |
| 14  | 4   | $\approx 1 \cdot 10^{13}$ |
| 15  | 4   | $\approx 2 \cdot 10^{13}$ |
| 16  | 4   | $\approx 2 \cdot 10^{13}$ |
| 17  | 5   | $\approx 2 \cdot 10^{23}$ |

#### 4.1.1 Impacto de Diferentes Atribuições de Estados

A lógica de controle de uma máquina de estados é responsável por gerar os sinais da saída primária, assim como os sinais que permitem definir o próximo estado. Isto é feito utilizando-se os sinais da entrada primária e os sinais de estado. Tradicionalmente, o circuito combinacional da lógica de controle é obtido utilizando-se os *mapas de transição* dos *flip-flops* (RHYNE, 1973). Para demonstrar o impacto da atribuição de estados na complexidade da lógica de controle, pode-se considerar o exemplo de uma máquina de estados com quatro estados designados

simbolicamente por  $s_0, s_1, s_2$  e  $s_3$ , um sinal de entrada  $I$  e um sinal de saída  $O$ . A Tabela 5 indica as transições de estado e duas codificações válidas para esta máquina, identificadas por *Atribuição<sub>1</sub>* e *Atribuição<sub>2</sub>*. Como existem quatro estados, dois *flip-flops*,  $W$  e  $X$ , são suficientes para o armazenamento dos sinais de estado. A tabela de transição correspondente à *Atribuição<sub>1</sub>* é mostrada na Tabela 6. O método tradicional de síntese da lógica de controle de máquinas de estados é descrito na Seção 1.3. Note que o exemplo usado nesta ilustração é o mesmo que o apresentado naquela seção.

Tabela 5: Tabela de estados com duas atribuições de estados possíveis

| Estado Atual | Próximo Estado |         | Saída ( $O$ ) |         | <i>Atribuição<sub>1</sub></i> | <i>Atribuição<sub>2</sub></i> |
|--------------|----------------|---------|---------------|---------|-------------------------------|-------------------------------|
|              | $I = 0$        | $I = 1$ | $I = 0$       | $I = 1$ |                               |                               |
| $s_0$        | $s_0$          | $s_1$   | 0             | 0       | 00                            | 00                            |
| $s_1$        | $s_2$          | $s_1$   | 0             | 1       | 11                            | 01                            |
| $s_2$        | $s_0$          | $s_3$   | 1             | 0       | 01                            | 11                            |
| $s_3$        | $s_2$          | $s_1$   | 1             | 1       | 10                            | 10                            |

Tabela 6: Tabela de transição de estados para *Atribuição<sub>1</sub>*

| Entrada $I$ | Estado Atual |     | Próximo Estado |       | Saída $O$ | Comportamento |          |
|-------------|--------------|-----|----------------|-------|-----------|---------------|----------|
|             | $W$          | $X$ | $W^+$          | $X^+$ |           | $W$           | $X$      |
| 0           | 0            | 0   | 0              | 0     | 0         | 0             | 0        |
| 0           | 0            | 1   | 0              | 0     | 1         | 0             | $\beta$  |
| 0           | 1            | 0   | 0              | 1     | 1         | $\beta$       | $\alpha$ |
| 0           | 1            | 1   | 0              | 1     | 0         | $\beta$       | 1        |
| 1           | 0            | 0   | 1              | 1     | 0         | $\alpha$      | $\alpha$ |
| 1           | 0            | 1   | 1              | 0     | 0         | $\alpha$      | $\beta$  |
| 1           | 1            | 0   | 1              | 1     | 1         | 1             | $\alpha$ |
| 1           | 1            | 1   | 1              | 1     | 1         | 1             | 1        |

Os mapas de transição dos *flip-flops*  $W$  e  $X$  são mostrados na Figura 20. Utilizando *flip-flops* tipo D, as equações de excitação dos *flip-flops* se tornam:

$$\begin{aligned} D_W &= I \\ D_X &= W + I. \end{aligned} \quad (27)$$

O mapa de *Karnaugh* para a função de saída,  $O$ , é mostrada na Figura 21, resultando na seguinte equação de controle do sinal de saída:

$$O = I \cdot W + W \cdot \bar{X} + \bar{I} \cdot \bar{W} \cdot X. \quad (28)$$

O esquemático do circuito da máquina de estados utilizando *Atribuição<sub>1</sub>* é mostrado na Figura 22.

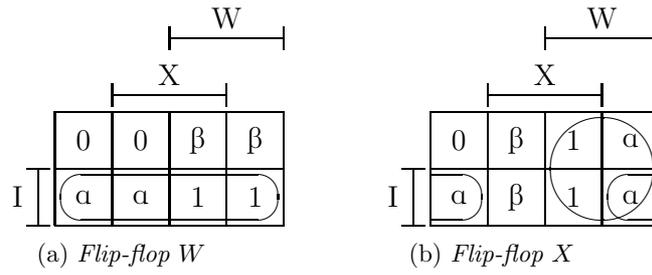


Figura 20: Mapas de transição dos *flip-flops*  $W$  e  $X$  para a *Atribuição* $_1$

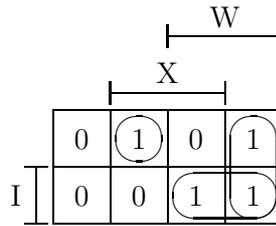


Figura 21: Mapa de *Karnaugh* da função de saída para a *Atribuição* $_1$

A tabela de transição correspondente à *Atribuição* $_2$  é mostrada na Tabela 7. Os mapas de transição dos *flip-flops* são mostrados na Figura 23. Utilizando *flip-flops* tipo D, as equações de excitação dos *flip-flops* se tornam:

Tabela 7: Tabela de transição de estados para *Atribuição* $_2$

| Entrada<br>$I$ | Estado Atual |     | Próximo Estado |       | Saída | Comportamento |          |
|----------------|--------------|-----|----------------|-------|-------|---------------|----------|
|                | $W$          | $X$ | $W^+$          | $X^+$ | $O$   | $W$           | $X$      |
| 0              | 0            | 0   | 0              | 0     | 0     | 0             | 0        |
| 0              | 0            | 1   | 1              | 1     | 0     | $\alpha$      | 1        |
| 0              | 1            | 0   | 1              | 1     | 1     | 1             | $\alpha$ |
| 0              | 1            | 1   | 0              | 0     | 1     | $\beta$       | $\beta$  |
| 1              | 0            | 0   | 0              | 1     | 0     | 0             | $\alpha$ |
| 1              | 0            | 1   | 0              | 1     | 1     | 0             | 1        |
| 1              | 1            | 0   | 0              | 1     | 1     | $\beta$       | $\alpha$ |
| 1              | 1            | 1   | 1              | 0     | 0     | 1             | $\beta$  |

$$\begin{aligned}
 D_W &= \bar{I} \cdot \bar{W} \cdot X + \bar{I} \cdot W \cdot \bar{X} + I \cdot W \cdot X \\
 D_X &= I \cdot \bar{W} + \bar{W} \cdot X + I \cdot X + \bar{I} \cdot W \cdot \bar{X}.
 \end{aligned}
 \tag{29}$$

O mapa de *Karnaugh* para a função de saída,  $O$ , é mostrada na Figura 24, resultando na seguinte equação de controle do sinal de saída:

$$O = I \cdot \bar{W} \cdot X + \bar{I} \cdot W + W \cdot \bar{X}.
 \tag{30}$$

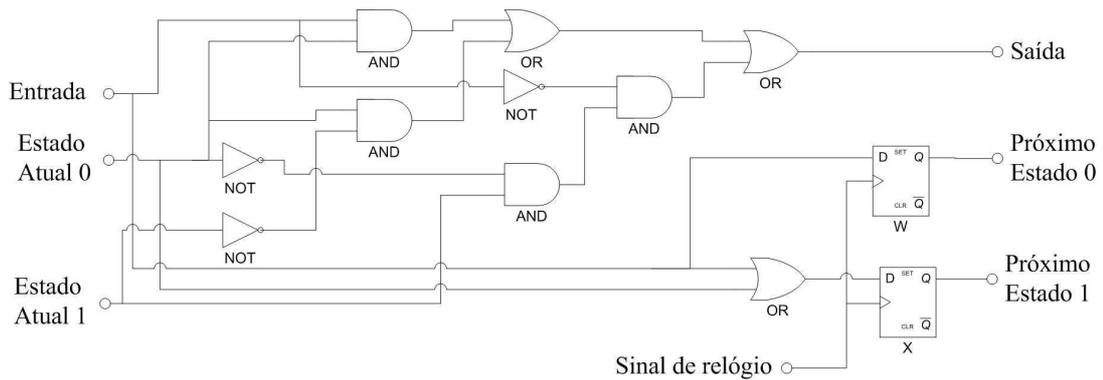


Figura 22: Esquemático da máquina de estados utilizando *Atribuição<sub>1</sub>*

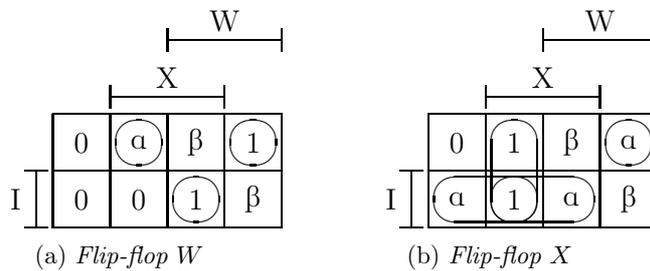
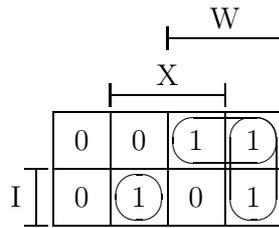
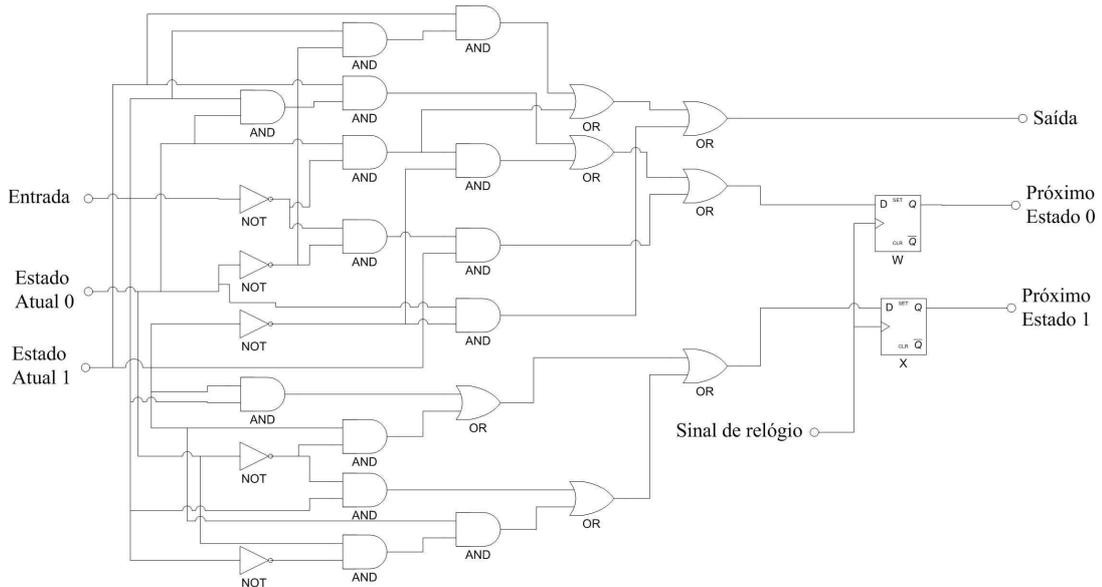


Figura 23: Mapas de transições dos *flip-flops W* e *X* para *Atribuição<sub>2</sub>*

O esquemático do circuito da máquina de estados utilizando *Atribuição<sub>2</sub>* é mostrado na Figura 25.

Este exemplo ilustra que a escolha de uma atribuição de estados apropriada pode reduzir o custo de implementação do circuito. Este custo é definido aqui como a quantidade de portas de NOT e portas AND e OR de duas entradas, necessárias para a realização do circuito. O sinal negado das saídas dos *flip-flops*,  $\overline{W}$  e  $\overline{X}$ , são considerados como sendo de custo zero para a implementação dos circuitos já que estão disponíveis como saídas dos mesmos. A Tabela 8 resume, para várias atribuições, incluindo *Atribuição<sub>1</sub>* e *Atribuição<sub>2</sub>*, a quantidade de portas necessárias para a implementação da lógica de controle dos *flip-flops W* e *X*, assim como do circuito responsável pela geração do sinal de saída *O*.

É importante observar que qualquer atribuição que mantenha a distinção entre os vários estados garante uma implementação correta da máquina de estados finitos, porém certas atribuições permitem uma realização mais econômica que outras (ARMSTRONG, 1962). Vale também ressaltar que para uma máquina mais complexa (isto é, com mais estados e mais transições), é esperado que o ganho seja mais significativo ainda.

Figura 24: Mapa de *Karnaugh* da função de saída para *Atribuição<sub>2</sub>*Figura 25: Esquemático da máquina de estados utilizando *Atribuição<sub>2</sub>*

### 4.1.2 Técnicas para Atribuição de Estados

Dada uma função de transição de estados, os requisitos de área e tempo de resposta variam para diferentes atribuições de estado. Portanto, o projetista ou a ferramenta ECAD (*Electronic Computer-Aided Design*) para síntese de circuitos deve sempre procurar a atribuição que minimize a complexidade e, por conseguinte, o custo da lógica combinacional necessária para controlar as transições de estado e as saídas. A realização de máquinas de estados com um custo mínimo tem motivado vários pesquisadores na busca por atribuições de estados otimizadas. Através deste estudo surgiram várias técnicas para resolver este problema. Pode-se destacar três destas técnicas, além das mais óbvias que são a atribuição *aleatória* que simplesmente atribui um código aleatório para cada um dos estados e a *seqüencial* que atribui um código binário para cada um dos estados simbólicos da máquina, seguindo a ordem em que os mesmos aparecem na tabela de estados.

- *One-hot*: Essa técnica de codificação associa um bit para cada estado (ERCEGOVAC;

Tabela 8: Comparação do número de portas necessárias para várias atribuições, inclusive *Atribuição<sub>1</sub>* e *Atribuição<sub>2</sub>*

| Atribuição       | #AND | #OR | #NOT | Total |
|------------------|------|-----|------|-------|
| [00, 11, 01, 10] | 4    | 3   | 1    | 8     |
| [00, 01, 10, 11] | 5    | 2   | 1    | 8     |
| [00, 10, 01, 11] | 5    | 2   | 1    | 8     |
| [00, 11, 10, 01] | 5    | 3   | 1    | 9     |
| [11, 00, 01, 10] | 5    | 3   | 1    | 9     |
| [00, 01, 11, 10] | 10   | 7   | 1    | 18    |
| [00, 10, 11, 01] | 11   | 6   | 1    | 18    |

LANG; MORENO, 1999). Está técnica tem a vantagem de necessitar de lógicas de controle relativamente simples para projetar, porém normalmente utiliza mais *flip-flops* que o mínimo necessário. Por exemplo, uma máquina com 16 estados requer 16 *flip-flops* para a codificação *one-hot*, enquanto apenas 4 seriam suficientes para codificar distintivamente todos os estados da máquina.

- Baseada em heurísticas: Essas técnicas buscam uma atribuição de estados otimizada usando heurísticas. (ARMSTRONG, 1962) e (HUMPHREY, 1958) apresentam uma heurística baseada na adjacência, ou seja, os estados que possuem os mesmos próximos estados ou que são os próximos estados de um mesmo estado recebem códigos adjacentes. O termo *códigos adjacentes* significa que estes diferem em uma única posição. Esta heurística será discutida com mais detalhes na Seção 4.2. Os algoritmos MUSTANG (DEVADAS *et al.*, 1988) e MUSE (RUDELL; SANGIOVANNI-VINCENTELLI, 1987) elaboram um grafo onde os vértices correspondem a cada estado da máquina e as arestas são utilizadas para conectar cada par de estados. Cada aresta tem um peso que mede o quão desejável seria que o par de estados recebesse códigos adjacentes. O problema então se reduz a uma minimização da soma dos pesos dos estados do grafo. Os sistemas KISS<sup>TM</sup> (MICHELI; BRAYTON; SANGIOVANNI-VINCENTELLI, 1984) e NOVA<sup>TM</sup> (VILLA; SANGIOVANNI-VINCENTELLI, 1990) são baseados na minimização de funções Booleanas para agrupar estados e então elaborar atribuições através do encaixe destes grupos nas faces de um hipercubo. Estes sistemas são amplamente utilizados na construção de sistemas comerciais.
- Baseada em algoritmos evolucionários: Recentemente, os algoritmos genéticos introduzidos no Capítulo 2 passaram a ser utilizados na solução do problema de atribuição de estados. Em (ALI, 2003) e (AMARAL; TUMER; GLOSH, 1995), os resultados alcançados

foram superiores aos obtidos utilizando-se o programa NOVA™.

Na Seção 4.2 será apresentado um Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ) projetado para encontrar uma atribuição de estados otimizada, a qual permita a utilização de uma lógica de controle da máquina de estados com a menor complexidade possível.

## 4.2 AEICQ Aplicado ao Problema de Atribuição de Estados

Nesta seção será apresentada uma nova técnica para a solução do problema de atribuição de estados em máquinas de estados síncronas. Esta técnica é baseada na utilização de um Algoritmo Evolucionário Inspirado na computação Quântica (AEICQ) em conjunto com as heurísticas apresentadas em (ARMSTRONG, 1962) e (HUMPHREY, 1958).

### 4.2.1 Estrutura do AEICQ

O AEICQ segue a estrutura básica dos algoritmos evolucionários inspirados na computação quântica, apresentada no Algoritmo 1 do Capítulo 3. O processo seqüencial simplificado do AEICQ é mostrado no Algoritmo 5.

A principal diferença do AEICQ frente ao Algoritmo 1 do Capítulo 3 consiste na inclusão da etapa de *restrição das probabilidades* e a etapa de *migração global*. Estas duas etapas foram introduzidas com o objetivo de permitir ao algoritmo de escapar de soluções locais. A etapa restrição de probabilidade é realizada após a operação de atualização dos q-bits. Esta etapa evita que ocorra a convergência prematura dos q-bits para os estados 0 ou 1, não permitindo que os valores de  $\alpha$  e  $\beta$  sejam menores que  $\sqrt{0,02}$  e maiores que  $\sqrt{0,98}$  durante o processo evolucionário. Desta forma, as probabilidades  $|\alpha|^2$  e  $|\beta|^2$  são limitadas em 2% no mínimo e 98% no máximo. A etapa de migração global realiza a mesma operação adotada no QEA (HAN; KIM, 2002) e que foi descrita na Seção 3.2. Porém, a condição que determina a realização da operação de migração é distinta nos dois algoritmos. No caso do AEICQ, a substituição de todas as soluções em  $B(g)$  por  $b$  ocorre apenas quando a melhor solução  $b$  permanece inalterada por  $G$  gerações, enquanto no QEA esta substituição ocorre em gerações pré-determinadas. A migração global pode induzir a uma variação das amplitudes de probabilidade dos q-bits de um indivíduo, aumentando assim a diversidade da população.

---

**Algoritmo 5** AEICQ - Algoritmo Evolucionário Inspirado na Computação Quântica

---

- 1:  $g := 0$ ;
  - 2: **Gere** a população inicial  $P(0)$  com  $n$   $q$ -indivíduos;
  - 3: **Observe**  $P(0)$  em  $S(0)$ ;
  - 4: **Calcule** a aptidão de cada solução em  $S(0)$ ;
  - 5: **Armazene** em  $B(0)$  as soluções de  $S(0)$ ;
  - 6: **Enquanto** (*a condição de término não estiver satisfeita*) **Faça**
  - 7:    $g := g + 1$ ;
  - 8:   **Observe**  $P(g - 1)$  em  $S(g)$ ;
  - 9:   **Calcule** a aptidão de cada solução em  $S(g)$ ;
  - 10:   **Atualize**  $P(g)$  utilizando a  $q$ -porta;
  - 11:   **Aplique** restrições de probabilidade;
  - 12:   **Armazene** em  $B(g)$  as melhores soluções dentre  $B(g - 1)$  e  $S(g)$ ;
  - 13:   **Armazene** em  $b$  a melhor solução que encontra-se em  $B(g)$ ;
  - 14:   **Se** (*a condição de migração global estiver satisfeita*) **Então**
  - 15:     **Faça** migração global de  $b$  para  $B(g)$ ;
  - 16:   **Fim Se**
  - 17: **Fim Enquanto**
- 

### 4.2.2 Codificação

Os  $q$ -indivíduos definidos na Seção 3.2 são utilizados para representar as possíveis atribuições de estados da máquina de estados finitos. Na Figura 26 é ilustrada a representação utilizada pelo AEICQ na solução do problema de atribuição de estados. Cada um dos  $q$ -indivíduos representa a superposição linear de todas as atribuições possíveis com a mesma probabilidade. Cada parte do  $q$ -indivíduo representa um dos estados da máquina. A quantidade de  $q$ -bits em cada parte é igual ao número de sinais de estado da máquina. Por exemplo, as duas últimas linhas da Figura 26 representam duas atribuições possíveis para uma máquina com 4 estados obtidas, do mesmo  $q$ -indivíduo, após serem realizadas duas observações dos estados dos  $q$ -bits.

Pode-se notar que quando ocorre uma observação, um mesmo código pode ser usado para representar dois ou mais estados distintos. Este tipo de atribuição de estados é considerada não válida. Para diminuir a probabilidade da observação de atribuições não válidas durante o processo evolucionário, é aplicada uma penalidade durante a avaliação da aptidão deste tipo de solução. A etapa de avaliação das soluções será discutida com detalhes na Seção 4.2.3.

| Estados                 | $s_0$        |              | $s_1$        |              | $s_2$        |              | $s_3$        |              |
|-------------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|
| q-indivíduo             | $\alpha_1^0$ | $\alpha_2^0$ | $\alpha_1^1$ | $\alpha_2^1$ | $\alpha_1^2$ | $\alpha_2^2$ | $\alpha_1^3$ | $\alpha_2^3$ |
|                         | $\beta_1^0$  | $\beta_2^0$  | $\beta_1^1$  | $\beta_2^1$  | $\beta_1^2$  | $\beta_2^2$  | $\beta_1^3$  | $\beta_2^3$  |
| Observação <sub>1</sub> | 1            | 1            | 0            | 1            | 0            | 0            | 1            | 0            |
| Observação <sub>2</sub> | 0            | 0            | 1            | 1            | 1            | 0            | 0            | 1            |

Figura 26: Exemplo da codificação para o problema de atribuição de estados e duas possíveis observações dos estados dos q-bits

### 4.2.3 Avaliação de Aptidão

O AEICQ avalia a aptidão de cada solução binária obtida através da observação dos estados dos q-indivíduos. A avaliação de aptidão de cada atribuição de estados é feita baseada em duas regras (ARMSTRONG, 1962) e (HUMPHREY, 1958):

1. Dois ou mais estados que possuam o mesmo próximo estado devem receber códigos adjacentes.
2. Dois ou mais estados que sejam os próximos estados de um mesmo estado devem receber códigos adjacentes.

Como definido anteriormente, os *códigos adjacentes* são aquelas representações binárias com mesmo número de bits que diferem em apenas uma única posição. Por exemplo, os códigos 0110 e 1110 são adjacentes, mas os códigos 0110 e 1111 não são.

Satisfazendo a primeira regra, ao determinar as excitações dos *flip-flops*, os códigos associados aos estados anteriores ficarão juntos no mapa de Karnaugh (vertical ou horizontalmente), já que estes têm uma única posição diferente, e portanto, serão agrupados permitindo uma simplificação das equações de controle dos *flip-flops*. Por outro lado, satisfazendo a segunda regra, o estado anterior sendo o mesmo, a única posição diferente será aquela associada ao sinal de controle que permite decidir qual dos próximos estados deve ser usado. Por conseqüência, as combinações associadas ao sinal de controle e ao estado atual para os dois possíveis próximos estados ficarão juntas no mapa de Karnaugh (vertical ou horizontalmente) e levarão então a uma simplificação das equações de excitação dos *flip-flops*.

O agrupamento sugerido pela primeira regra deve prevalecer perante o agrupamento da segunda regra (ARMSTRONG, 1962). Isto é devido ao fato de que é mais provável que a primeira regra ocasione mais simplificações que a segunda, dependendo da combinação dos sinais da entrada primária da máquina de estados. A Figura 27 ilustra os três casos possíveis quanto à aplicação da primeira regra, enquanto a Figura 28 ilustra os dois casos possíveis quanto

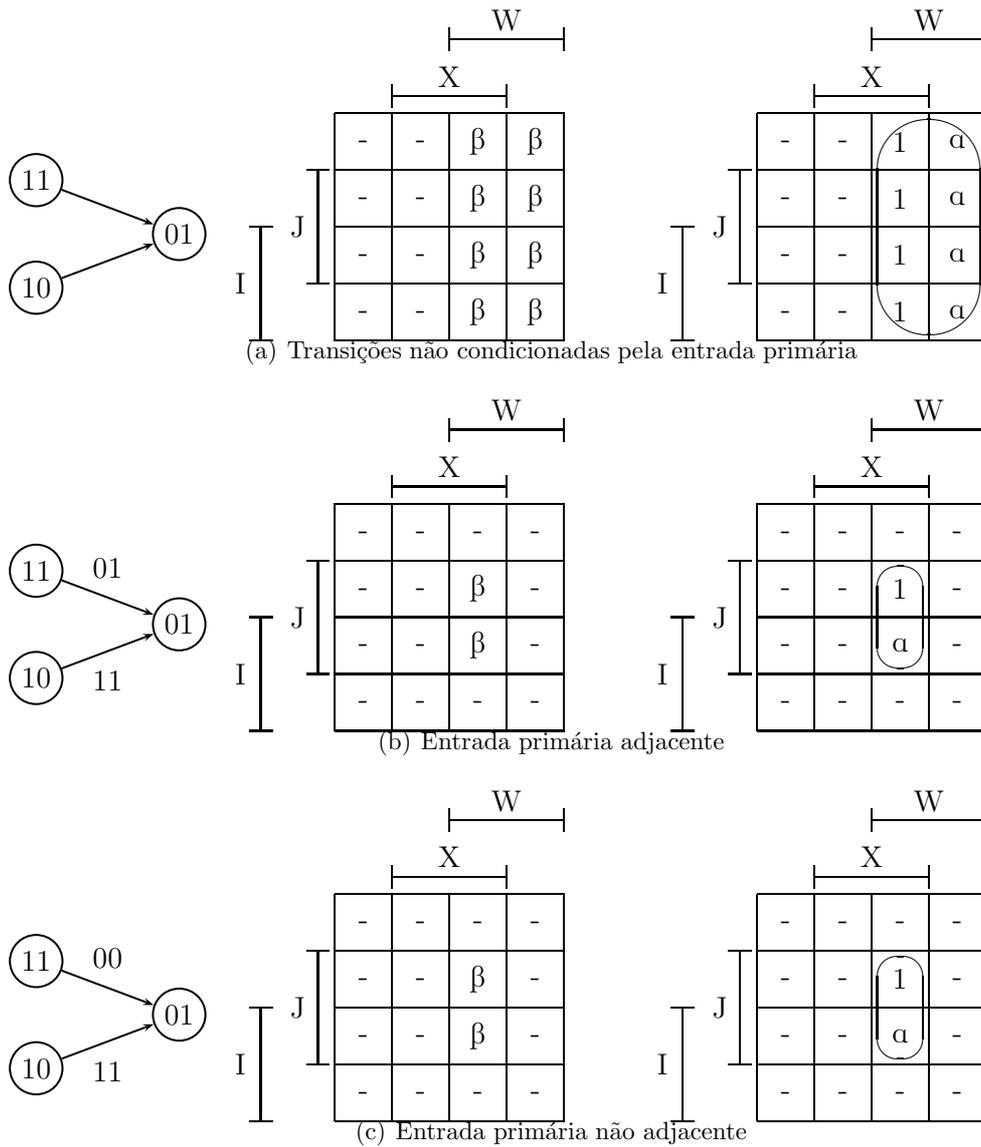


Figura 27: Exemplo ilustrativo das simplificações permitidas pela aplicação da primeira regra à aplicação da segunda. Note que nestes exemplos, *flip-flops* de tipo D são usados, os dois sinais da entrada primária são representadas por  $I$  e  $J$  e os sinais de estado são representados por  $X$  e  $W$ .

Embora úteis na determinação de atribuições de estados otimizadas, estas duas heurísticas são simples. Deve-se notar também que estas regras não têm nenhuma influência sobre a lógica de controle requerida para a geração dos sinais da saída primária.

Com o objetivo de permitir uma computação eficiente da aptidão de uma atribuição de estados tomando por base as duas heurísticas citadas, pode-se utilizar uma *matriz de adjacências*  $n \times n$ , onde  $n$  representa o número de estados da máquina. A parte triangular inferior

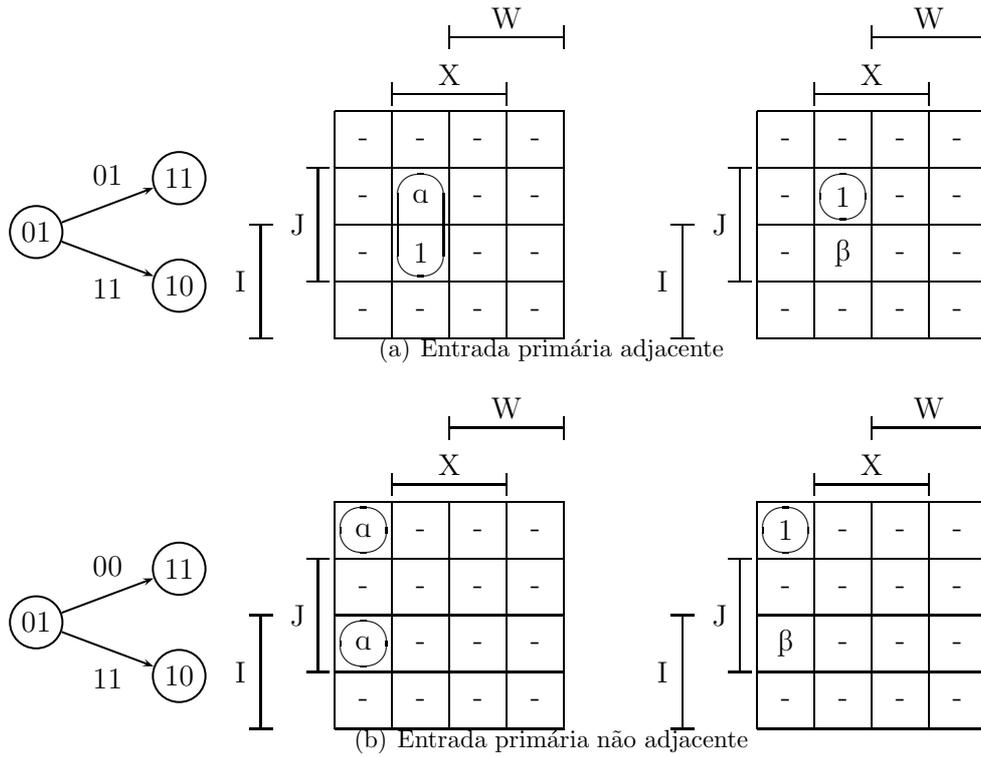


Figura 28: Exemplo ilustrativo das simplificações permitidas pela aplicação da segunda regra da matriz indica as adjacência esperadas dos estados com respeito à primeira regra, enquanto a parte triangular superior da matriz indica as adjacências esperadas com respeito à segunda regra. Os valores das entradas da matriz são calculados conforme descrito na Equação 31, tal que

$$MA_{i,j} = \begin{cases} \#(\text{próximo}(s_i) \cap (\text{próximo}(s_j))) & \text{se } i > j \\ \#(\text{predecessor}(s_i) \cap (\text{predecessor}(s_j))) & \text{se } i < j \\ 0 & \text{if } i = j, \end{cases} \quad (31)$$

onde  $MA$  indica a Matriz de Adjacências, as funções  $\text{próximo}(s)$  e  $\text{predecessor}(s)$  representam o conjunto de estados que são os próximos estados e os estados predecessores do estado  $s$ , respectivamente. Por exemplo, para a máquina de estados da Tabela 5, tem-se uma matriz de adjacências  $4 \times 4$  conforme ilustrado na Figura 29. Note que o conjunto  $S = \text{próximo}(s_i) \cap \text{próximo}(s_j)$  contém todos os estados que são próximos a ambos os estados  $s_i$  e  $s_j$ . Portanto, atribuir códigos adjacentes a estes estados tem uma importância proporcional à cardinalidade do conjunto  $S$ . Caso os códigos atribuídos aos estados  $s_i$  e  $s_j$  não fossem adjacentes, isso equivale a quebrar a primeira regra  $\#S$  vezes. Analogamente, o conjunto  $S' = \text{predecessor}(s_i) \cap \text{predecessor}(s_j)$  contém todos que são estados anteriores a ambos os estados  $s_i$  e  $s_j$ . Por

conseqüência, atribuir códigos adjacentes a estes estados tem uma importância proporcional à cardinalidade do conjunto  $S'$ . Caso  $s_i$  e  $s_j$  não recebessem códigos adjacentes, isso equivale a quebrar a segunda regra  $\#S$  vezes.

|                             | $s_0$ | $s_1$ | $s_2$ | $s_3$ |                      |
|-----------------------------|-------|-------|-------|-------|----------------------|
| $s_0$                       | 0     | 1     | 0     | 1     |                      |
| $s_1$                       | 1     | 0     | 2     | 0     | <i>Segunda regra</i> |
| <i>Primeira regra</i> $s_2$ | 1     | 0     | 0     | 0     |                      |
| $s_3$                       | 1     | 2     | 0     | 0     |                      |

Figura 29: Exemplo de uma matriz de adjacências

Utilizando a matriz de adjacências, a função de avaliação aplica um fator de 2 ou 1, toda a vez que a primeira ou a segunda regra são quebradas, respectivamente. Isso permite priorizar a primeira regra com relação à segunda. A Equação 32 mostra os detalhes da função de avaliação aplicada à uma atribuição de estados  $\sigma$ ,

$$f(\sigma) = \sum_{i \neq j \ \& \ s_i = s_j} \psi + \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} (MA_{i,j} + 2 * MA_{j,i}) * na(s_i, s_j), \quad (32)$$

onde a função  $na(s_i, s_j)$  retorna o valor 0 se os códigos que representam os estados  $s_i$  e  $s_j$  são adjacentes, e 1 caso contrário. Note que as atribuições de estados que codificam dois estados distintos com o mesmo código binário são penalizadas, onde  $\psi$  representa a penalidade. O valor de  $\psi$  foi ajustado em 30 para todos os experimentos descritos na Seção 4.3.

Por exemplo, para a máquina de estados representada na Tabela 5, a *Atribuição*<sub>01</sub> ( $s_0 \equiv 00, s_1 \equiv 11, s_2 \equiv 01, s_3 \equiv 10$ ) possui uma aptidão igual a 3, pois os códigos dos estados  $s_0$  e  $s_1$  não são adjacentes, portanto  $MA_{0,1} = 1$  e  $MA_{1,0} = 1$ . No caso da *Atribuição*<sub>02</sub> ( $s_0 \equiv 00, s_1 \equiv 10, s_2 \equiv 01, s_3 \equiv 11$ ) a aptidão é igual a 5, pois os códigos dos estados  $s_0$  e  $s_3$  não são adjacentes, portanto  $MA_{0,3} = 1$  e  $MA_{3,0} = 1$ . Os códigos dos estados  $s_1$  e  $s_2$  também não são adjacentes, portanto  $MA_{1,2} = 2$ .

O objetivo do AEICQ é encontrar uma atribuição de estados que minimize a função de avaliação da aptidão descrita na Equação 32. As atribuições com aptidão igual a 0 satisfazem todas as restrições de adjacência. Deve-se notar que tal atribuição nem sempre existe.

#### 4.2.4 Operador de Atualização

A porta quântica *rotação* descrita na Seção 3.2 é utilizada como operador de variação no AEICQ para a solução do problema de atribuição de estados. A Figura 30 mostra um exemplo de um q-indivíduo na geração inicial, onde as amplitudes de probabilidade dos q-bits são inicializadas com o valor  $1/\sqrt{2}$ , fazendo com que o q-indivíduo represente a superposição linear de todas as soluções possíveis com a mesma probabilidade.

| Estados     | $s_0$                |                      | $s_1$                |                      | $s_2$                |                      | $s_3$                |                      |
|-------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| q-indivíduo | $\frac{1}{\sqrt{2}}$ |
|             | $\frac{1}{\sqrt{2}}$ |

Figura 30: Exemplo de um q-indivíduo na geração inicial

A Figura 31 mostra um exemplo dos valores das amplitudes de probabilidade deste mesmo q-indivíduo, após a aplicação da q-porta. Conforme descrito na Seção 3.2, um número aleatório  $r$  é gerado no intervalo  $[0, 1]$ , se  $r < |\beta_k|^2$ , então o q-bit observado é 1, caso contrário, o q-bit observado é 0. Pode-se verificar na Figura 31 que a amplitude de probabilidade  $\beta$  referente ao primeiro e ao terceiro q-bit aumentou. O que significa que na próxima geração tem-se uma probabilidade maior (75%) de se observar o estado 1 para estes q-bits. No caso do sexto e do oitavo q-bit a amplitude de probabilidade  $\alpha$  aumentou, o que indica uma maior probabilidade do estado 0 ser observado na próxima geração.

| Estados     | $s_0$                |                      | $s_1$                |                      | $s_2$                |                      | $s_3$                |                      |
|-------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| q-indivíduo | $\frac{1}{2}$        | $\frac{1}{\sqrt{2}}$ | $\frac{1}{2}$        | $\frac{1}{\sqrt{2}}$ | $\frac{1}{\sqrt{2}}$ | $\frac{\sqrt{3}}{2}$ | $\frac{1}{\sqrt{2}}$ | $\frac{\sqrt{3}}{2}$ |
|             | $\frac{\sqrt{3}}{2}$ | $\frac{1}{\sqrt{2}}$ | $\frac{\sqrt{3}}{2}$ | $\frac{1}{\sqrt{2}}$ | $\frac{1}{\sqrt{2}}$ | $\frac{1}{2}$        | $\frac{1}{\sqrt{2}}$ | $\frac{1}{2}$        |

Figura 31: q-indivíduo após a operação de atualização das probabilidades

### 4.3 Resultados Experimentais

Nesta seção são apresentados os resultados do AEICQ aplicado ao problema de atribuição de estados. Em um primeiro momento são apresentados os resultados comparativos do desempenho do AEICQ frente ao algoritmo genético apresentado em (NEDJAH; MOURELLE, 2005a), o qual também utilizou as heurísticas de Armstrong e Humphrey na avaliação das aptidões das soluções. Em seguida, é feita uma comparação das lógicas de controle sintetizadas para cada uma das máquinas de estado utilizando-se as atribuições geradas pelo AEICQ, pelos

algoritmos genéticos (AMARAL; TUMER; GLOSH, 1995) e (ALI, 2003) e pelo método não evolucionário NOVA™ (VILLA; SANGIOVANNI-VINCENTELLI, 1990). Esta comparação tem como objetivo comprovar que as atribuições de estados obtidas pelo AEICQ, segundo as heurísticas de Armstrong e Humphrey, podem de fato conduzir a implementações de máquinas de estados com lógicas de controle otimizadas.

### 4.3.1 Comparação dos Resultados

Inicialmente, as atribuições de estados evoluídas pelo AEICQ são comparadas com as atribuições evoluídas pelo algoritmo genético (NEDJAH; MOURELLE, 2005a), tratado nesta dissertação como AG<sub>1</sub>, no qual também foram utilizadas as heurísticas de Armstrong e Humphrey na avaliação de aptidão das soluções. Para a realização dos experimentos foi selecionado um conjunto de máquinas de estados obtido de (ACM/SIGDA, 1989). Estas máquinas são referências (*benchmarks*) para testes envolvendo máquinas de estados finitos síncronas. A Tabela 9 apresenta os detalhes destas máquinas de estados, que estão listadas na ordem alfabética. As especificações completas destas máquinas encontram-se no Apêndice A.

Tabela 9: Características das máquinas de estados

| Máquina         | #Entradas | #Saídas | #Estados | #Transições |
|-----------------|-----------|---------|----------|-------------|
| <i>bbara</i>    | 4         | 2       | 10       | 60          |
| <i>bbsse</i>    | 7         | 7       | 16       | 56          |
| <i>dk14</i>     | 3         | 5       | 7        | 56          |
| <i>dk16</i>     | 2         | 3       | 27       | 108         |
| <i>donfile</i>  | 2         | 1       | 24       | 96          |
| <i>lion9</i>    | 2         | 1       | 9        | 25          |
| <i>modulo12</i> | 1         | 1       | 12       | 24          |
| <i>shiftreg</i> | 1         | 1       | 8        | 16          |
| <i>train11</i>  | 2         | 1       | 11       | 25          |

Os valores utilizados no ajuste de cada um dos parâmetros do AEICQ foram determinados após a realização de uma série de corridas para algumas das máquinas de estados da Tabela 9. Através destes testes, puderam-se observar algumas faixas dentro da qual o AEICQ apresentou melhores resultados e, então, o valor médio de cada faixa foi fixado para a realização dos experimentos.

Os valores dos ângulos utilizados na etapa de atualização dos q-bits foram ajustados em 0 para  $\theta_1, \theta_2, \theta_4, \theta_6, \theta_7, \theta_8$  e  $0,005\pi$  e  $-0,005\pi$  para  $\theta_3$  e  $\theta_5$ , respectivamente. O módulo dos ângulos  $\theta_3$  e  $\theta_5$  foram testados em uma faixa que variou de  $0,001\pi$  a  $0,1\pi$ , conforme recomendado em (HAN, 2003). Os melhores resultados durante os testes foram obtidos para uma faixa com

variação de  $0,003\pi$  a  $0,007\pi$ , e portanto, foram ajustados em  $0,005\pi$  para a realização dos experimentos.

O parâmetro  $G$ , que representa o número de gerações sem atualização da melhor solução, e que é utilizado na operação de migração global foi testado com valores que variaram de 50 a 1000 gerações. Os melhores resultados foram obtidos para uma faixa com variação de 300 a 500, sendo então fixado em 400 para a realização dos experimentos.

Para o ajuste do tamanho da população, pôde-se observar durante os testes que o aumento da quantidade de  $q$ -indivíduos, na média, conduziu às melhores soluções. Porém, um aumento no tamanho da população aumenta também o tempo de execução do algoritmo. Portanto, optou-se por fixar o tamanho da população em 50  $q$ -indivíduos.

No caso do ajuste do número máximo de gerações foram realizados alguns testes onde observou-se para cada uma das máquinas a geração a partir da qual o AEICQ não conseguia mais melhorar a solução obtida. Além deste critério de término, o algoritmo também é encerrado caso encontre uma solução que atenda todas as restrições de adjacência, o que nem sempre é possível conforme descrito na Seção 4.2. Para as máquinas de estados utilizadas nestes experimentos, o número máximo de gerações foi fixado em 5000. Apenas para as máquinas *dk16* e *donfile*, as quais possuem um número grande de estados e transições de estados, o número de gerações foi fixado em 10000.

A Tabela 10 mostra as melhores atribuições de estados obtidas pelo  $AG_1$  e pelo AEICQ e a Tabela 11 mostra a aptidão das atribuições obtidas. A coluna *#Res. Adj.* indica o número esperado de restrições de adjacência. Por exemplo, no caso da máquina de estados *shiftreg*, todas as 24 restrições de adjacência esperadas foram atendidas pelas atribuições de estado obtidas pelos dois algoritmos. No entanto, as atribuições obtidas para a máquina de estados *lion9*, não atenderam 21 das restrições de adjacência esperadas. Os resultados demonstram que o AEICQ foi capaz de evoluir atribuições de estados melhores ou iguais às obtidas pelo  $AG_1$  segundo as heurísticas de Armstrong e Humphrey. Os gráficos apresentados na Figura 32 – Figura 36 mostram o progresso da evolução da aptidão da melhor solução e da aptidão média da população para cada uma das máquinas de estados usadas neste experimento.

Nesta segunda parte dos experimentos, as lógicas de controle das máquinas de estados indicadas na Tabela 12 foram sintetizadas. Para efeito de comparação, estas lógicas foram sintetizadas não só para as atribuições obtidas pelo AEICQ, como também para as atribuições apresentadas em (AMARAL; TUMER; GLOSH, 1995) e (ALI, 2003). Nestes trabalhos, os autores utilizaram os algoritmos genéticos e os seus resultados foram comparados aos obtidos pelo

Tabela 10: Melhor atribuição de estados obtida por cada um dos algoritmos

| Máquina         | Algoritmo       | Atribuição de Estados   |
|-----------------|-----------------|---|
| <i>bbara</i>    | AG <sub>1</sub> | [3,0,8,12,1,9,13,11,10,2]                                       |
|                 | AEICQ           | [4,5,1,9,13,12,14,15,7,6]                                       |
| <i>bbsse</i>    | AG <sub>1</sub> | [15,14,9,12,1,4,3,7,6,10,2,11,13,0,5,8]                         |
|                 | AEICQ           | [5,3,11,7,9,6,14,10,8,12,4,1,0,2,13,15]                         |
| <i>dk14</i>     | AG <sub>1</sub> | [3,7,1,0,5,6,2]   |
|                 | AEICQ           | [5,7,4,0,6,3,1]   |
| <i>donfile</i>  | AG <sub>1</sub> | [2,18,17,1,29,21,6,22,7,0,4,20,19,3,23,16,9,8,13,5,12,28,25,24] |
|                 | AEICQ           | [7,6,23,31,26,27,15,14,13,5,10,4,22,30,12,8,11,9,18,19,2,0,3,1] |
| <i>lion9</i>    | AG <sub>1</sub> | [10,8,12,9,13,15,7,3,11]  |
|                 | AEICQ           | [11,9,3,1,2,0,8,10,14]  |
| <i>shiftreg</i> | AG <sub>1</sub> | [5,7,4,6,1,3,0,2]   |
|                 | AEICQ           | [4,0,2,6,5,1,3,7]   |
| <i>train11</i>  | AG <sub>1</sub> | [2,6,1,4,0,14,10,9,8,11,3]                                      |
|                 | AEICQ           | [9,11,13,3,1,2,0,12,8,5,4]                                      |

Tabela 11: Aptidão das melhores atribuições obtidas pelos algoritmos

| Máquinas de Estados | #Res. Adj. | AG <sub>1</sub> | AEICQ |
|---------------------|------------|-----------------|-------|
| <i>bbara</i>        | 225        | 127             | 126   |
| <i>bbsse</i>        | 328        | 223             | 217   |
| <i>dk14</i>         | 139        | 68              | 68    |
| <i>donfile</i>      | 432        | 247             | 246   |
| <i>lion9</i>        | 69         | 21              | 21    |
| <i>shiftreg</i>     | 24         | 0               | 0     |
| <i>train11</i>      | 57         | 18              | 17    |

método não evolucionário NOVA<sup>TM</sup> (VILLA; SANGIOVANNI-VINCENTELLI, 1990). O algoritmo genético implementados em (AMARAL; TUMER; GLOSH, 1995) e (ALI, 2003), serão tratados nesta dissertação como AG<sub>2</sub> e AG<sub>3</sub>, respectivamente. Os resultados obtidos pelo NOVA<sup>TM</sup> e que são apresentados em (AMARAL; TUMER; GLOSH, 1995) e (ALI, 2003) também serão usados nesta comparação.

Para a obtenção dos circuitos, foi utilizada a ferramenta de síntese de circuitos seqüenciais ABC<sup>TM</sup> distribuída pelo departamento EECS (*Electrical Engineering and Computer Sciences*) da Universidade da Califórnia (ABC, 2005). A biblioteca de portas lógicas que foi utilizada para obtenção dos circuitos é composta das seguintes portas: NOT, AND, OR, NAND, NOR, XOR e XNOR. A ferramenta ABC<sup>TM</sup> fornece como saída as equações do circuito sintetizado, assim como, a quantidade de portas, a área consumida e o atraso de propagação do circuito.

A Tabela 13 mostra as características dos circuitos sintetizados para cada uma das atribuições obtidas pelos algoritmos genéticos AG<sub>2</sub> e AG<sub>3</sub> e pelo NOVA<sup>TM</sup>. A Tabela 14 mostra as

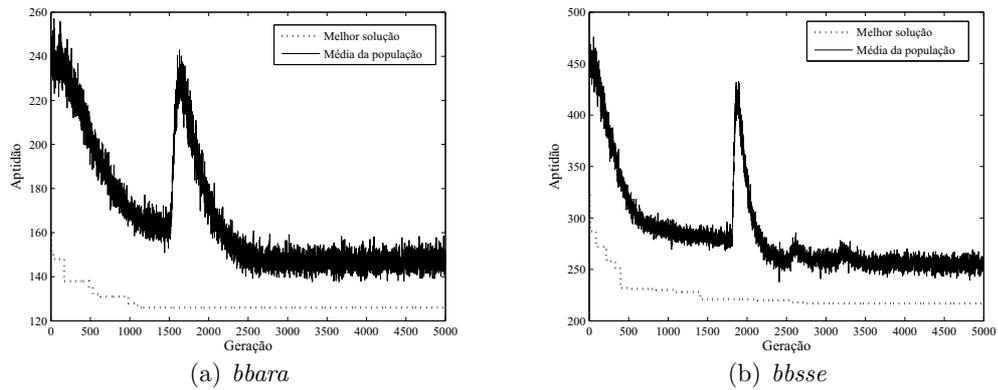


Figura 32: Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados *bbara* e *bsse*

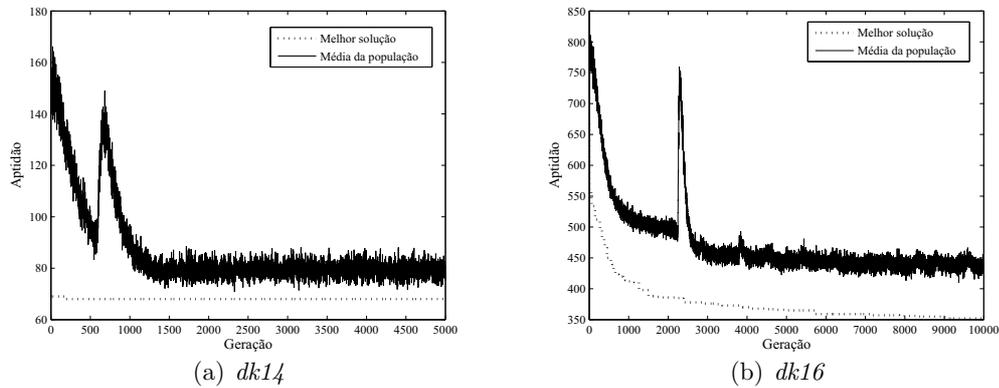


Figura 33: Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados *dk14* e *dk16*

características dos circuitos sintetizados com as atribuições obtidas pelo AEICQ. Os resultados apresentados na Tabela 13 e Tabela 14 são mostrados na forma de histogramas na Figura 37 para comparação do número de portas, na Figura 38 para comparação da área e na Figura 39 para comparação do atraso de propagação.

Os melhores valores obtidos para a quantidade de portas, área consumida e atraso de cada uma das máquinas estão marcados em **negrito**. Pode-se observar que as atribuições de estados obtidas pelo AEICQ geraram circuitos com a menor quantidade de portas e o menor consumo de área que os demais métodos. O atraso de propagação dos circuitos que foram sintetizados com as atribuições geradas pelo AEICQ é comparável aos atrasos dos circuitos obtidos por outros métodos, sendo inclusive inferior na maioria dos casos.

Os resultados obtidos demonstram que a utilização dos algoritmos evolucionários inspirados na computação quântica em conjunto com as heurísticas de Armstrong e Humphrey é

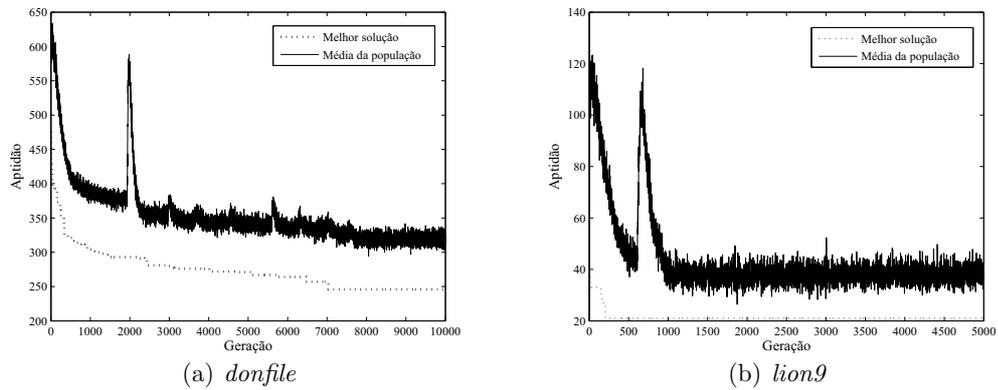


Figura 34: Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados *donfile* e *lion9*

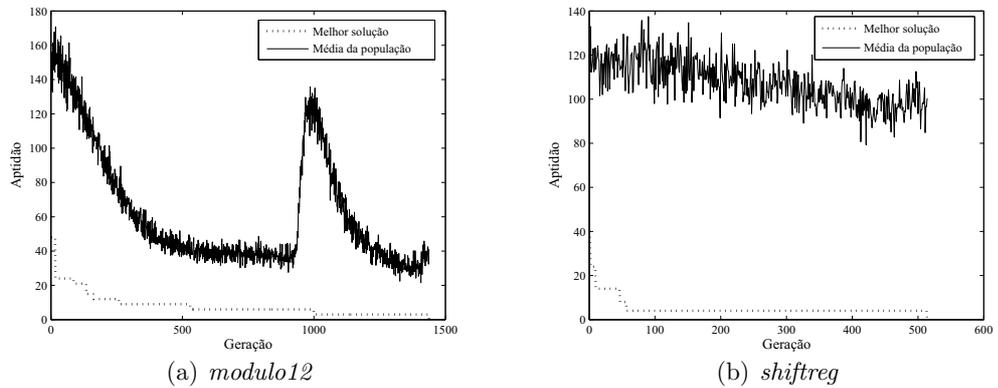


Figura 35: Progresso da aptidão da melhor solução e da aptidão média da população para as máquinas de estados *modulo12* e *shiftreg*

capaz de produzir boas soluções para o problema de atribuição de estados.

### 4.3.2 Discussão dos Resultados

Os resultados apresentados na seção anterior mostram que o AEICQ é capaz de obter atribuições de estados melhores que as obtidas pelos algoritmos genéticos e pela ferramenta NOVA™. Para determinar se os resultados obtidos pelo AEICQ são significativos, é preciso realizar um teste estatístico de aderência ou confiança. O teste mais usado para comparação de proporções é o teste  $\chi^2$  (ANDRE, 2008). Este teste é capaz de descobrir se a diferença que existe entre dois grupos de dados é significativa ou aconteceu por pura coincidência. A seguir, é explicado sucintamente como se aplica o teste  $\chi^2$ .

O teste  $\chi^2$  leva em conta a diferença entre os valores *observados* e os *esperados* para cada um dos quesitos considerados. Os valores observados são aqueles obtidos experimentalmente,

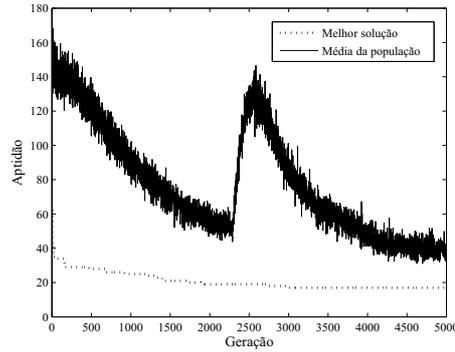
(a) *train11*

Figura 36: Progresso da aptidão da melhor solução e da aptidão média da população para a máquina de estados *train11*

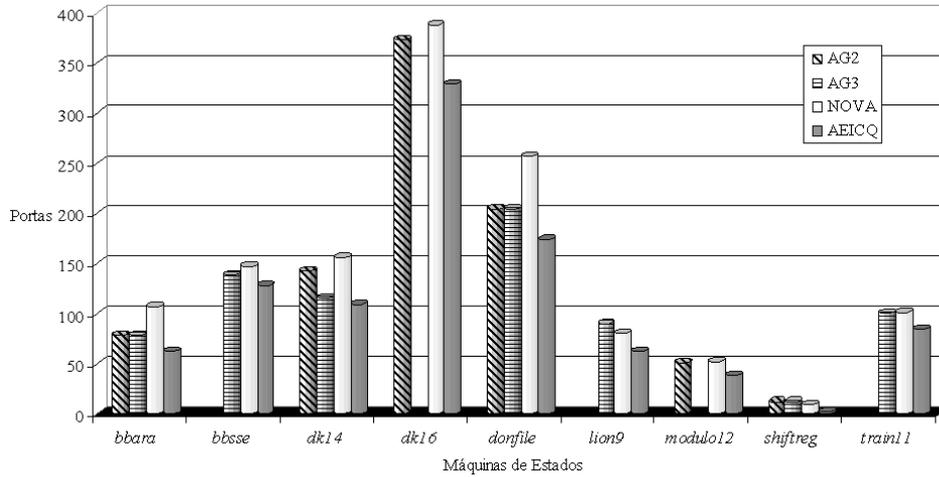


Figura 37: Comparação das lógicas de controle em termos da quantidade de portas

enquanto que os valores esperados se referem à distribuição hipotética baseada nas proporções globais dentro dos contextos comparados. Seja  $\lambda_o^{(a,m,q)}$  o valor observado para o quesito  $q$  obtido pelo algoritmo  $a$  quando aplicado à máquina de estados  $m$  e  $\lambda_e^{(a,m,q)}$  o valor esperado correspondente. Foram realizados os testes AEICQ  $\times$  NOVA<sup>TM</sup>, AEICQ  $\times$  AG<sub>2</sub> e AEICQ  $\times$  AG<sub>3</sub> separadamente para os quesitos *número de portas*, *área* e *atraso*. O valor  $\lambda_e^{(a,m,q)}$  é calculado conforme mostra a Equação 33:

$$\lambda_e^{(a,m,q)} = \frac{\sum_{(x,z) \in A \times Q} \lambda_o^{(x,m)} \times \sum_{y \in M} \lambda_o^{(a,y)}}{\sum_{(x,y,z) \in A \times M \times Q} \lambda_o^{(x,y,x)}}, \quad (33)$$

onde  $A = \{\text{AEICQ}, \text{NOVA}^{\text{TM}}\}$ ,  $A = \{\text{AEICQ}, \text{AG}_2\}$  ou  $A = \{\text{AEICQ}, \text{AG}_3\}$ ,  $M \subseteq \{bbara, bbsse, dk14, dk16, donfile, lion9, modulo12, shiftreg, train11\}$  e  $Q = \{\text{número de portas}, \text{área},$

Tabela 12: Melhor atribuição de estados obtida por cada um dos métodos

| Máq.            | Alg.               | Atribuição de Estados  |
|-----------------|--------------------|--|
| <i>bbara</i>    | AG <sub>2</sub>    | [0,6,2,14,4,5,13,7,3,1]  |
|                 | AG <sub>3</sub>    | [0,6,2,14,4,5,13,7,3,1]  |
|                 | NOVA <sup>TM</sup> | [9,0,2,13,3,8,15,5,4,1]  |
|                 | AEICQ              | [4,5,1,9,13,12,14,15,7,6]  |
| <i>bbsse</i>    | AG <sub>3</sub>    | [0,4,10,5,12,13,11,14,15,8,9,2,6,7,3,1]                                    |
|                 | NOVA <sup>TM</sup> | [12,0,6,1,7,3,5,4,11,10,2,13,9,8,15,14]                                    |
|                 | AEICQ              | [5,3,11,7,9,6,14,10,8,12,4,1,0,2,13,15]                                    |
| <i>dk14</i>     | AG <sub>2</sub>    | [5,7,1,3,6,0,4]  |
|                 | AG <sub>3</sub>    | [0,4,2,1,5,7,3]  |
|                 | NOVA <sup>TM</sup> | [1,4,0,2,7,5,3]  |
|                 | AEICQ              | [5,7,4,0,6,3,1]  |
| <i>dk16</i>     | AG <sub>2</sub>    | [12,8,1,27,13,28,14,29,0,16,26,9,2,4,3,10,11,17,24,5,18,7,21,25,6,20,19]   |
|                 | NOVA <sup>TM</sup> | [12,7,1,3,4,10,23,24,5,27,15,16,11,6,0,20,31,2,13,25,21,14,18,19,30,17,22] |
|                 | AEICQ              | [14,30,22,6,4,5,13,25,18,20,31,9,10,26,23,28,29,7,15,3,16,8,21,17,1,11,24] |
| <i>donfile</i>  | AG <sub>2</sub>    | [0,12,9,1,6,7,2,14,11,,17,20,23,8,15,10,16,21,19,4,5,22,18,13,3]           |
|                 | NOVA <sup>TM</sup> | [12,14,13,5,23,7,15,31,10,8,29,25,28,6,3,2,4,0,30,21,9,17,12,1]            |
|                 | AEICQ              | [7,6,23,31,26,27,15,14,13,5,10,4,22,30,12,8,11,9,18,19,2,0,3,1]            |
| <i>lion9</i>    | AG <sub>3</sub>    | [0,4,12,13,15,1,3,7,5]   |
|                 | NOVA <sup>TM</sup> | [2,0,4,6,7,5,3,1,11]   |
|                 | AEICQ              | [11,9,3,1,2,0,8,10,14]   |
| <i>mod12</i>    | AG <sub>2</sub>    | [0,8,1,2,3,9,10,4,11,12,5,6]   |
|                 | NOVA <sup>TM</sup> | [0,15,1,14,2,13,3,12,4,11,5,10]  |
|                 | AEICQ              | [15,7,6,14,10,2,3,1,5,13,9,11]   |
| <i>shiftreg</i> | AG <sub>2</sub>    | [0,2,5,7,4,6,1,3]  |
|                 | AG <sub>3</sub>    | [0,2,5,7,4,6,1,3]  |
|                 | NOVA <sup>TM</sup> | [0,4,2,6,3,7,1,5]  |
|                 | AEICQ              | [4,0,2,6,5,1,3,7]  |
| <i>train11</i>  | AG <sub>3</sub>    | [0,8,2,9,13,12,4,7,5,3,1]  |
|                 | NOVA <sup>TM</sup> | [0,8,2,9,1,10,4,6,5,3,7]   |
|                 | AEICQ              | [9,11,13,3,1,2,0,12,8,5,4]   |

*atraso*}. Quando o teste  $\chi^2$  é usado, a decisão de aderência depende do valor de  $\chi^2$  que é calculado conforme descrito na Equação 34, onde  $A$ ,  $M$  e  $Q$  são definidos da mesma forma que para Equação 33. Os valores de  $\chi^2$  para cada uma das comparações são apresentados na Tabela 15. O uso do teste  $\chi^2$  não é recomendável quando as proporções são pequenas. Portanto, os valores de atrasos foram convertidos para 0,1 ns ao invés de ns, evitando assim a limitação de aplicação do teste, tal que

$$\chi^2 = \sum_{(a,m,q) \in A \times M \times Q} \frac{\left(\lambda_o^{(a,m,q)} - \lambda_e^{(a,m,q)}\right)^2}{\lambda_e^{(a,m,q)}}. \quad (34)$$

O valor crítico para o nível de aderência de 0,05 (ou 95% para o nível de confiança)

Tabela 13: Comparação das lógicas de controle utilizando as atribuições obtidas pelo AG<sub>2</sub>, AG<sub>3</sub> e NOVA<sup>TM</sup>

| Máquina<br>de Estados | AG <sub>2</sub> |      |             | AG <sub>3</sub> |      |             | NOVA <sup>TM</sup> |      |             |
|-----------------------|-----------------|------|-------------|-----------------|------|-------------|--------------------|------|-------------|
|                       | portas          | área | atraso      | portas          | área | atraso      | portas             | área | atraso      |
| <i>bbara</i>          | 78              | 78   | <b>0,60</b> | 78              | 78   | <b>0,60</b> | 107                | 107  | 0,81        |
| <i>bbsse</i>          | –               | –    | –           | 139             | 140  | <b>0,67</b> | 147                | 150  | <b>0,67</b> |
| <i>dk14</i>           | 143             | 146  | 0,60        | 115             | 116  | 0,53        | 156                | 157  | 0,53        |
| <i>dk16</i>           | 374             | 379  | 0,74        | –               | –    | –           | 388                | 393  | 0,81        |
| <i>donfile</i>        | 204             | 207  | 0,67        | 204             | 207  | 0,67        | 257                | 260  | 0,81        |
| <i>lion9</i>          | –               | –    | –           | 90              | 90   | 0,60        | 80                 | 80   | 0,60        |
| <i>modulo12</i>       | 51              | 51   | 0,42        | –               | –    | –           | 52                 | 52   | <b>0,39</b> |
| <i>shiftreg</i>       | 12              | 12   | 0,32        | 12              | 12   | 0,32        | 9                  | 9    | <b>0,18</b> |
| <i>train11</i>        | –               | –    | –           | 100             | 100  | 0,63        | 101                | 102  | 0,60        |

Tabela 14: Comparação das lógicas de controle utilizando as atribuições obtidas pelo AEICQ

| Máquina<br>de Estados | AEICQ      |            |             |
|-----------------------|------------|------------|-------------|
|                       | portas     | área       | atraso      |
| <i>bbara</i>          | <b>62</b>  | <b>63</b>  | 0,67        |
| <i>bbsse</i>          | <b>128</b> | <b>128</b> | 0,70        |
| <i>dk14</i>           | <b>109</b> | <b>110</b> | <b>0,53</b> |
| <i>dk16</i>           | <b>329</b> | <b>333</b> | <b>0,74</b> |
| <i>donfile</i>        | <b>174</b> | <b>174</b> | <b>0,60</b> |
| <i>lion9</i>          | <b>62</b>  | <b>63</b>  | <b>0,53</b> |
| <i>modulo12</i>       | <b>38</b>  | <b>38</b>  | 0,42        |
| <i>shiftreg</i>       | <b>2</b>   | <b>6</b>   | 0,30        |
| <i>train11</i>        | <b>85</b>  | <b>85</b>  | <b>0,53</b> |

é considerado como limite para acatar a hipótese de aderência. O grau de liberdade depende da quantidade de resultados usados para calcular  $\chi^2$ . Assumindo que esses resultados são organizados em uma estrutura matricial de  $l$  linhas e  $c$  colunas, o grau de liberdade é  $(l - 1) \times (c - 1)$ . Nesta comparação, o número de linhas coincide com o de máquinas de estados e o número de colunas é 6, sendo duas para cada quesito (uma para cada algoritmo). No caso da comparação AEICQ  $\times$  NOVA<sup>TM</sup>, todas as 9 máquinas de estados listadas na Tabela 13 foram usadas, enquanto no caso das comparações AEICQ  $\times$  AG<sub>2</sub> e AEICQ  $\times$  AG<sub>3</sub>, somente algumas máquinas, 6 e 7 respectivamente, foram usadas, tendo em vista a disponibilidade de resultados, conforme mostra a Tabela 13. Com base nesta análise estatística, o AEICQ pode ser considerado significativamente melhor do que NOVA<sup>TM</sup>, AG<sub>2</sub> e AG<sub>3</sub>.

Nos experimentos apresentados, foi utilizada uma população com tamanho igual a 50 q-indivíduos. No entanto, foi observado que para algumas máquinas de estados, como por exemplo, as máquinas *shiftreg* e a *lion9*, a melhor solução obtida com uma população com

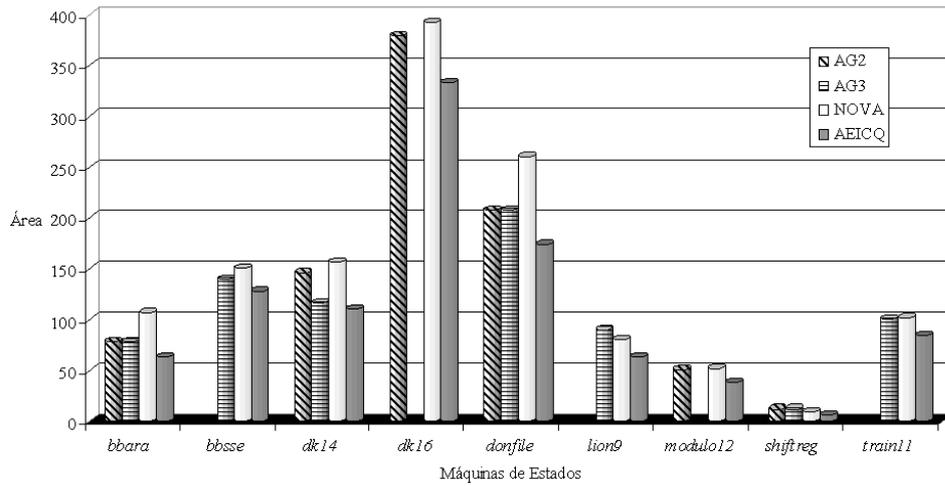


Figura 38: Comparação das lógicas de controle em termos de área consumida

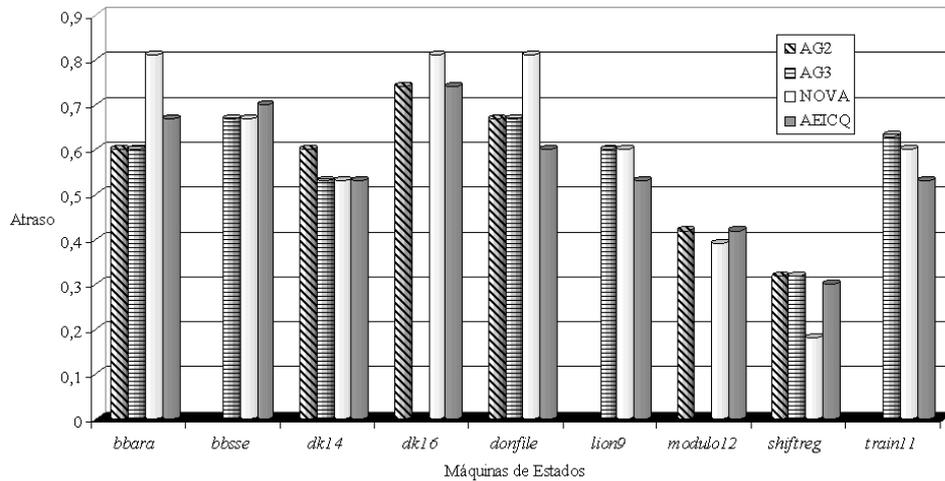


Figura 39: Comparação das lógicas de controle em termos de atraso de propagação

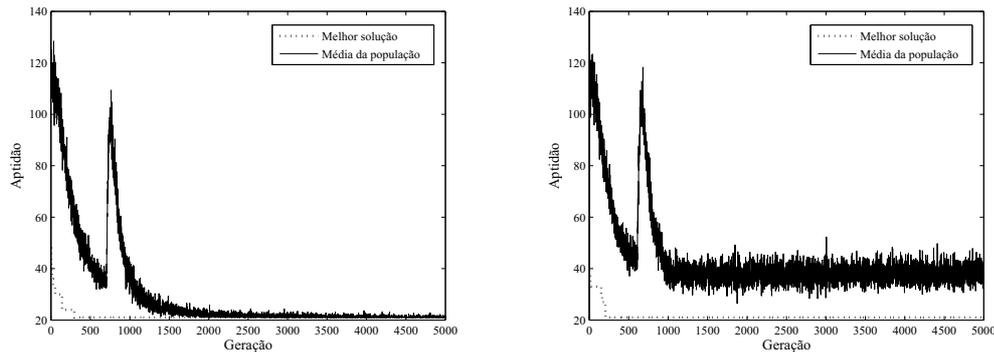
tamanho igual a 50 também pode ser obtida com populações muito menores, podendo chegar a apenas um único  $\varphi$ -indivíduo. Porém, neste caso o número de corridas que atingem o melhor resultado diminui. No caso da máquina *shiftreg*, por exemplo, o ótimo global é obtido em todas as corridas e em apenas 50% destas, quando a população tem tamanho igual a 50  $\varphi$ -indivíduos e 1  $\varphi$ -indivíduo, respectivamente.

Durante os experimentos feitos, também foi possível observar que o AEICQ se mostrou muito robusto quanto à escolha da magnitude dos ângulos  $\theta_3$  e  $\theta_5$  dentro da faixa sugerida por Han e Kim em (HAN; KIM, 2002), por isso foram utilizados os mesmos valores para todas as máquinas.

O impacto da etapa de limitação das amplitudes de probabilidade dos  $\varphi$ -bits pode ser observado na Figura 40. A Figura 40–(a) mostra que quando não existe a limitação das

Tabela 15: Grau de liberdade, valores de  $\chi^2$  calculado, valores críticos de  $\chi^2$  para o nível de confiança 99,5% e níveis de confiança obtidos para as diferentes comparações

| Comparação                        | Grau de liberdade | $\chi^2$ | Valor crítico | Nível de confiança |
|-----------------------------------|-------------------|----------|---------------|--------------------|
| AEICQ $\times$ NOVA <sup>TM</sup> | 40                | 73,302   | 66,766        | >99,5%             |
| AEICQ $\times$ AG <sub>2</sub>    | 25                | 68,281   | 46,928        | >99,5%             |
| AEICQ $\times$ AG <sub>3</sub>    | 30                | 64,740   | 53,672        | >99,5%             |



(a) Sem a etapa de limitação das amplitudes de probabilidade      (b) Com a etapa de limitação das amplitudes de probabilidade

Figura 40: Observação do efeito da etapa de limitação das amplitudes de probabilidade

amplitudes de probabilidade e o algoritmo não consegue obter nenhuma nova solução melhor, a média das aptidões da população se aproxima muito da aptidão do melhor  $q$ -indivíduo, obtido até então. Isto ocorre devido ao fato das probabilidades dos estados estarem praticamente iguais a 100%, o que leva a observação da mesma solução em quase todas as gerações. Na Figura 40–(b) pode-se observar que a média da aptidão da população se mantém afastada da aptidão da melhor solução. Isto ocorre devido à limitação das amplitudes de probabilidade dos  $q$ -bits, a qual permite que novas soluções sejam testadas ainda que o algoritmo já tenha encontrado o ótimo global ou esteja preso em uma solução local. Portanto, a etapa de limitação das amplitudes de probabilidade permite que o algoritmo escape de soluções locais.

Os efeitos da etapa de migração global podem ser observados na Figura 41. Nesta etapa, as melhores soluções em  $B(g)$ , as quais estão sendo utilizadas na operação de atualização dos  $q$ -bits, são substituídas pela melhor solução  $b$ . Esta operação introduz uma perturbação na população cada vez que é executada, aumentando assim a sua diversidade. Os picos que estão assinalados no gráfico indicam três momentos onde pode-se perceber claramente o efeito da migração global na aptidão média da população. Estes picos surgem logo após a execução da etapa de migração, a qual foi ajustada para ocorrer sempre que o algoritmo atinja 400 gerações sem conseguir uma nova melhor solução. Portanto, assim como a etapa de limitação

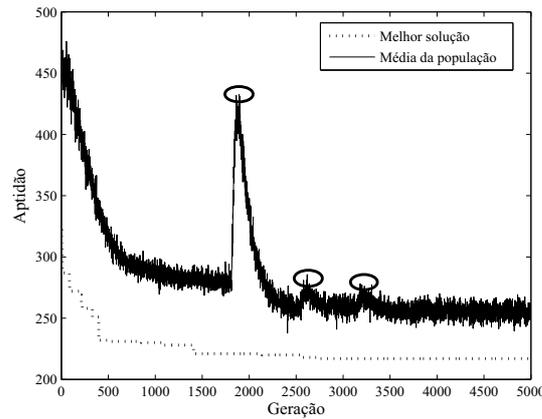


Figura 41: Demonstração do efeito da etapa de migração global

das amplitudes de probabilidade, a etapa de migração global também permite ao algoritmo escapar de soluções locais.

O AEICQ foi implementado na linguagem de programação do MATLAB<sup>TM</sup>(versão 7.0) e foi executado em um micro-computador com processador de 2,13 GHz, 2 MB de memória *cache* e 2 GB de memória RAM. O tempo médio de execução do algoritmo variou de poucos segundos até cerca de 20 minutos, dependendo da máquina de estados e dos valores dos parâmetros de ajuste.

## 4.4 Resumo do Capítulo

A atribuição de estados consiste de uma das etapas importantes no projeto de circuitos seqüenciais síncronos. A Seção 4.1 ressaltou a importância da utilização de uma atribuição de estados otimizada e demonstrou também a dificuldade de se obter tal atribuição. Na Seção 4.2 foi apresentada uma nova abordagem para a solução do problema de atribuição de estados baseada em um algoritmo evolucionário inspirado na computação quântica, chamado de AEICQ. Na Seção 4.3 os resultados alcançados pelo AEICQ foram comparados com os obtidos pelos algoritmos genéticos e pela ferramenta NOVA<sup>TM</sup>. Os resultados apresentados demonstram que o AEICQ é uma ferramenta adequada para ser usada para gerar atribuições de estados na síntese automática de máquinas de estados finitos. No Capítulo 5 são abordados alguns conceitos de uma nova área de pesquisa chamada de *eletrônica evolucionária*. Além disso, é investigada a aplicação do AEICQ na síntese da lógica de controle de máquinas de estados.

# Capítulo 5

## Projeto Evolucionário de Máquinas de Estados Finitos

**E**STE capítulo propõe o uso do Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ), apresentado no Capítulo 4, como uma ferramenta para a síntese de lógicas compactas e eficientes para o controle de sistemas seqüenciais síncronos. Na Seção 5.1, são apresentados os conceitos básicos da eletrônica evolucionária. Na Seção 5.2, são destacadas as características dos projetos evolucionários no domínio da eletrônica. Na Seção 5.3, são apresentadas algumas aplicações envolvendo a eletrônica evolucionária. Na Seção 5.4, o AEICQ é aplicado na síntese evolucionária da lógica de controle das máquinas de estados finitos. Na Seção 5.5, os resultados experimentais obtidos são apresentados. Já na Seção 5.5.1, alguns dos parâmetros de mérito dos circuitos lógicos evoluídos pelo AEICQ são comparados com aqueles obtidos por outros métodos. Em seguida, na Seção 5.5.2, a qualidade dos resultados obtidos é discutida.

### 5.1 Eletrônica Evolucionária

Tarefas cada vez mais complexas têm sido demandadas dos circuitos eletrônicos, ao passo que aumentam as exigências quanto aos requisitos de desempenho destes circuitos. Conseqüentemente, técnicas de projeto de circuitos cada vez mais aprimoradas se fazem necessárias para atender a estes requisitos.

Recentemente, surgiu uma nova área de pesquisa que aplica técnicas evolucionárias no projeto, otimização e síntese de circuitos eletrônicos. Esta área é denominada de *eletrônica evolucionária*. A principal motivação na criação de uma ferramenta automática para o projeto de circuitos eletrônicos baseada nos princípios da evolução natural é poder substituir o projetista de circuitos em determinadas tarefas, permitindo que ele se preocupe somente com a especificação do sistema.

A eletrônica evolucionária pode ser classificada de acordo com as seguintes propriedades (ZEBULUM, 1999) e (ZEBULUM; PACHECO; VELLASCO, 2002): *tipo de projeto, natureza do projeto e plataforma evolucionária*.

### 5.1.1 Tipo de Projeto

A utilização da eletrônica evolucionária pode ter como objetivos a *otimização* e/ou *síntese* de circuitos.

A aplicação da eletrônica evolucionária em otimização acontece quando um projeto inicial do circuito, funcional mas não otimizado, encontra-se disponível. Esta otimização normalmente é um problema de múltiplos objetivos que, em geral, possuem diversas restrições que devem ser levadas em consideração na avaliação do desempenho dos circuitos.

No caso da síntese de circuitos eletrônicos, a eletrônica evolucionária é utilizada na obtenção de circuitos que atendam a uma determinada especificação, que pode ser fornecida no domínio digital ou analógico. Esta tarefa é mais complexa que a de otimização, uma vez que envolve a busca de uma topologia, escolha de componentes, escolha de valores de componentes e, se for o caso, a otimização do circuito sintetizado (ZEBULUM, 1999) (ZEBULUM; PACHECO; VELLASCO, 2002).

### 5.1.2 Natureza do Projeto

A eletrônica evolucionária pode ser usada para evoluir circuitos *analógicos* e/ou *digitais*. Na síntese de circuitos digitais pode-se observar na literatura duas formas básicas do algoritmo evolucionário manipular o circuito: a *nível de portas* (THOMPSON, 1996) e a *nível de funções* (LIU; MURAKAWA; HIGUCHI, 1997). No primeiro caso, a unidade básica do circuito digital que é manipulada pelo algoritmo evolucionário é uma porta lógica, tais como, AND, OR e NOT. Já no segundo caso, o algoritmo evolucionário manipula blocos funcionais digitais mais complexos, tais como unidades lógicas e aritméticas, multiplicadores e registradores. Em geral, a *representação* é elaborada tomando por base que cada gene codifica a natureza da unidade básica de circuito, que pode ser uma porta lógica ou um bloco funcional, e a origem das entradas destes. A forma básica de *avaliação* das soluções em termos de precisão e qualidade consiste na comparação com a especificação do circuito digital, dada na forma de uma tabela verdade.

Embora a maior parte dos circuitos integrados modernos utilize tecnologia digital, circuitos analógicos são necessários para implementação da interface destes com o mundo exterior (LOHN; COLOMBANO, 1998). Em geral, na *representação* de circuitos analógicos os genes in-

cluem informações a respeito do valor dos componentes, além da natureza dos mesmos e da origem das entradas. A *avaliação* de circuitos analógicos é feita, em geral, através do emprego de simuladores. As três formas de análise mais comuns são: a análise AC (domínio da frequência); transferência DC e análise transitória (domínio do tempo). Uma medida do erro é calculada ao comparar-se a resposta de cada solução evoluída com a resposta desejada (ZEBULUM, 1999).

### 5.1.3 Plataforma Evolucionária

As plataformas usadas em aplicações de eletrônica evolucionária podem ser classificadas como *extrínsecas* ou *intrínsecas*, dependendo de como é realizada a avaliação das soluções evoluídas.

Nas plataformas extrínsecas, os circuitos são avaliados através de simuladores e apenas a melhor configuração obtida pelo algoritmo evolucionário é implementada fisicamente ao final do processo evolutivo. O projeto extrínseco marcou o início dos estudos na área da eletrônica evolucionária. Este método consiste basicamente no uso da simulação por ferramentas de *software* para avaliação de circuitos que estejam participando de um processo evolucionário.

As plataformas intrínsecas simulam a evolução diretamente em circuitos integrados reconfiguráveis, ou seja, o circuito é reconfigurado o mesmo número de vezes que o tamanho da população em cada geração. A metodologia intrínseca de evolução de circuitos é denominada na literatura de *Evolvable Hardware* (GARIS, 1993). A abordagem intrínseca é mais complexa que a extrínseca, pois nesta somente o melhor circuito é implementado fisicamente (MILLER *et al.*, 1999).

Os circuitos integrados reconfiguráveis mais utilizados nos experimentos em *Evolvable Hardware* são as FPGAs (*Field-Programmable Gate Array*) para projetos digitais e FPAAs (*Field-Programmable Analog Array*) para projetos analógicos, devido à rapidez na programação e a capacidade de reconfiguração.

As FPGAs são arranjos ordenados de duas dimensões de elementos lógicos, sendo que a sua estrutura exata depende do fabricante, dentre estes pode-se citar as empresas *Xilinx* e *Altera*. Cada elemento lógico pode implementar uma determinada função que usualmente consiste de uma lógica combinacional com um ou mais elementos de memória (*flip-flops*) para implementação de comportamentos seqüenciais. A complexidade e estrutura da função programável podem variar consideravelmente de um tipo de FPGA para outro. A comunicação entre os elementos é realizada através de conexões programáveis que também podem variar em função do tipo de FPGA.

As FPGAs e FPAAs são basicamente circuitos integrados que podem ser configurados, ou seja, programados por *software*, para realizarem qualquer função (ou seja, implementar qualquer circuito). Atualmente, as FPGAs representam o tipo de plataforma mais desenvolvido que encontra-se disponível para as pesquisas envolvendo eletrônica evolucionária intrínseca.

A funcionalidade e conexão de um elemento são controladas pela sua configuração. A arquitetura de um circuito integrado programável, e portanto a sua função é determinada por um conjunto de bits de configuração, os quais podem ser alterados, ou seja reconfigurados. O conjunto de bits de configuração, que consiste da soma de todas as configurações dos elementos da FPGA, determina o comportamento global do circuito integrado. Uma FPGA pode ser configurada para implementar qualquer circuito lógico digital, desde que uma quantidade suficiente de recursos esteja disponível.

Com FPGAs cada vez mais modernas, a arquitetura dos circuitos eletrônicos se tornou tão maleável quanto um *software*. Com isso, a utilização de um computador para simular a evolução do circuito eletrônico, carregando apenas o melhor indivíduo obtido para o circuito integrado programável, deixou de ser a única opção disponível. As FPGAs permitem a avaliação em tempo real da evolução dos circuitos eletrônicos, neste caso todos os circuitos evoluídos podem ser carregados na FPGA e avaliados como uma configuração real do circuito eletrônico (YAO; HIGUCHI, 1999).

Os bits de configuração da FPGA podem ser tratados como os indivíduos que formam a população usada em algoritmos evolucionários, o qual pode resolver difíceis tarefas de aprendizado e busca. Então, o indivíduo pode especificar o tipo de função das unidades de evolução e a interconexão entre as mesmas. Desta forma, pode-se pensar em um circuito integrado capaz de aprender pelo conhecimento adquirido e que possa ser reprogramado para que tenha um desempenho melhor na próxima vez (ALI, 2003).

As FPAAs são os equivalentes analógicos das FPGAs. Diferentemente das FPGAs, que contém um grande número de elementos lógicos e interconexões permitindo configurações arbitrárias de lógicas combinacionais e seqüenciais, as FPAAs tipicamente possuem um número menor de blocos analógicos configuráveis. Os recursos disponíveis em um bloco básico variam bastante em função do modelo da FPAA, as quais encontram-se disponíveis comercialmente e/ou desenvolvidas em centros de pesquisa (KEYMEULEN *et al.*, 2000). As FPAAs voltadas para os projetos analógicos normalmente possuem um amplificador operacional, arranjos de capacitores e resistores programáveis por bloco (KEYMEULEN *et al.*, 2000).

## 5.2 Características da Eletrônica Evolucionária

A eletrônica evolucionária tem sido utilizada como uma alternativa ao projeto convencional de circuitos eletrônicos. O projeto de circuitos utilizando-se técnicas evolucionárias possui alguns aspectos que diferem do projeto convencional de circuitos realizado por humanos. Alguns destes aspectos são destacados nas seis seções seguintes.

### 5.2.1 Espectro de Soluções

O projeto evolucionário pode explorar um espectro de soluções muito mais amplo que as soluções obtidas em projetos desenvolvidos por seres humanos. A eletrônica evolucionária pode evoluir soluções que seriam difíceis de serem descobertas pelos humanos (YAO; HIGUCHI, 1999). O espectro bastante abrangente das soluções possíveis é considerado um dos pontos mais favoráveis ao uso da eletrônica evolucionária.

### 5.2.2 Conhecimento do Projeto

O processo evolucionário de circuitos pode ser aplicado por usuários que possuem apenas um conhecimento básico sobre o domínio em que o projeto é requerido, podendo também ser usado nos domínios em que se dispõe de pouca informação ou nos quais a obtenção destas informações é muito custosa. Em essência, o método de projeto convencional especifica *como* se projetar e implementar um circuito, enquanto o método evolucionário apenas especifica *o que* o circuito deve implementar, ou seja, que função ou comportamento requerido o circuito deve ter sem se preocupar como alcançá-los (YAO; HIGUCHI, 1999).

### 5.2.3 Restrições do Projeto

O processo evolucionário de circuitos pode lidar com diferentes graus de restrições e requisitos especiais. Isto é feito através da incorporação dessas restrições na representação dos indivíduos e função de avaliação de aptidão. Conforme mencionado acima, o método evolucionário pode trabalhar com um pequeno conhecimento do domínio. Porém, se alguma informação do domínio estiver disponível, esta pode ser utilizada para aumentar a eficiência do processo evolucionário (YAO; HIGUCHI, 1999).

### 5.2.4 Escalabilidade

A propriedade de escalabilidade é de suma importância em qualquer tipo de projeto e em particular nos projetos de circuitos eletrônicos (HIGUCHI *et al.*, 1992) (HEMMI; MIZOGUCHI;

SHIMOHARA, 1996). Este problema é encontrado não só nas pesquisas envolvendo eletrônica evolucionária, como em outras pesquisas no campo da inteligência computacional.

Segundo (YAO; HIGUCHI, 1999), a escalabilidade na eletrônica evolucionária pode ser vista de duas maneiras relacionadas entre si. A primeira lida com a escalabilidade do indivíduo que representa o circuito eletrônico. Pode-se dizer, de maneira aproximada, que se nenhuma restrição de *conectividade* for estabelecida, ou seja, qualquer roteamento entre as unidades básicas é possível, então o tamanho do indivíduo irá crescer na ordem de  $O(n^2)$ , onde  $n$  é o número de componentes funcionais, tais como, portas lógicas. Se a conectividade ficar restrita a certo local na vizinhança ao redor do componente funcional, a ordem  $O(n)$  pode ser atingida. Porém, isto introduz uma restrição no projeto evolucionário. A segunda maneira de ver a escalabilidade em eletrônica evolucionária diz respeito a complexidade computacional do algoritmo evolucionário. Esta é a questão mais importante que a da escalabilidade da representação do indivíduo. Atualmente, é comum a realização de experimentos envolvendo a eletrônica evolucionária que necessitem de uma execução com vários dias de duração. Hoje em dia, a escalabilidade é considerada o ponto mais desfavorável ao uso da eletrônica evolucionária.

### 5.2.5 Avaliação das Soluções

Uma questão que surge no projeto evolucionário de circuitos eletrônicos é como se verificar se o circuito que foi evoluído está correto. Algumas vezes é difícil encontrar uma função de avaliação da aptidão sem que se tenha um custo computacional grande. Por exemplo, no projeto de circuitos digitais, um circuito com  $n$  entradas externas irá requerer  $2^n$  combinações possíveis das entradas quando do cálculo da funcionalidade. Portanto, a obtenção de resultados para circuitos com um grande número de entradas se torna difícil (YAO; HIGUCHI, 1999).

### 5.2.6 Compreensão e Manutenção

Outras questões que são importantes nas pesquisas envolvendo a eletrônica evolucionária incluem a *compreensão* e *manutenção* dos circuitos evoluídos (YAO; HIGUCHI, 1999). Os circuitos evoluídos, via de regra são de difícil compreensão e, portanto, de difícil manutenção para os seres humanos. Eles são basicamente caixas pretas. Se a manutenção for realizada por especialistas humanos, eles devem ser capazes de entender o circuito que foi evoluído, o que pode ser uma tarefa árdua. Neste caso, uma direção a ser seguida deverá ser o desenvolvimento de uma metodologia completamente diferente para a manutenção do circuito, na qual a própria eletrônica evolucionária poderia detectar e corrigir as suas falhas.

## 5.3 Aplicações Envolvendo a Eletrônica Evolucionária

A utilização da evolução no projeto de circuitos eletrônicos tem permitido a automação e inovação para um número crescente de aplicações. Com isso, todas as maneiras de se projetar circuitos podem potencialmente ser submetidas à evolução. A seguir são apresentados sucintamente alguns trabalhos referentes à eletrônica evolucionária. Os trabalhos citados nesta seção representam uma pequena amostra das aplicações da eletrônica evolucionária encontrada na literatura.

Em (HIGUCHI *et al.*, 1999), os autores apresentam várias aplicações da eletrônica evolucionária em problemas práticos, dos quais pode-se destacar a aplicação no ajuste dos filtros da frequência intermediária (FI) de telefones celulares. Frequentemente, os componentes dos circuitos analógicos diferem ligeiramente das suas especificações originais devido à variações durante o processo de fabricação. Foi mostrado que estas discrepâncias podem ser corrigidas usando a evolução para guiar o circuito para o comportamento desejado.

A eletrônica evolucionária em nível de funções foi investigada em (KALGANOVA, 2000). Neste trabalho, funções lógicas de múltiplas entradas e saídas são utilizadas como células lógicas em uma abordagem extrínseca da eletrônica evolucionária. O trabalho está focado no problema do projeto de circuitos lógicos combinacionais. Para os casos investigados, o método apresentou um desempenho melhor que o apresentado pela evolução em nível de portas lógicas em termos da quantidade de portas primitivas utilizadas na configuração final dos circuitos.

Em (NEDJAH; MOURELLE, 2005) a eletrônica evolucionária extrínseca foi aplicada na geração automática de projetos de circuitos digitais mínimos e seguros de sistemas criptográficos de chave pública, como o sistema de criptografia RSA. Neste trabalho foi utilizada a programação genética na evolução dos circuitos e foi empregada a representação em nível de portas lógicas. Os resultados obtidos foram superiores aos obtidos por projetos convencionais.

Em (KOZA; KEANE; STREETER, 2005) foram obtidos bons resultados no projeto evolucionário de circuitos analógicos que possuem patentes do século XXI, como por exemplo, circuitos conversores de voltagem-corrente, filtros ativos e circuitos geradores de sinais cúbicos. O trabalho apresenta os circuitos que foram evoluídos através da programação genética, a qual empregou um simulador de circuito comercial para a análise dos circuitos candidatos. Os circuitos obtidos apresentam funcionalidades superiores quando comparados com os circuitos que foram patenteados para desempenhar a função.

Em (AMARAL *et al.*, 2005) é discutido o uso de uma plataforma reconfigurável para a exploração da evolução intrínseca de circuitos analógicos para Controladores Lógicos *Fuzzy*. Neste

trabalho os autores utilizam uma plataforma chamada de PAMA-NG (*Programmable Analog Multiplexer Array - Next Generation*) que é classificada como uma FPAA (*Field Programmable Analog Array*). Uma característica importante da PAMA-NG perante as demais FPAAs diz respeito a sua proteção intrínseca quanto às configurações ilegais que poderiam danificar os componentes eletrônicos. A PAMA-NG é composta de componentes discretos que são interconectados através de multiplexadores/demultiplexadores. A evolução intrínseca do circuito, que é guiada por um algoritmo genético, é realizada através da programação das conexões internas da plataforma. Apesar da plataforma utilizada ainda não permitir a implementação de um controlador lógico *fuzzy* analógico completo, a evolução de funções de pertinência e circuitos *fuzzy* foram bem sucedidas.

Os algoritmos genéticos e a programação genética, descritos no Capítulo 2, são normalmente aplicados na simulação da evolução dos circuitos eletrônicos. Na Seção 5.4, a aplicação dos algoritmos evolucionários inspirados na computação quântica à síntese de máquinas de estados finitos é investigada.

## 5.4 AEICQ Aplicado na Eletrônica Evolucionária

Nesta seção será apresentada uma nova ferramenta para a síntese da lógica de controle de máquinas de estados finitos utilizando a evolução inspirada nos princípios da computação quântica. Nesta ferramenta, o Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ) é usado para evoluir lógicas de controle otimizadas para os sistemas seqüenciais síncronos. Isto é realizado para uma atribuição de estados pré-determinada.

### 5.4.1 Metodologia

A metodologia proposta para a síntese evolucionária de máquinas de estados finitos é mostrada na Figura 42. Na primeira e segunda etapa, as quais não são tratadas neste trabalho, é definida a especificação da máquina de estados utilizando-se estados simbólicos, assim como a minimização dos estados da máquina, quando necessário. Na terceira etapa, o AEICQ é utilizado para a obtenção de uma atribuição de estados otimizada para a máquina de estados, conforme foi descrito no Capítulo 4, e em seguida, este mesmo algoritmo é utilizado na síntese da lógica de controle da máquina de estados.

Conforme descrito no Capítulo 1, um sistema seqüencial consiste de uma *via de dados* e um *controlador*. O controlador é uma máquina de estados finitos que implementa o controle requerido através de uma lógica combinacional. A via de dados pode incluir registradores,

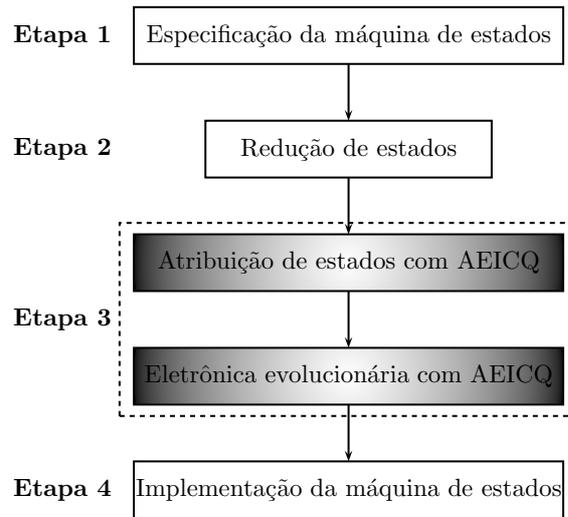


Figura 42: Metodologia proposta para a síntese evolucionária de máquinas de estados

contadores, multiplexadores e barramento. O controlador implementa a máquina de estados responsável pela seqüência das etapas do sistema seqüencial em desenvolvimento. Com isso, o processo de síntese da lógica de controle das máquinas de estados finitos pode ser tratado da mesma forma que os circuitos lógicos combinacionais, ou seja, a funcionalidade desejada para a lógica de controle pode ser avaliada utilizando-se uma tabela verdade.

A estrutura básica do AEICQ aplicado na síntese de máquinas de estados finitos é a mesma apresentada no Algoritmo 5 do Capítulo 4 e que foi aplicado no problema da atribuição de estados. As diferenças encontram-se na codificação das soluções e na função de avaliação.

### 5.4.2 Codificação

O AEICQ aplicado na síntese de máquinas de estados finitos codifica os circuitos através de uma matriz de células. A estrutura genérica dessa matriz é mostrada na Figura 43. Cada

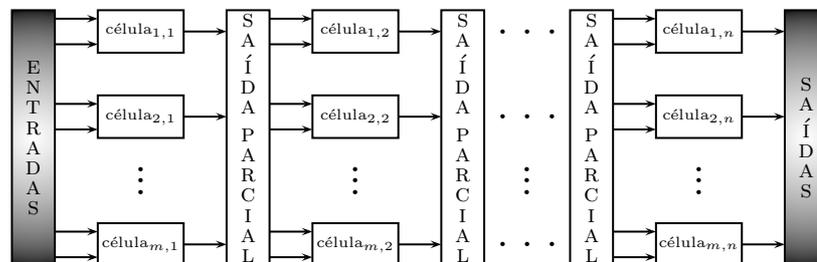


Figura 43: Exemplo de uma matriz de células

célula da matriz é constituída de duas entradas  $A$  e  $B$ , uma porta lógica e uma saída. Os sinais de entradas das células localizadas na primeira coluna da matriz são obtidos diretamente

|            | Porta ( $p$ ) |            |            | Entrada A ( $e_a$ ) |            |            |            | Entrada B ( $e_b$ ) |            |               |               |
|------------|---------------|------------|------------|---------------------|------------|------------|------------|---------------------|------------|---------------|---------------|
| Célula     | $\alpha_1$    | $\alpha_2$ | $\alpha_3$ | $\alpha_4$          | $\alpha_5$ | $\alpha_6$ | $\alpha_7$ | $\alpha_8$          | $\alpha_9$ | $\alpha_{10}$ | $\alpha_{11}$ |
|            | $\beta_1$     | $\beta_2$  | $\beta_3$  | $\beta_4$           | $\beta_5$  | $\beta_6$  | $\beta_7$  | $\beta_8$           | $\beta_9$  | $\beta_{10}$  | $\beta_{11}$  |
| Observação | 0             | 0          | 1          | 0                   | 0          | 1          | 0          | 1                   | 0          | 1             | 0             |

Figura 44: Exemplo da codificação de uma célula com 3 q-bits para a seleção das portas e 4 q-bits para a seleção dos sinais de cada entrada

das entradas da máquina de estados. Os sinais de entrada das células das demais colunas são obtidos das saídas das células da coluna imediatamente anterior à coluna onde encontra-se a célula em questão. Cada célula é codificada com uma quantidade total de q-bits que é suficiente para codificar todas as portas lógicas disponíveis e também todas as saídas das células da coluna anterior.

A Figura 44 mostra o exemplo da codificação de uma célula que possui 3 q-bits para a seleção das portas e 4 q-bits para a seleção do sinal de cada uma das entradas. Neste caso, pode-se concluir que o algoritmo faz a seleção entre 8 tipos possíveis de portas ( $2^3$  possibilidades) e que a coluna anterior possui no máximo 16 saídas ( $2^4$  possibilidades). Quando a quantidade de saídas da coluna anterior não é uma potência de 2, algumas destas saídas são repetidas com o objetivo de evitar o uso de estruturas de dados dinâmicas que acabariam deteriorando o desempenho do AEICQ.

Uma possível observação dos estados dos q-bits é mostrada na última linha da Figura 44, onde os códigos obtidos definem a porta lógica e as células da coluna anterior de onde serão obtidos os sinais para as entradas da porta lógica observada. Pode-se observar também que o número de q-bits necessários para codificar cada uma das células varia em função do número de portas disponíveis para a seleção pelo algoritmo e do número de células da coluna anterior. No caso das células localizadas na primeira coluna da matriz, o número de q-bits necessários para a codificação dependem do número de portas disponíveis para a seleção pelo algoritmo e do número de entradas da máquina de estados.

As portas lógicas utilizadas pelo AEICQ durante o processo evolucionário, assim como seus respectivos códigos são mostradas na Tabela 16. Além das portas lógicas, também foi incluída uma possibilidade que é a ausência de porta lógica na célula e que pode também vista como um tipo de *buffer*. Esta possibilidade, indicada como *fluo*, quando selecionada, faz apenas uma conexão do sinal presente na entrada A da célula com a sua saída. Esta opção permite que um sinal de saída de uma célula que não pertença à coluna imediatamente anterior possa ser usado.

Tabela 16: Portas utilizadas pelo AEICQ e seus respectivos códigos

| Porta | Código |
|-------|--------|
| NOT   | 000    |
| AND   | 001    |
| OR    | 010    |
| XOR   | 011    |
| NAND  | 100    |
| NOR   | 101    |
| XNOR  | 110    |
| FIO   | 111    |

A Figura 45 mostra o circuito que seria obtido considerando a observação indicada na Figura 44 e a codificação das portas indicadas na Tabela 16. Este circuito é composto de uma porta AND com suas entradas  $A$  e  $B$  conectadas nas saídas da segunda ( $0010_2 = 2_{10}$ ) e da décima ( $1010_2 = 10_{10}$  na base decimal) célula da coluna anterior, respectivamente.

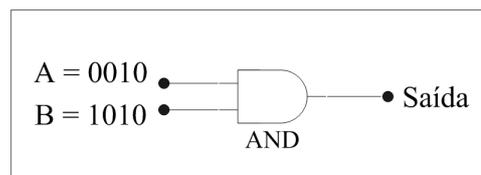


Figura 45: Exemplo do circuito codificado por uma célula

Os sinais de saída do circuito evoluído pelo algoritmo são os sinais de saída da última coluna de células. Quando o número de saídas do circuito é menor do que o número de células existente nessa coluna, então as últimas células são ignoradas durante o processo evolucionário.

O poder da representação inspirada na computação quântica pode ser evidenciada pela Figura 46, onde é mostrado que todos os possíveis circuitos podem ser representados por um único  $q$ -indivíduo de uma maneira probabilística, conforme explicado no Capítulo 3.

O número de células por  $q$ -indivíduo é uma parâmetro do algoritmo, o qual deve ser ajustado considerando-se a complexidade da máquina de estados. Esta complexidade depende do número de entradas e saídas primárias, do número de estados e transições de estados.

### 5.4.3 Avaliação da Aptidão

Um aspecto importante na evolução de circuitos é a forma utilizada na avaliação das soluções. Para avaliar a aptidão de uma solução, os seguintes critérios foram considerados neste trabalho:

- O circuito evoluído deve ser totalmente funcional, ou seja, deve atender o comportamento exigido para as entradas/saídas.

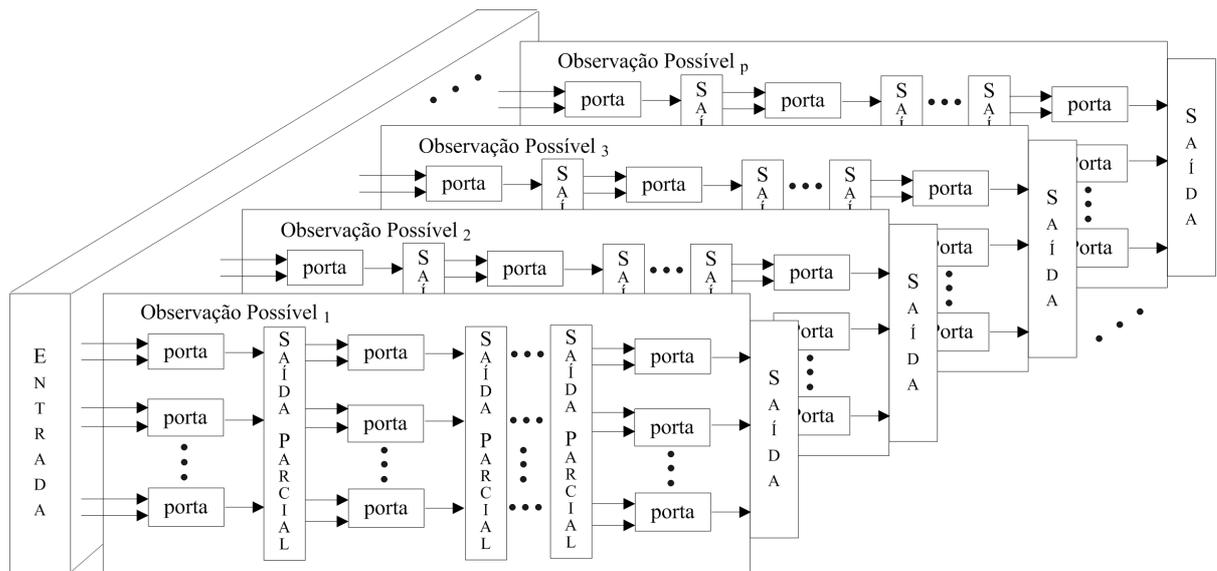


Figura 46: Poder de representação do AEICQ

- O circuito deve ser otimizado com relação à área ocupada e ao atraso de propagação dos sinais de saídas.

A avaliação de um circuito quanto à funcionalidade é feita através da tabela de transições de estados, ou seja, as saídas do circuito evoluído devem coincidir com as saídas da tabela de transições de estados. Portanto, cada circuito obtido pelo AEICQ durante o processo evolucionário deve ser avaliado para todas as combinações dos sinais de entrada presentes na tabela de transições de estados.

Quanto à otimização, qualquer tipo de parâmetro do circuito pode ser utilizado na definição da qualidade do circuito evoluído. A área ocupada pelo circuito e o atraso de propagação dos sinais são utilizados como parâmetros que devem ser otimizados pelo AEICQ. A otimização da área ocupada garante uma implementação mais compacta de circuitos, enquanto a otimização do atraso de propagação resulta em uma implementação mais eficiente destes.

O processamento de múltiplos objetivos pelos algoritmos evolucionários pode ser classificado em três categorias dependendo da maneira como os indivíduos são avaliados e selecionados (FONSECA; FLEMING, 1995), (DEB, 2001).

- *Seleção por Critério:* Os métodos classificados sob este esquema avaliam a aptidão de cada solução evoluída utilizando as funções objetivo separadamente. Para um problema com  $n$  objetivos, cada indivíduo carrega consigo  $n$  valores de aptidão, um para cada critério.

- *Seleção por Pareto*: Os métodos sob este esquema avaliam a aptidão de cada solução usando a relação de dominância. As soluções não dominadas com respeito aos indivíduos da população atual são consideradas as mais aptas. Uma solução  $S_1$  *domina* outra solução  $S_2$  se e somente se  $S_1$  não é pior que  $S_2$  com relação a todos os objetivos e se  $S_1$  é estritamente melhor que  $S_2$  em pelo menos um objetivo. De outra forma, a solução  $S_1$  não domina a solução  $S_2$  (PARETO, 1896).
- *Seleção por Agregação dos Objetivos*: Os métodos sob este esquema combinam as  $n$  funções objetivo em uma única função objetivo usando operadores de agregação. A principal característica destes métodos é que eles reduzem os multi-objetivos para apenas um único. Um dos métodos de agregação dos objetivos é a *soma ponderada*. Este método agrega os vários objetivos linearmente. Para este propósito, o método utiliza pesos para ponderar as funções objetivo (FONSECA; FLEMING, 1995), permitindo assim priorizar um ou mais objetivos sobre outros, isto é:

$$Aptidão(x) = \sum_{i=1}^n \omega_i f_i(x) \quad (35)$$

onde  $f_i(x)$  representa a aptidão do indivíduo  $x$  em relação ao objetivo  $i$  e  $\omega_i$  é o respectivo peso, para um total de  $n$  objetivos.

O método da soma ponderada é utilizado pelo AEICQ para a otimização da área e do tempo de propagação dos sinais de saída do circuito. Seja  $C$  um circuito digital que utiliza um subconjunto ou o conjunto completo das portas lógicas indicadas na Tabela 16, considerando a restrição quanto à funcionalidade do circuito e a otimização da sua área e do seu atraso de propagação, pode-se definir a função de avaliação da aptidão, como segue

$$Aptidão(C) = Precisão(C) + \omega_1 \times Área(C) + \omega_2 \times Atraso(C), \quad (36)$$

onde o objetivo do AEICQ é a minimização desta função.

A *Precisão(C)* é uma função que usa a distância de Hamming para avaliar a funcionalidade do circuito  $C$  com relação ao comportamento desejados das entradas/saídas. Esta função retorna um valor proporcional a quantidade de erros resultantes da comparação entre os sinais de saída do circuito evoluído e os sinais de saída esperados para cada uma das combinações dos sinais de entrada. A função pode ser representada da seguinte forma

$$Precisão(C) = \sum_{j=1}^p |y_j - x_j| \times \psi \quad (37)$$

onde  $p$  é o número de combinações possíveis das entradas,  $|y_j - x_j|$  é a diferença absoluta entre os sinais de saída do circuito evoluído e os sinais de saída esperados,  $x_j$  e  $y_j$  respectivamente. Os argumentos  $x_j$  e  $y_j$  são vetores representando a  $j$ -ésima saída do circuito e  $\psi$  é o valor constante da penalidade referente a um único erro.

Observe que os circuitos  $C$  com  $Precisão(C)$  positiva não implementam a funcionalidade desejada corretamente e portanto, este parâmetro é considerado como uma *penalidade* dos indivíduos que codificam tais circuitos.

Nos experimentos realizados optou-se por privilegiar a otimização da área ocupada pelo circuito evoluído. Os coeficientes de peso  $\omega_1$  e  $\omega_2$  da Equação 36 foram ajustados ambos com o valor 0,5. Tendo em vista que os valores retornados pela função  $\acute{A}rea(C)$  são de ordem de grandeza maior que os valores retornados pela função  $Atraso(C)$  como será visto a diante, a função objetivo naturalmente privilegia a área com relação ao atraso de propagação.

A função  $\acute{A}rea(C)$  retorna a área necessária para a implementação do circuito  $C$ , a qual é estimada através do conceito de *portas equivalentes*. Esta é uma unidade básica para a medida da complexidade de um circuito (ERCEGOVAC; LANG; MORENO, 1999). Esta medição é baseada no número de portas lógicas que devem ser interconectadas para desempenharem o mesmo comportamento de entrada/saída, sendo mais precisa que a simples quantidade de portas, já que é relacionada ao número de transistores empregados no projeto da porta (ERCEGOVAC; LANG; MORENO, 1999).

Considere um circuito cuja geometria é representada por uma matriz  $C : n \times m$ . Vale lembrar que cada célula  $C_{i,j}$  do circuito é formada pelo tipo da porta  $p$ , juntamente com as duas entradas  $e_a$  e  $e_b$ . A função  $\acute{A}rea(C)$  é definida na Equação 38. Esta definição é elaborada com o auxílio da função recursiva  $\acute{A}rea(C)_{i,j}$  que permite calcular a área ocupada pela parte do circuito  $C$  que produz a saída da porta encontrada na célula  $C_{i,j}$ . Esta função é definida na Equação 39. Note que a área correspondente às portas compartilhadas só podem ser computadas uma única vez. Para isso, é usada uma matriz Booleana  $V : n \times m$  cujas entradas são atualizadas ao passo que a célula correspondente na representação do circuito é visitada.

$$\acute{A}rea(C) = \sum_{i=1}^s \acute{A}rea_{i,m} \quad (38)$$

onde  $s$  é o número de sinais de saída do circuito, com  $s \leq m$ .

$$\text{Área}_{i,j} = \begin{cases} PE_{C_{i,j}^p} & \text{Se } j = 1 \\ PE_{C_{i,j}^p} + \sum_{x \in \{a,b\} e^{-V_{C_{i,j}^{e_x}, j-1}}} (\text{Área}_{C_{i,j}^{e_x}, j-1}) & \text{Se } j \in [2, m] \end{cases} \quad (39)$$

onde  $PE_{C_{i,j}^p}$  representa o número de portas equivalentes para a porta  $p$  encontrada na célula  $C_{i,j}$  do circuito e  $C_{i,j}^{e_x}$  representa uma das entradas da porta presente na célula  $C_{i,j}$ . O AEICQ considera os valores de área ocupada pelas portas lógicas da família CMOS. Estes valores estão indicados na última coluna da Tabela 17 e encontram-se disponíveis em (ERCEGOVAC; LANG; MORENO, 1999).

Para ilustrar o cômputo de área, conforme indicado na Equação 38, é considerado o circuito da Figura 47 e o cálculo em seguida. A área total ocupada pelo circuito da Figura 47 é então a soma  $10 + 8 + 1 + 6 = 25$  portas equivalentes, conforme valores indicados na última coluna da matriz  $\text{Área}_{(-,3)}$ . Observe que através deste cálculo de área, todas as células que não participam efetivamente da formação do circuito são ignoradas.

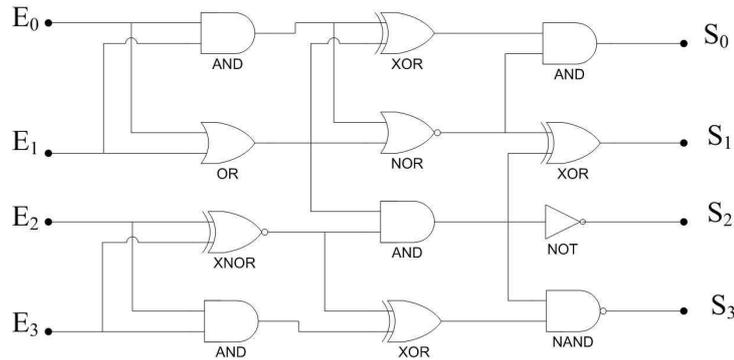


Figura 47: Exemplo de um circuito evoluído

$$\text{Área}_{(-,1)} = \begin{bmatrix} 2 & - & - \\ 2 & - & - \\ 3 & - & - \\ 2 & - & - \end{bmatrix}, \quad \text{Área}_{(-,2)} = \begin{bmatrix} 2 & 7 & - \\ 2 & 1 & - \\ 3 & 5 & - \\ 2 & 5 & - \end{bmatrix} \text{ e } \text{Área}_{(-,3)} = \begin{bmatrix} 2 & 7 & 10 \\ 2 & 1 & 8 \\ 3 & 5 & 1 \\ 2 & 5 & 6 \end{bmatrix}$$

O atraso de propagação de um circuito é definido pelo atraso do *caminho crítico*. Considerando todos os caminhos de um circuito, o caminho crítico é aquele que proporciona o maior atraso.

O atraso de propagação de um caminho em um circuito é definido por meio da soma dos atrasos proporcionados por cada porta lógica percorrida pelo sinal desde a entrada até alcançar a saída do circuito.

Tabela 17: Portas utilizadas pelo AEICQ e seus respectivos valores de atraso de propagação, fator de carga e área

| Tipo de Porta | Atraso de Propagação |                 | Fator de Carga<br>(carga-padrão) | Área<br>(porta-equivalente) |
|---------------|----------------------|-----------------|----------------------------------|-----------------------------|
|               | $t_{pLH}$ (ns)       | $t_{pHL}$ (ns)  |                                  |                             |
| NOT           | $0,02 + 0,038L$      | $0,05 + 0,017L$ | 1,0                              | 1                           |
| AND           | $0,15 + 0,037L$      | $0,16 + 0,017L$ | 1,0                              | 2                           |
| OR            | $0,12 + 0,037L$      | $0,20 + 0,019L$ | 1,0                              | 2                           |
| XOR           | $0,30 + 0,036L$      | $0,30 + 0,021L$ | 1,1                              | 3                           |
| NAND          | $0,05 + 0,038L$      | $0,08 + 0,027L$ | 1,0                              | 1                           |
| NOR           | $0,06 + 0,075L$      | $0,07 + 0,016L$ | 1,0                              | 1                           |
| XNOR          | $0,30 + 0,036L$      | $0,30 + 0,021L$ | 1,1                              | 3                           |

Quando a entrada de uma determinada porta é alterada de 0 para 1 ou de 1 para 0, existe um tempo de atraso finito antes que esta alteração seja percebida no terminal de saída dessa porta. Este tempo é chamado de *atraso de propagação* da porta e depende do tipo de porta lógica, da tecnologia utilizada na construção da porta lógica e da carga colocada na saída da porta. O atraso de propagação de uma porta lógica também é função do tipo de transição do sinal, ou seja, os atrasos de propagação são diferentes quando ocorre uma transição de baixo para alto ( $t_{pLH}$ ) ou de alto para baixo ( $t_{pHL}$ ). Os valores de atraso de propagação utilizados referentes às portas da família CMOS e que encontram-se disponíveis em (ERCEGOVAC; LANG; MORENO, 1999) são mostrados na Tabela 17, onde  $L$  representa a carga total da saída. Os valores médios dos atrasos de cada porta mostrados na Tabela 18 são utilizados na estimativa do atraso de propagação dos circuitos evoluídos pelo AEICQ.

O atraso de cada porta lógica é função da *carga total* de saída. Para o cálculo da carga total de uma porta, uma unidade de *carga-padrão* é definida para a família. Esta carga-padrão normalmente corresponde à carga de entrada de uma porta básica como, por exemplo, a porta NOT. A carga das entradas de cada porta é dada em termos desta unidade. Esta medida é chamada de *fator de carga* da entrada. A carga total de uma saída é calculada como a soma dos fatores de carga de todas as entradas conectadas nesta saída.

Considerando um circuito cuja geometria é representada por uma matriz  $C : n \times m$ . Vale lembrar novamente que cada célula  $C_{i,j}$  do circuito é formada pelo tipo da porta  $p$ , juntamente com as duas entradas  $e_a$  e  $e_b$  e a saída é identificada por  $i$ . O atraso ocasionado pela porta da célula  $C_{i,j}$  é definido conforme apresentado na Equação 40, tal que

$$\tau_{porta_{i,j}} = \alpha_{C_{i,j}^p} + \beta_{C_{i,j}^p} \times \left( \sum_{\substack{k \in [1, n], x \in \{a, b\} \\ C_{k,j+1}^{e_x} = i}} fator(C_{k,j+1}^p) \right), \quad (40)$$

onde  $\alpha_{C_{i,j}^p}$  representa a média do atraso intrínseco da porta  $p$  presente na célula  $C_{i,j}$ , ou seja, o atraso médio da porta quando a carga total é nula e  $\beta_{C_{i,j}^p}$  representa o atraso médio devido a um *fanout* da saída da porta  $p$  da mesma célula. Como antes,  $C_{i,j}^{e_x}$  identifica uma das entradas da porta que se encontra na célula  $C_{i,j}$ . A Tabela 18 mostra os valores de  $\alpha$  e  $\beta$  de cada uma das portas utilizadas pelo AEICQ.

Tabela 18: Valores de  $\alpha$  e  $\beta$  de cada uma das portas lógicas utilizadas pelo AEICQ

| Tipo de Porta | $\alpha$ | $\beta$ |
|---------------|----------|---------|
| NOT           | 0,035    | 0,0465  |
| AND           | 0,155    | 0,0270  |
| OR            | 0,160    | 0,0280  |
| XOR           | 0,300    | 0,0285  |
| NAND          | 0,065    | 0,0325  |
| NOR           | 0,065    | 0,0455  |
| XNOR          | 0,300    | 0,0285  |

Considerando novamente o circuito da Figura 47. O atraso de cada uma das portas, calculado segundo a Equação 40, é detalhado conforme apresentado a seguir:

$$\tau_{porta} = \begin{bmatrix} 0,155 & 0,300 & 0,155 \\ 0,160 & 0,065 & 0,300 \\ 0,300 & 0,155 & 0,035 \\ 0,155 & 0,300 & 0,065 \end{bmatrix} + \begin{bmatrix} 0,0270 \times (1,1 + 1,0) & 0,0285 \times 1,0 & 0,0270 \times 1,0 \\ 0,0280 \times (1,1 + 1,0 + 1,0) & 0,0455 \times (1,0 + 1,1) & 0,0285 \times 1,0 \\ 0,0285 \times (1,0 + 1,1) & 0,0270 \times (1,1 + 1,0 + 1,0) & 0,0275 \times 1,0 \\ 0,0270 \times 1,1 & 0,0285 \times 1,0 & 0,0325 \times 1,0 \end{bmatrix}$$

$$\tau_{porta} = \begin{bmatrix} 0,21170 & 0,32850 & 0,18200 \\ 0,24680 & 0,16055 & 0,32850 \\ 0,35985 & 0,23870 & 0,06250 \\ 0,18470 & 0,32850 & 0,09750 \end{bmatrix}.$$

A determinação do atraso de propagação de cada caminho é realizada por meio de uma função recursiva definida conforme a Equação 41.

$$\tau_{caminho_{i,j}} = \begin{cases} \tau_{porta_{i,j}} & \text{Se } j = 1 \\ \tau_{porta_{i,j}} + \max_{x \in \{a,b\}} (\tau_{caminho_{C_{i,j}^{ex}, j-1}}) & \text{Se } j \in [2, m] \end{cases} \quad (41)$$

O exemplo a seguir ilustra o procedimento utilizado para o cálculo do atraso de propagação dos caminhos do circuito da Figura 47.

$$\tau_{caminho_{(-,1)}} = \begin{bmatrix} 0, 21170 & - & - \\ 0, 24680 & - & - \\ 0, 35985 & - & - \\ 0, 18470 & - & - \end{bmatrix}$$

$$\tau_{caminho_{(-,2)}} = \begin{bmatrix} 0, 21170 & 0, 32850 + \max(0, 21170, 0, 24680) & - \\ 0, 24680 & 0, 16055 + \max(0, 21170, 0, 24680) & - \\ 0, 35985 & 0, 23870 + \max(0, 35985, 0, 18470) & - \\ 0, 18470 & 0, 32850 + \max(0, 35985, 0, 18470) & - \end{bmatrix}$$

$$\tau_{caminho_{(-,3)}} = \begin{bmatrix} 0, 21170 & 0, 57530 & 0, 18200 + \max(0, 57530, 0, 40735) \\ 0, 24680 & 0, 40735 & 0, 32850 + \max(0, 57530, 0, 40735) \\ 0, 35985 & 0, 59855 & 0, 06250 + \max(0, 59855) \\ 0, 18470 & 0, 68835 & 0, 09750 + \max(0, 59855, 0, 68835) \end{bmatrix}$$

$$\tau_{caminho} = \begin{bmatrix} 0, 21170 & 0, 5758 & 0, 75730 \\ 0, 24680 & 0, 4038 & 0, 92705 \\ 0, 35985 & 0, 5969 & 0, 66105 \\ 0, 18470 & 0, 6899 & 0, 78585 \end{bmatrix}$$

Considerando um circuito com  $s \leq n$  saídas, o atraso de propagação do circuito é determinado através da identificação do maior atraso de todos os caminhos que levam até as  $s$  portas que se encontram na última coluna da matriz que representa o circuito. A função  $Atraso(C)$  é então definida conforme a Equação 42.

$$Atraso(C) = \max_{i \in [1, s]} \tau_{caminho_{i,m}} \quad (42)$$

Para completar o exemplo, o cálculo do atraso de propagação dos sinais de saída do circuito apresentado na Figura 47 é mostrado a seguir

$$Atraso(C) = \max(0, 75730/0, 92705/0, 66105/0, 78585) = 0, 92705 \text{ ns.}$$

## 5.5 Resultados Experimentais

Nesta seção, são apresentados os resultados da metodologia proposta na seção anterior para a síntese da lógica de controle de máquinas de estados finitos. Os circuitos evoluídos pelo

AEICQ são comparados com os circuitos obtidos por outros métodos em termos de quantidade de portas, área ocupada e atraso de propagação.

### 5.5.1 Comparação dos Resultados

Para a realização dos experimentos foram selecionadas algumas máquinas de estados disponíveis em (ACM/SIGDA, 1989). Estas máquinas são referências (*benchmarks*) para testes envolvendo projetos de circuitos seqüenciais. As atribuições de estados utilizadas nos experimentos descritos nesta seção são as melhores atribuições de estado obtidas pelo AEICQ segundo as heurísticas de (ARMSTRONG, 1962) e (HUMPHREY, 1958). A Tabela 19 apresenta estas atribuições. A especificação completa de cada uma destas máquinas de estados encontra-se disponível no Apêndice A.

Tabela 19: Atribuições de estados utilizadas para a síntese das lógicas de controle

| Máquina de Estados | Atribuição de Estados                 |
|--------------------|---------------------------------------|
| <i>bbara</i>       | [4,5,1,9,13,12,14,15,7,6]             |
| <i>bbtas</i>       | [3,7,1,5,4,6]                         |
| <i>dk15</i>        | [0,2,3,1]                             |
| <i>dk27</i>        | [2,0,1,6,5,7,4]                       |
| <i>dk512</i>       | [6,4,15,14,2,10,8,3,5,11,1,0,13,9,12] |
| <i>lion9</i>       | [11,9,3,1,2,0,8,10,14]                |
| <i>modulo12</i>    | [15,7,6,14,10,2,3,1,5,13,9,11]        |
| <i>shiftreg</i>    | [6,7,4,5,2,3,0,1]                     |
| <i>tav</i>         | [1,0,3,2]                             |
| <i>train11</i>     | [9,11,13,3,1,2,0,12,8,5,4]            |

Após a implementação do AEICQ para a aplicação na evolução de circuitos, a tarefa seguinte consistiu-se do ajuste dos seus parâmetros. As máquinas *lion9* e *train11* foram utilizadas nos testes realizados para o ajustes destes parâmetros.

A população foi ajustada em 200  $q$ -indivíduos. Durante os testes foi observado que as soluções obtidas com uma população com poucos  $q$ -indivíduos não são satisfatórias, enquanto a utilização de populações muito grandes aumenta sensivelmente o tempo de simulação. Portanto, um compromisso entre o tamanho da população e o tempo de simulação deve sempre ser observado.

Para o ajuste dos valores dos ângulos utilizados na etapa de atualização dos  $q$ -bits foram testados diversos valores entre  $0,001\pi$  e  $0,1\pi$  para os ângulos  $\theta_3$  e  $\theta_5$ , conforme recomendado em (HAN, 2003). Os melhores resultados foram obtidos dentro de uma faixa variando de  $0,003\pi$  a  $0,005\pi$ . O valor médio desta faixa foi então adotado, ou seja,  $\theta_3$  e  $\theta_5$  foram ajustados em  $0,004\pi$  e  $-0,004\pi$ , respectivamente.

O parâmetro  $G$ , que representa o número de gerações sem atualização da melhor solução, e que é utilizado na operação de migração global foi testado com valores que variaram de 50 a 1000 gerações. Os melhores resultados foram obtidos para uma faixa com variação de 700 a 900, sendo então fixado em 800 para a realização dos experimentos. Para as máquinas de estados utilizadas nestes experimentos, o número máximo de gerações foi fixado em 5000.

Os ajustes utilizados para os parâmetros do AEICQ para a evolução da lógica de controle das máquinas de estados *lion9* e *train11* foram então estendidos para a síntese da lógica de controle das demais máquinas selecionadas para os experimentos.

A área de busca nas aplicações relacionadas à síntese evolucionária de circuitos eletrônicos é muito grande. Por exemplo, quando a geometria  $8 \times 4$  é utilizada para uma matriz de células que codifica um dado circuito com 8 entradas, são necessários 288 q-bits, o que define uma área de busca igual a  $2^{288}$ . Para um circuito também com 8 entradas, onde devido a sua maior complexidade se faz necessária a utilização de uma geometria igual a  $32 \times 4$ , são necessários 1536 q-bits, definindo assim uma área de busca igual a  $2^{1536}$ . Por isso, para cada uma das máquinas de estados buscou-se a identificação da menor geometria suficiente para a síntese de cada circuito. A geometria utilizada para cada uma das máquinas de estado que fazem parte destes experimentos, assim como as suas especificações e as equações das lógicas de controle obtidas pelo AEICQ encontram-se descritas no Apêndice B.

A Tabela 20 mostra um resumo das características dos circuitos que foram sintetizados utilizando-se a programação genética (PG) (NEDJAH; MOURELLE, 2005a), o algoritmo genético ( $AG_3$ ) (ALI, 2003) e a ferramenta ABC (ABC, 2005). Em (NEDJAH; MOURELLE, 2005a) os circuitos foram evoluídos em nível de portas lógicas e otimizados em termos de área consumida e atraso de propagação, enquanto em (ALI, 2003) os circuitos foram evoluídos em nível de funções e otimizados com relação a quantidade de portas. O ABC é uma ferramenta que utiliza uma metodologia determinística baseada em heurísticas, as quais tentam conduzir a implementações de circuitos otimizados com relação à área e ao atraso de propagação (MISHCHENKO; CHATTERJEE; BRAYTON, 2006). Para a obtenção dos resultados apresentados neste trabalho, o ABC foi configurado para trabalhar com as mesmas portas lógicas básicas utilizadas pelo AEICQ, assim como com os mesmos valores de portas equivalentes e atraso de propagação indicados na Tabela 17.

A Tabela 21 mostra as características dos melhores circuitos evoluídos pelo AEICQ para cada uma das máquinas de estados finitos. A coluna  $atraso_1$  lista os valores dos atrasos de propagação dos circuitos sem considerar o impacto da carga das portas lógicas. Estes valores

Tabela 20: Características dos circuitos gerados por PG, AG<sub>3</sub> e ABC

| Máquina de Estados | PG      |      |        | AG <sub>3</sub> |      |        | ABC     |      |        |
|--------------------|---------|------|--------|-----------------|------|--------|---------|------|--------|
|                    | #portas | área | atraso | #portas         | área | atraso | #portas | área | atraso |
| <i>bbara</i>       | –       | –    | –      | 60              | –    | –      | 62      | 63   | 0,67   |
| <i>bbtas</i>       | –       | –    | –      | 19              | –    | –      | 24      | 24   | 0,32   |
| <i>dk15</i>        | –       | –    | –      | 53              | –    | –      | 92      | 92   | 0,46   |
| <i>dk27</i>        | –       | –    | –      | 16              | –    | –      | 25      | 25   | 0,32   |
| <i>dk512</i>       | –       | –    | –      | 47              | –    | –      | 63      | 63   | 0,46   |
| <i>lion9</i>       | 21      | 39   | 0,70   | 50              | –    | –      | 62      | 63   | 0,53   |
| <i>modulo12</i>    | –       | –    | –      | –               | –    | –      | 38      | 38   | 0,42   |
| <i>shiftreg</i>    | 5       | 14   | 0,60   | 8               | –    | –      | 2       | 6    | 0,30   |
| <i>tav</i>         | –       | –    | –      | 26              | –    | –      | 31      | 31   | 0,46   |
| <i>train11</i>     | 22      | 43   | 0,56   | –               | –    | –      | 85      | 85   | 0,53   |

permitted uma comparação com os valores gerados pelo ABC e PG, os quais não consideram o efeito da carga das portas lógicas. Os resultados apresentados na Tabela 20 e Tabela 21 são mostrados na forma de histogramas na Figura 48 para comparação do número de portas, na Figura 49 para comparação da área e na Figura 50 para comparação do atraso de propagação.

Os valores marcados em negrito na Tabela 21 indicam os casos em que o AEICQ obteve resultados iguais ou superiores aos demais métodos. A última coluna da Tabela 21 indica os atrasos de propagação ( $atraso_2$ ) obtidos pelo AEICQ considerando o efeito da carga das portas lógicas.

Tabela 21: Resultados experimentais do AEICQ

| Máquina de Estados | #portas   | área      | $atraso_1$  | $atraso_2$ |
|--------------------|-----------|-----------|-------------|------------|
| <i>bbara</i>       | <b>54</b> | 78        | 0,88        | 1,22       |
| <i>bbtas</i>       | 21        | 27        | 0,73        | 1,14       |
| <i>dk15</i>        | 65        | 109       | 0,92        | 1,30       |
| <i>dk27</i>        | <b>15</b> | 26        | 0,43        | 0,56       |
| <i>dk512</i>       | <b>47</b> | 78        | 0,84        | 1,29       |
| <i>lion9</i>       | <b>20</b> | <b>29</b> | <b>0,52</b> | 0,81       |
| <i>modulo12</i>    | <b>19</b> | <b>34</b> | 0,56        | 0,82       |
| <i>shiftreg</i>    | <b>2</b>  | <b>2</b>  | <b>0,04</b> | 0,04       |
| <i>tav</i>         | <b>26</b> | <b>24</b> | <b>0,32</b> | 0,64       |
| <i>train11</i>     | 25        | <b>37</b> | <b>0,52</b> | 0,79       |

Para determinar se os resultados obtidos pelo AEICQ são significativos, foi aplicado o teste  $\chi^2$  de confiança. Mais detalhes sobre os teste  $\chi^2$  encontram-se na Seção 4.3. Os valores de  $\chi^2$  para as comparações com PG e ABC são apresentados na Tabela 22. Como explicado anteriormente, o uso do teste  $\chi^2$  não é recomendável quando as proporções são pequenas. Por

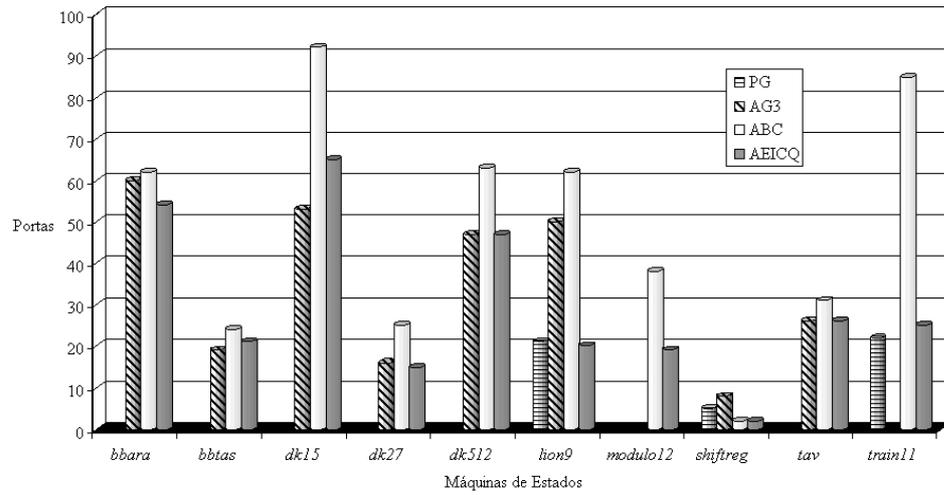


Figura 48: Comparação das lógicas de controle em termos da quantidade de portas

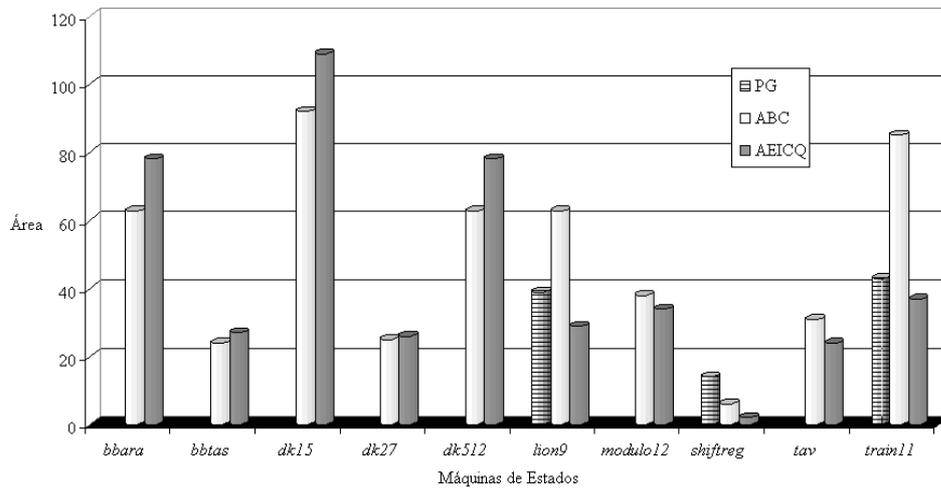


Figura 49: Comparação das lógicas de controle em termos de área consumida

tanto, os valores de atrasos foram convertidos para 0,1 ns ao invés de ns, evitando assim a limitação de aplicação do teste.

No caso da comparação AEICQ  $\times$  ABC, todas as 10 máquinas de estados listadas na Tabela 21 foram usadas, enquanto no caso das comparações AEICQ  $\times$  PG, somente três máquinas foram usadas, tendo em vista a disponibilidade de resultados, conforme mostra a Tabela 20. Com base nesta análise estatística, os resultados obtidos pelo AEICQ podem ser considerados significativos.

Os gráficos apresentados na Figura 51 – Figura 55 mostram o progresso da evolução da aptidão da melhor solução e da aptidão média da população para cada uma das máquinas de estados usadas neste experimento.

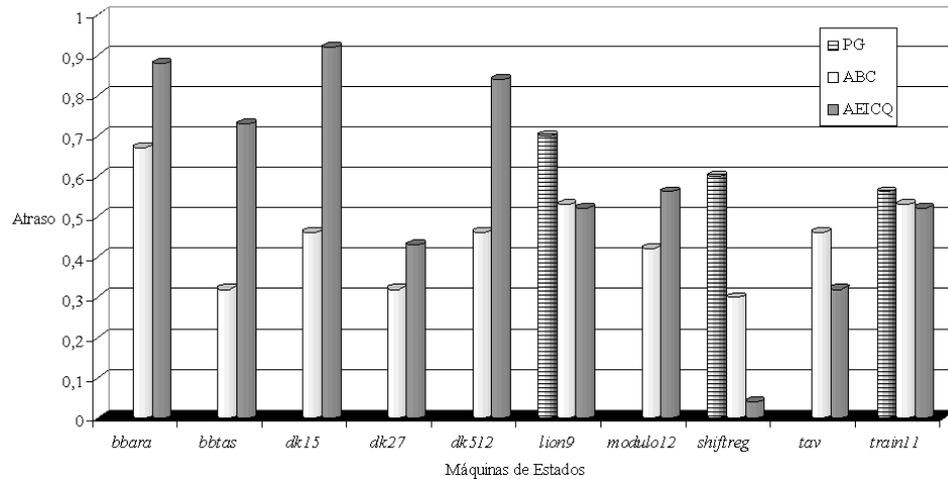


Figura 50: Comparação das lógicas de controle em termos de atraso de propagação

Tabela 22: Grau de liberdade, valores de  $\chi^2$  calculado, valores críticos de  $\chi^2$  para o nível de confiança 99,5% e níveis de confiança obtidos para as diferentes comparações

| Comparação         | Grau de liberdade | $\chi^2$ | Valor crítico | Nível de confiança |
|--------------------|-------------------|----------|---------------|--------------------|
| AEICQ $\times$ PG  | 10                | 18,898   | 18,31         | >95,0%             |
| AEICQ $\times$ ABC | 45                | 97,823   | 69,96         | >99,5%             |

O problema da escalabilidade descrito na Seção 5.1 e que já foi relatado em outros trabalhos que fizeram uso dos algoritmos genéticos para evolução de circuitos, como por exemplo, (ALI, 2003) e (ZEBULUM, 1999), também foi percebido durante os experimentos realizados com o AEICQ. Para uma máquina de estados de menor complexidade como a *shiftreg*, é possível codificar o circuito com uma geometria  $4 \times 3$ , composta de 108 q-bits. Neste caso, uma população composta de apenas 20 q-indivíduos é suficiente para a obtenção de boas soluções. Porém para as máquinas mais complexas, como por exemplo, a *bbara*, é necessária a utilização de uma geometria de  $32 \times 5$ , composta de 2080 q-bits. Neste caso, o espaço de busca fica extremamente grande, tornando necessário o aumento da população para que uma parte significativa deste espaço possa ser explorada. Com isso, a combinação do aumento da geometria e da população devido à complexidade do circuito leva a um aumento considerável no tempo médio de execução. Este tempo, que no caso da máquina *shiftreg* é de 3 minutos para 5000 gerações com uma população de 20 q-indivíduos, passa para 5 horas para a máquina *bbara*, considerando uma população de 200 q-indivíduos e 5000 gerações.

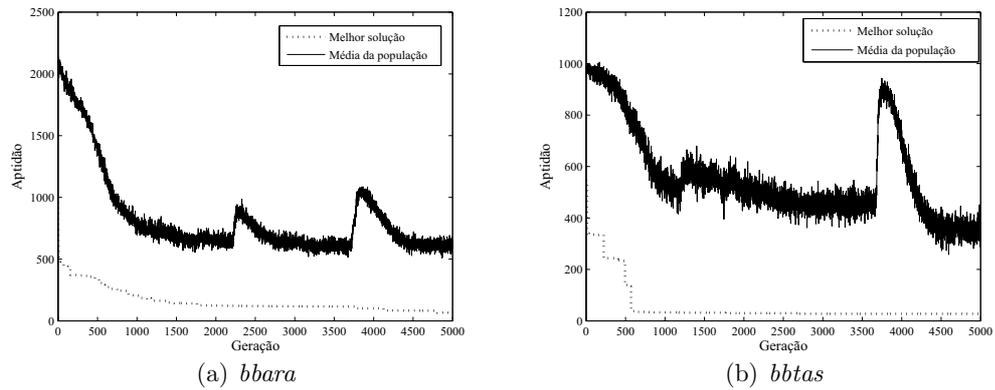


Figura 51: Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados *bbara* e *bbtas*

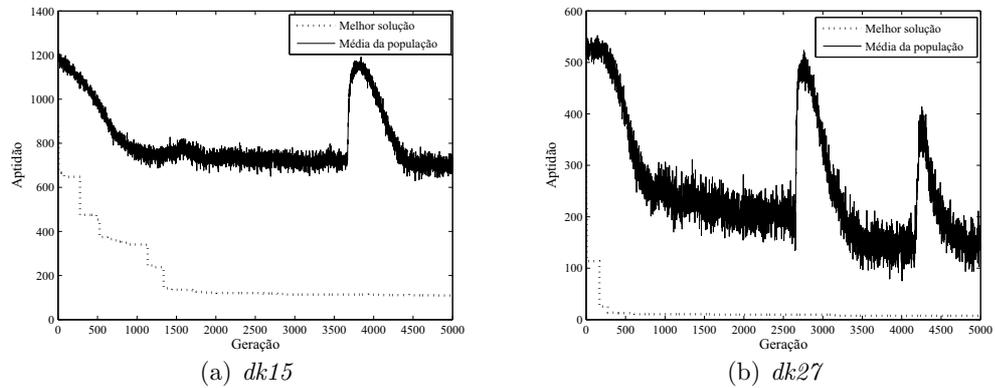


Figura 52: Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados *dk15* e *dk27*

### 5.5.2 Discussão dos Resultados

O aumento do tempo requerido para a evolução dos circuitos acaba trazendo uma dificuldade adicional para o ajuste ótimo dos parâmetros do AEICQ, pois o elevado tempo de simulação inviabiliza a execução de um número grande de experimentos necessário para o ajuste destes parâmetros para cada uma das máquinas. Por este motivo, estes parâmetros foram ajustados de forma mais apurada para as máquinas *lion9* e *train11* e os seus valores utilizados na evolução da lógica de controle das demais máquinas. Ainda assim, os resultados obtidos para estas máquinas de estados podem ser considerados satisfatórios. Os resultados apresentados confirmam que os algoritmos evolucionários inspirados na computação quântica apresentam uma boa robustez quanto ao ajuste dos seus parâmetros.

Os resultados mostrados na seção anterior sugerem que o AEICQ é uma ferramenta com

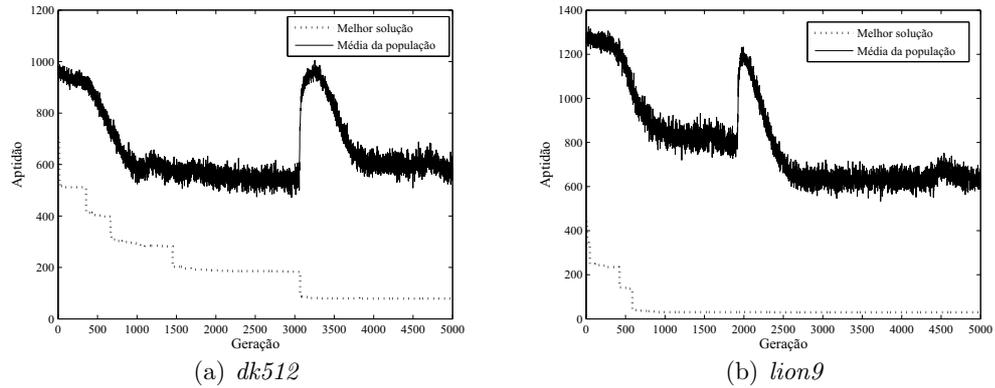


Figura 53: Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados *dk512* e *lion9*

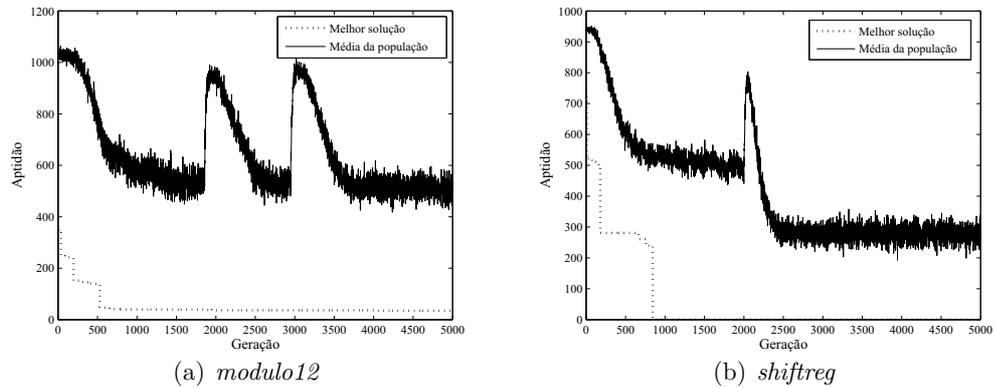


Figura 54: Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados *modulo12* e *shiftreg*

um grande potencial para ser aplicada no domínio da eletrônica evolucionária. Os circuitos evoluídos possuem características similares aos circuitos obtidos por outros métodos, entre eles a ferramenta ABC, que é reconhecida como uma ferramenta poderosa na síntese de circuitos. Em alguns casos os circuitos evoluídos pelo AEICQ possuem área e/ou atraso de propagação menores que os obtidos pelos demais métodos, inclusive ABC.

## 5.6 Resumo do Capítulo

Este capítulo investigou a aplicação do Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ) como ferramenta de síntese automática de circuitos digitais. O AEICQ foi aplicado com sucesso na evolução da lógica de controle de máquinas de estados finitos. Os resultados alcançados são compatíveis com os resultados obtidos por outros métodos evo-

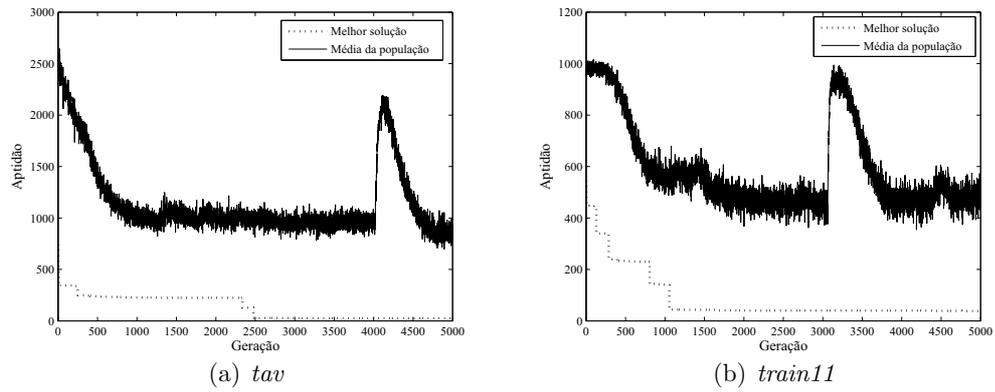


Figura 55: Progresso da aptidão da melhor solução e da aptidão média da população para a síntese da lógica de controle das máquinas de estados *tav* e *train11*

lucionários e pela ferramenta de síntese ABC. No próximo capítulo, conclui-se este trabalho apresentando todos os pontos relevantes desenvolvidos nesta dissertação.

# Capítulo 6

## Conclusões e Trabalhos Futuros

**E**STA dissertação apresentou um estudo da aplicação de algoritmos evolucionários inspirados nos princípios da computação quântica no projeto de sistemas seqüenciais síncronos. O objetivo deste trabalho foi verificar a aplicação destes algoritmos na obtenção de circuitos eletrônicos otimizados segundo a área ocupada e o atraso de propagação dos mesmos.

### 6.1 Conclusões

A metodologia adotada neste trabalho para o projeto de sistemas seqüenciais foi estruturada em três etapas principais: *(i)* A implementação do Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ); *(ii)* A adaptação da implementação do AEICQ para ser aplicado na resolução do problema de atribuição de estados e *(iii)* A adaptação da implementação do AEICQ para ser aplicado na síntese automática de circuitos digitais evolucionários.

O Algoritmo Evolucionário Inspirado na Computação Quântica (AEICQ), utilizado nos experimentos apresentados neste trabalho, segue a estrutura básica dos algoritmos evolucionários inspirados nos princípios da computação quântica que encontram-se descritos na literatura. Os indivíduos são representados de forma probabilística por meio de um conjunto de bits quânticos ou  $q$ -bits. A porta *rotação* foi utilizada como um operador de variação pelo AEICQ. Foram implementados também dois mecanismos que têm como objetivo permitir que o AEICQ escape de soluções locais: a *restrição de probabilidades* e a *migração global*. A restrição de probabilidade evita a ocorrência de uma convergência prematura dos  $q$ -bits para os estados 0 ou 1. Nas operações de migração global as melhores soluções obtidas por cada um dos  $q$ -indivíduos e que são utilizadas no processo de atualização das amplitudes de probabilidade dos  $q$ -bits são substituídas pela melhor solução encontrada até a geração atual. Esta operação é executada pelo AEICQ sempre que a processo evolucionário permanece um número pré-determinado de gerações sem conseguir obter uma solução melhor que a obtida até a geração atual. Desta

forma, a migração global permite aumentar a diversidade da população e, por consequência, a convergência do processo.

A primeira aplicação do AEICQ foi na resolução do problema de atribuição de estados. O objetivo neste caso foi a obtenção de atribuições de estados capazes de conduzirem a implementações de lógicas de controle otimizadas para máquinas de estados finitos. Os indivíduos correspondentes às atribuições possíveis foram representados por um conjunto de  $q$ -bits de tamanho adequado para identificar todos os estados da máquina de estados. Para avaliar a aptidão das atribuições evoluídas, o AEICQ utilizou duas heurísticas que atribuem códigos adjacentes aos estados que têm o mesmo próximo estado e também aos estados que têm o mesmo estado antecessor (ARMSTRONG, 1962) e (HUMPHREY, 1958) para avaliação das atribuições.

Para a validação do desempenho do AEICQ foram realizados vários experimentos, utilizando máquinas de estados finitos citadas como referência nas pesquisas envolvendo sistemas seqüenciais e disponíveis em (ACM/SIGDA, 1989). Os resultados alcançados pelo AEICQ (ARAUJO; NEDJAH; MOURELLE, 2008b) e (ARAUJO; NEDJAH; MOURELLE, 2008c) foram expressivos, tendo sido, na maioria dos casos, superiores aos obtidos pelos algoritmos genéticos e pela ferramenta NOVA™.

Na segunda aplicação, o AEICQ foi adaptado para ser usado na síntese da lógica de controle das máquinas de estados finitos (ACM/SIGDA, 1989). Segundo a terminologia utilizada na eletrônica evolucionária, estes experimentos são classificados como extrínsecos. A unidade de circuito manipulada pelo AEICQ é a porta lógica, caracterizando portanto uma aplicação no nível de portas lógicas. Os circuitos foram representados através de matrizes de células, onde cada célula codifica uma porta lógica assim como o mapeamento dos seus respectivos sinais de entrada. A função de avaliação das soluções utilizada pelo AEICQ possui dois objetivos, os quais são: (i) O circuito evoluído deve ser totalmente funcional; (ii) O circuito deve ser otimizado com relação à área ocupada e ao atraso de propagação dos sinais de saída. Para a otimização dos circuitos foi utilizada a técnica da soma ponderada para agregar os objetivos relacionados à área e ao atraso de propagação dos mesmos.

Durante os experimentos, a síntese da lógica de controle de algumas das máquinas de estados finitos foi mais explorada. Para estas máquinas buscou-se um ajuste ótimo de todos os parâmetros relacionados ao AEICQ, tais como: número máximo de gerações; ângulo da porta rotação; tamanho da população; e o momento em que deve acontecer a migração global. Com o uso destes parâmetros, os resultados alcançados pelo AEICQ foram superiores aos obtidos por outros métodos, mostrando que esta é uma ferramenta promissora nas aplicações envolvendo

a eletrônica evolucionária. Os experimentos foram então estendidos para outras máquinas de estados finitos. Devido aos tempos de simulação serem relativamente longos, não foi possível, dentro do tempo disponível para esta pesquisa, obter parâmetros otimizados para cada uma das máquinas, sendo então realizadas apenas algumas variações em torno dos parâmetros utilizados naquelas máquinas. Ainda assim, os resultados alcançados pelo AEICQ (ARAUJO; NEDJAH; MOURELLE, 2008a) foram compatíveis com os resultados obtidos pelos algoritmos genéticos, pela programação genética e com a ferramenta não evolucionária ABC<sup>TM</sup>. Estes resultados demonstram que os algoritmos evolucionários inspirados na computação quântica são robustos quanto ao ajuste dos seus parâmetros. Vale ressaltar que o problema da escalabilidade, o qual é citado nos trabalhos (ALI, 2003), (ZEBULUM, 1999) e (YAO; HIGUCHI, 1999), também foi percebido nos experimentos envolvendo as máquinas de estados mais complexas, realizados neste trabalho.

Em resumo, o principal objetivo deste trabalho foi atingido. Uma nova técnica utilizando os algoritmos evolucionários inspirados na computação quântica foi implementada para o projeto de sistemas sequenciais síncronos. A técnica proposta atingiu resultados comparáveis com os de outras técnicas e muitas vezes superiores.

## 6.2 Trabalhos Futuros

A utilização dos algoritmos evolucionários no domínio da eletrônica é uma área com inúmeras possibilidades de pesquisa. Portanto, existem muitas direções que podem ser seguidas em trabalhos futuros.

Para o problema da atribuição de estados pode ser investigada a utilização de outras heurísticas em conjunto com as heurísticas de (ARMSTRONG, 1962) e (HUMPHREY, 1958) para otimizar ainda mais a lógica de controle evoluída.

Para a síntese de circuitos, pode-se estudar a utilização dos algoritmos evolucionários inspirados na computação quântica na evolução de circuitos em nível de funções. O trabalho também pode ser estendido para a síntese de circuitos lógicos diretamente em dispositivos reconfiguráveis, ou seja, em aplicação do tipo intrínseca.

A co-evolução pode ser investigada como uma possível solução para a redução do tempo computacional na síntese de máquinas com elevado número de estados e/ou transições de estados.

Uma outra direção para a continuidade deste trabalho é a investigação do ajuste adaptativo de alguns dos parâmetros do AEICQ, principalmente nas aplicações em eletrônica evo-

lucionária. A geometria da matriz de células que codifica os circuitos é um exemplo de um dos parâmetros que poderia ser ajustado de forma adaptativa.

O projeto dos sistemas seqüenciais assíncronos também representa um aspecto importante para a continuidade deste trabalho. Os projetos assíncronos possuem um nível de complexidade ainda maior que os síncronos e ainda são poucos os trabalhos disponíveis na literatura sobre este assunto.

Por fim, o processo evolucionário também pode ser aplicado no desenvolvimento de sistemas capazes de se adaptarem em tempo real a uma condição de falha ou alteração do ambiente em que o mesmo encontra-se inserido.

# REFERÊNCIAS

- ABC: A System for Sequential Synthesis and Verification, Release 70930. 2005. Disponível em: <<http://www.eecs.berkeley.edu/~alanmi/abc/>>. Acesso em: 30 jul. 2008.
- ABRAHAM, A.; NEDJAH, N.; MOURELLE, L. M. Evolutionary Computation: from Genetic Algorithms to Genetic Programming. In: ABRAHAM, A.; NEDJAH, N.; MOURELLE, L. M. (Ed.). *Genetic Systems Programming*. 1. ed. Berlin, Germany: Springer-Verlag, 2005. p. 1–20.
- ACM/SIGDA. *Collaborative Benchmarking and Experimental Algorithmics*. 1989. Disponível em: <<http://www.cbl.ncsu.edu:16080/benchmarks/LGSynth89/fsmexamples/>>. Acesso em: 01 jun. 2008.
- AKBARZADEH-T, M.-R.; KHORSAND, A.-R. Quantum Gate Optimization in a Meta-Level Genetic Quantum Algorithm. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS, 2005, Hawaii, USA. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 2005. v. 4, p. 3055–3062.
- ALFARES, F. S.; ESAT, I. I. Real-Coded Quantum Inspired Evolution Algorithm Applied to Engineering Optimization Problems. In: INTERNATIONAL SYMPOSIUM ON LEVERAGING APPLICATIONS OF FORMAL METHODS, VERIFICATION AND VALIDATION, 2., 2006, Paphos, Cyprus. *Proceedings...* Los Alamitos, CA, USA: IEEE Press, 2007. p. 169–176.
- ALI, B. *Evolutionary Algorithms for Synthesis and Optimization of Sequential Logic Circuits*. 208 p. Tese (Doctor of Philosophy) — School of Engineering of Napier University, Edinburgh, UK, 2003.
- AMARAL, J. F. M. do *et al.* Intrinsic Evolutionary Design of Analog Building Blocks for Fuzzy Logic Controllers. In: NEDJAH, N.; MOURELLE, L. M. (Ed.). *Evolutionary Machine Design: Methodology and Applications*. 1. ed. New York, USA: Nova Science Publisher, Inc, 2005. p. 59–80.

- AMARAL, J. N.; TUMER, K.; GLOSH, J. Designing Genetic Algorithms for the State Assignment Problem. *IEEE Transactions on Systems, Man, and Cybernetics*, v. 25, n. 4, p. 686–694, apr. 1995.
- ANDRE, J. C. S. *Probabilidades e Estatística para Engenharia*. USA: Lidel, 2008. 600 p.
- ARAUJO, M. P. M.; NEDJAH, N.; MOURELLE, L. M. Optimised State Assignment for FSMs Using Quantum Inspired Evolutionary Algorithm. In: INTERNATIONAL CONFERENCE ON EVOLVABLE SYSTEMS: FROM BIOLOGY TO HARDWARE, 8., 2008, Prague, Czech Republic. *Proceedings...* Berlin, Germany: Springer-Verlag, 2008a. (Lecture Notes in Computer Science, v. 5216), p. 332–341.
- ARAUJO, M. P. M.; NEDJAH, N.; MOURELLE, L. M. Logic Synthesis for FSMs Using Quantum Inspired Evolution. In: INTERNATIONAL CONFERENCE ON INTELLIGENT DATA ENGINEERING AND AUTOMATED LEARNING, 9., 2008, Daejeon, Korea. *Proceedings...* Berlin, Germany: Springer-Verlag, 2008b. (Lecture Notes in Computer Science, v. 5326), p. 32–39.
- ARAUJO, M. P. M.; NEDJAH, N.; MOURELLE, L. M. Quantum-Inspired Evolutionary State Assignment for Synchronous Finite State Machines. *Journal of Universal Computer Science*, v. 14, n. 15, p. 2532–2548, dec. 2008c.
- ARMSTRONG, D. B. A Programmed Algorithm for Assigning Internal Codes to Sequential Machines. *IRE Transactions on Electronic Computers*, EC-11, n. 4, p. 466–472, feb. 1962.
- BABU, G.; DAS, D.; PATVARDHAN, C. Real-parameter quantum evolutionary algorithm for economic load dispatch. *Generation, Transmission and Distribution, IET*, v. 2, n. 1, p. 22–31, jan. 2008.
- BALL, P. 2020 computing: Champing at the bits. *Nature*, n. 440, p. 398–401, mar. 2006.
- BOOTH, T. L. *Sequential Machines and Automata Theory*. New York, USA: John Wiley & Sons, 1967. 592 p.
- COELHO, L. S.; NEDJAH, N.; MOURELLE, L. M. Gaussian Quantum-Behaved Particle Swarm Optimization Applied to Fuzzy PID Controller Design. In: NEDJAH, N.; COELHO, L. S.; MOURELLE, L. M. (Ed.). *Quantum Inspired Intelligent Systems*. 1. ed. Berlin, Germany: Springer-Verlag, 2008. p. 1–15.

- CRUZ, A. V. A. da; VELLASCO, M. M. B. R.; PACHECO, M. A. C. Quantum-Inspired Evolutionary Algorithm for Numerical Optimization. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2006, Vancouver, BC, Canada. *Proceedings...* Los Alamitos, CA, USA: IEEE Press, 2006. p. 2630–2637.
- CRUZ, A. V. A. da; VELLASCO, M. M. B. R.; PACHECO, M. A. C. Quantum-Inspired Evolutionary Algorithm for Numerical Optimization. In: NEDJAH, N.; COELHO, L. S.; MOURELLE, L. M. (Ed.). *Quantum Inspired Intelligent Systems*. 1. ed. Berlin, Germany: Springer-Verlag, 2008. p. 115–132.
- DEB, K. Encoding and Decoding Functions. In: BACK, T.; FOGEL, D. B.; MICHALEWICZ, Z. (Ed.). *Evolutionary Computation 2: Advanced Algorithms and Operators*. 1. ed. Bristol, UK: Institute of Physics Publishing, 2000. p. 4–11.
- DEB, K. *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, USA: John Wiley & Sons, 2001. 542 p.
- DEUTSCH, D. Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. In: *Proceedings of the Royal Society of London*. London, UK: The Royal Society, 1985. (Series A, Mathematical and Physical Sciences, v. 400), p. 97–117.
- DEUTSCH, D. Quantum Computational Networks. In: *Proceedings of the Royal Society of London*. London, UK: The Royal Society, 1989. (Series A, Mathematical and Physical Sciences, v. 425), p. 73–90.
- DEVADAS, S. *et al.* MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations. *IEEE Transactions on Computer-Aided Design*, v. 7, n. 12, p. 1290–1300, dec. 1988.
- DIRAC, P. A. M. *The Principles of Quantum Mechanics*. 4. ed. [S.l.]: Oxford University Press, 1958.
- ERCEGOVAC, M.; LANG, T.; MORENO, J. H. *Introdução aos Sistemas Digitais*. Porto Alegre, Brasil: Bookman, 1999. 512 p.
- FEYNMAN, R. P. Simulating Physics with Computers. *International Journal of Theoretical Physics*, v. 21, n. 6/7, p. 467–488, 1982.

- FOGEL, L. J. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ, USA: IEEE Press, 1999.
- FOGEL, L. J.; OWENS, A. J.; WALSH, M. J. *Artificial Intelligence through Simulated Evolution*. New York, USA: John Wiley & Sons, 1966.
- FONSECA, C. M.; FLEMING, P. J. An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation*, v. 3, n. 1, p. 1–16, 1995.
- GAJSKI, D.; KUHN, R. Guest Editors' Introduction: New VLSI Tools. *IEEE Computer*, v. 16, n. 12, p. 11–14, dec. 1983.
- GAREY, M. R.; JOHNSON, D. S. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. USA: W.H. Freeman, 1979.
- GARIS, H. de. Evolvable Hardware: Genetic Programming of a Darwin Machine. In: INTERNATIONAL CONFERENCE ON ARTIFICIAL NEURAL NETS AND GENETIC ALGORITHMS, 1., 1993, San Francisco, CA, USA. *Proceedings...* Berlin, Germany: Springer-Verlag, 1993. p. 441–449.
- GLOVER, F. Tabu Search: A Tutorial. *Interfaces*, v. 20, n. 4, p. 74–94, 1990.
- GROVER, L. K. A fast quantum mechanical algorithm for database search. In: ANNUAL ACM SYMPOSIUM ON THE THEORY OF COMPUTING, 28., 1996, Philadelphia, PA, USA. *Proceedings...* New York, USA: ACM, 1996. p. 212–219.
- HALLGREN, S. Polynomial-time quantum algorithms for Pell's equation and the principal ideal problem. In: ANNUAL ACM SYMPOSIUM ON THE THEORY OF COMPUTING, 34., 2002, Quebec, Canada. *Proceedings...* New York, USA: ACM, 2002. p. 653–658.
- HAN, K.-H. *Quantum-inspired Evolutionary Algorithm*. 127 p. Tese (Doctor of Philosophy) — Korea Advanced Institute of Science and Technology, Daejeon, Korea, 2003.
- HAN, K.-H.; KIM, J.-H. Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 6, p. 580–593, dec. 2002.
- HAN, K.-H.; KIM, J.-H. Quantum-Inspired Evolutionary Algorithm With a New Termination Criterion, He Gate, and Two-phase Scheme. *IEEE Transactions on Evolutionary Computation*, v. 2, n. 8, p. 156–169, dec. 2004.

- HAN, K.-H.; KIM, J.-H. On the Analysis of the Quantum-inspired Evolutionary Algorithm with a Single Individual. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2006, Vancouver, BC, Canada. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 2006. p. 2622–2629.
- HAN, K.-H.; KIM, J.-H. *et al.* A Quantum-Inspired Evolutionary Computing Algorithm for Disk Allocation Method. *IEICE Transactions on Information and Systems*, E86-D, n. 3, p. 645–649, mar. 2003.
- HARTMANIS, J. On the state assignment problem for sequential machines. *IRE Transactions on Electronic Computers*, EC-10, n. 2, p. 157–165, dec. 1961.
- HEMMI, H.; MIZOGUCHI, J.; SHIMOHARA, K. Development and evolution of hardware behaviors. In: SANCHEZ, E.; TOMASSINI, M. (Ed.). *Toward Evolvable Hardware: The Evolutionary Engineering Approach*. 1. ed. Berlin, Germany: Springer-Verlag, 1996, (Lecture Notes in Computer Science, v. 1062). p. 250–265.
- HEY, T. Quantum computing: an introduction. *Computing Control Engineering Journal*, v. 10, n. 3, p. 105–112, june 1999.
- HIGUCHI, T. *et al.* Evolving hardware with genetic learning: A first step toward building a Darwin machine. In: INTERNATIONAL CONFERENCE ON SIMULATION ADAPTIVE BEHAVIOUR, 2., 1992, Massachusetts, USA. *Proceedings...* Cambridge, MA, USA: MIT Press, 1992. p. 417–424.
- HIGUCHI, T. *et al.* Real-World Applications of Analog and Digital Evolvable Hardware. *IEEE Transactions on Evolutionary Computation*, v. 3, n. 3, p. 220–235, sep. 1999.
- HINTERDING, R. Representation, Constraint Satisfaction and the Knapsack Problem. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 1999, Washington DC, USA. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 1999. v. 2, p. 1286–1292.
- HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975.
- HUMPHREY, W. S. *Switching Circuits with Computer Applications*. New York, USA: McGraw-Hill, 1958. 272 p.
- JANG, J.-S.; HAN, K.-H.; KIM, J.-H. Face Detection using Quantum-inspired Evolutionary Algorithm. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2004, Portland, OR, USA. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 2004. v. 2, p. 2100–2106.

- KALGANOVA, T. An Extrinsic Function-Level Evolvable Hardware Approach. In: EUROPEAN CONFERENCE ON GENETIC PROGRAMMING, 3., 2000, Edinburgh, UK. *Proceedings...* Berlin, Germany: Springer-Verlag, 2000. p. 60–75.
- KEYMEULEN, D. *et al.* Fault-Tolerant Evolvable Hardware Using Field Programmable Transistor Arrays. *IEEE Transactions on Reliability*, v. 49, n. 3, p. 305–316, sep. 2000.
- KHAN, F. N. *FSM State-Assignment for Area, Power and Testability Using Non-Deterministic Evolutionary Heuristics*. 234 p. Dissertação (Master of Science) — King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, 2005.
- KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983.
- KOEHN, P. *Combining Genetic Algorithms and Neural Networks: The Encoding Problem*. 67 f. Dissertação (Master of Science) — University of Tennessee, Tennessee, USA, 1994.
- KOZA, J. R. *Genetic programming: On the programming of computers by natural selection*. Cambridge, MA, USA: MIT Press, 1992.
- KOZA, J. R.; KEANE, M. A.; STREETER, M. J. Routine High-Return Human-Competitive Evolvable Hardware. In: NEDJAH, N.; MOURELLE, L. M. (Ed.). *Evolutionary Machine Design: Methodology and applications*. 1. ed. New York, USA: Nova Science Publisher, Inc, 2005. p. 3–32.
- KRUISKAMP, W.; LEENAERTS, D. Darwin: CMOS opamp synthesis by means of a genetic algorithm. In: ACM/IEEE CONFERENCE ON DESIGN AUTOMATION, 32., 1995, San Francisco, CA, USA. *Proceedings...* New York, USA: ACM, 1995. p. 433–438.
- LIU, W.; MURAKAWA, M.; HIGUCHI, T. ATM Cell Schedule by Function Level Evolvable Hardware. In: INTERNATIONAL CONFERENCE ON EVOLVABLE SYSTEMS: FROM BIOLOGY TO HARDWARE, 1., 1996, Tsukuba, Japan. *Proceedings...* Berlin, Germany: Springer-Verlag, 1997. (Lecture Notes in Computer Science, v. 1259), p. 180–194.
- LOHN, J. D.; COLOMBANO, S. P. Automated Analog Circuit Synthesis Using a Linear Representation. In: INTERNATIONAL CONFERENCE ON EVOLVABLE SYSTEMS: FROM BIOLOGY TO HARDWARE, 2., 1998, Lausanne, Switzerland. *Proceedings...* [S.l.]: Springer-Verlag, 1998. (Lecture Notes in Computer Science, v. 1478), p. 125–133.

- MAZUMDER, P.; RUDNICK, E. *Genetic Algorithm for VLSI Design, Layout and Test Automation*. New York, USA: Prentice Hall, 1998.
- MICHALEWICZ, Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Germany: Springer-Verlag, 1992.
- MICHELI, G. de; BRAYTON, R.; SANGIOVANNI-VINCENTELLI, A. KISS: A program for Optimal State Assignment of Finite State Machines. In: INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN, 1984, California, USA. *Proceedings...* New York, USA: ACM Press, 1984. p. 209–211.
- MILLER, J. F. *et al.* The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles. In: SYMPOSIUM ON CREATIVE EVOLUTIONARY SYSTEMS, 1., 1999, Edimburg, UK. *Proceedings...* San Francisco, CA, USA: Morgan Kaufman Pub, 1999. p. 65–74.
- MILLER, J. F.; THOMSON, P. Combinational and Sequential Logic Optimisation Using Genetic Algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS IN ENGINEERING SYSTEMS, 1., 1995, Galesia. *Proceedings...* [S.l.]: IEEE Press, 1995. p. 34–38.
- MISHCHENKO, A.; CHATTERJEE, S.; BRAYTON, R. DAG-Aware AIG Rewriting: A Fresh Look at Combinational Logic Synthesis. In: DESIGN AUTOMATION CONFERENCE, 43., 2006, California, USA. *Proceedings...* New York, USA: ACM Press, 2006. p. 532–535.
- NARAYANAN, A. Quantum computing for beginners. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 1999, Washington D.C., USA. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 1999. v. 3, p. 2231–2238.
- NARAYANAN, A.; MOORE, M. Quantum-Inspired Genetic Algorithms. In: INTERNATIONAL CONFERENCE ON EVOLUTIONARY COMPUTATION, 1996, Nogaya, Japan. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 1996. p. 61–66.
- NEDJAH, N.; MOURELLE, L. M. Real-world Evolutionary Designs: Secure Evolvable Hardware for Public-Key Cryptosystems. In: *Evolutionary Machine Design: Methodology and applications*. 1. ed. New York, USA: Nova Science Publisher, Inc, 2005. p. 111–138.

- NEDJAH, N.; MOURELLE, L. M. Evolutionary Synthesis of Synchronous Finite State Machines. In: NEDJAH, N.; MOURELLE, L. M. (Ed.). *Evolvable Machines: Theory and practice*. 1. ed. Berlin, Germany: Springer-Verlag, 2005a. p. 103–128.
- OLIVEIRA, I. S. *et al.* Computação Quântica: Manipulando a informação oculta do mundo quântico. *Ciência Hoje*, v. 33, n. 193, p. 22–29, maio 2003.
- PAPADIMITRIOU, C. H. *Computational Complexity*. Massachusetts, USA: Addison-Wesley, 1994. 523 p.
- PAPPAS, T. N. An adaptive clustering algorithm for image segmentation. *IEEE Transactions on Signal Processing*, v. 40, n. 4, p. 901–914, may 1992.
- PARETO, V. *Cours deconomie politique*. Rouge, Lausanne: [s.n.], 1896.
- PLATEL, M. D.; SCHLIEBS; KASABOV, N. A Versatile Quantum-inspired Evolutionary Algorithm. In: CONGRESS ON EVOLUTIONARY COMPUTATION, 2007, Singapore. *Proceedings...* [S.l.]: IEEE Press, 2007. p. 423–430.
- PORTUGAL, R. *et al.* *Uma Introdução à Computação Quântica*. São Carlos, SP, Brasil: SBMAC, 2004. 62 p. (Notas em Matemática Aplicada; 8).
- RECHENBERG, I. *Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog, 1973.
- RHYNE, V. T. *Fundamentals of digital systems design*. [S.l.]: Prentice-Hall, 1973. 560 p. (Computer Applications in Electrical Engineering Series).
- ROGENMOSER, R.; KAESLIN, H.; BLICKLE, T. Stochastic Methods for Transistor Size Optimization of CMOS VLSI Circuits. In: INTERNATIONAL CONFERENCE ON PARALLEL PROBLEM SOLVING FROM NATURE, 4., 1996, Berlin, Germany. *Proceedings...* London, UK: Springer-Verlag, 1996. (Lecture Notes In Computer Science, v. 1141), p. 849–858.
- RUDELL, R. L.; SANGIOVANNI-VINCENTELLI, A. Multiple-Valued Minimization for PLA Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 6, n. 5, p. 727–750, 1987.
- SCHWEFEL, H.-P. *Numerische Optimierung von Computer-Modellen mittels der Evolutionstrategie*. Basel: Birkhäuser, 1977.

- SHOR, P. W. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In: ANNUAL SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, 35., 1994, Santa Fe, NM. *Proceedings...* California, USA: IEEE Computer Society Press, 1994. p. 124–134.
- TALBI, H.; BATOUCHE, M.; DRAA, A. A Quantum-Inspired Evolutionary Algorithm for Multiobjective Image Segmentation. *International Journal of Mathematical, Physical and Engineering Sciences*, v. 1, n. 2, p. 109–114, 2007.
- THOMPSON, A. Silicon Evolution. In: GENETIC PROGRAMMING CONFERENCE, 1., 1996, California, USA. *Proceedings...* [S.l.]: MIT Press, 1996. p. 444–452.
- TU, Z.; LU, Y. A robust stochastic genetic algorithm (stGA) for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, v. 8, n. 5, p. 456–470, oct. 2004.
- UYEMURA, J. P. *Sistemas Digitais: Uma abordagem integrada*. São Paulo, SP, Brasil: Pioneira Thomson Learning, 2002. 433 p.
- VILLA, T.; SANGIOVANNI-VINCENTELLI, A. NOVA: state assignment of finite state machines for optimal two-level logic implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, v. 9, n. 9, p. 905–924, sep. 1990.
- YAO, X.; HIGUCHI, T. Promises and Challenges of Evolvable Hardware. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, v. 29, n. 1, p. 87–97, feb. 1999.
- ZEBULUM, R. S. *Síntese de Circuitos Eletrônicos por Computação Evolutiva*. 196 p. Tese (Doutorado em Ciências de Engenharia Elétrica) — Departamento de Engenharia Elétrica, PUC-Rio, Rio de Janeiro, Brasil, 1999.
- ZEBULUM, R. S.; PACHECO, M. A.; VELLASCO, M. M. B. R. *Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. USA: CRC Press LLC, 2002. 328 p. (The CRC Press international series on computational intelligence).
- ZHANG, G. *et al.* A Novel Genetic Algorithm and its Application to Digital Filter Design. In: IEEE CONFERENCE ON INTELLIGENT TRANSPORTATION SYSTEMS, 2003, Shanghai, China. *Proceedings...* Piscataway, NJ, USA: IEEE Press, 2003. v. 2, p. 1600–1605.
- ZHANG, G. *et al.* Novel Quantum Genetic Algorithm and its Applications. *Frontiers of Electrical and Electronic Engineering in China*, v. 1, n. 1, p. 31–36, jan. 2006.

# APÊNDICE A – Especificação das Máquinas de Estados de Referência

## A.1 Máquina de estados: *bbara*

Esta máquina de estados possui 10 estados e 60 transições de estado. A Tabela A.1 mostra a função de transição de estados da máquina.

Tabela A.1: Descrição da máquina de estados *bbara*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| --01    | $s_0$        | $s_0$          | 00    |
| --10    | $s_0$        | $s_0$          | 00    |
| --00    | $s_0$        | $s_0$          | 00    |
| 0011    | $s_0$        | $s_0$          | 00    |
| -111    | $s_0$        | $s_1$          | 00    |
| 1011    | $s_0$        | $s_4$          | 00    |
| --01    | $s_1$        | $s_1$          | 00    |
| --10    | $s_1$        | $s_1$          | 00    |
| --00    | $s_1$        | $s_1$          | 00    |
| 0011    | $s_1$        | $s_0$          | 00    |
| -111    | $s_1$        | $s_2$          | 00    |
| 1011    | $s_1$        | $s_4$          | 00    |
| --01    | $s_2$        | $s_2$          | 00    |
| --10    | $s_2$        | $s_2$          | 00    |
| --00    | $s_2$        | $s_2$          | 00    |
| 0011    | $s_2$        | $s_1$          | 00    |
| -111    | $s_2$        | $s_3$          | 00    |
| 1011    | $s_2$        | $s_4$          | 00    |
| --01    | $s_3$        | $s_3$          | 10    |
| --10    | $s_3$        | $s_3$          | 10    |
| --00    | $s_3$        | $s_3$          | 10    |
| 0011    | $s_3$        | $s_7$          | 00    |
| -111    | $s_3$        | $s_3$          | 10    |
| 1011    | $s_3$        | $s_4$          | 00    |
| --01    | $s_4$        | $s_4$          | 00    |
| --10    | $s_4$        | $s_4$          | 00    |
| --00    | $s_4$        | $s_4$          | 00    |
| 0011    | $s_4$        | $s_0$          | 00    |
| -111    | $s_4$        | $s_1$          | 00    |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 1011    | $s_4$        | $s_5$          | 00    |
| --01    | $s_5$        | $s_5$          | 00    |
| --10    | $s_5$        | $s_5$          | 00    |
| --00    | $s_5$        | $s_5$          | 00    |
| 0011    | $s_5$        | $s_4$          | 00    |
| -111    | $s_5$        | $s_1$          | 00    |
| 1011    | $s_5$        | $s_6$          | 00    |
| --01    | $s_6$        | $s_6$          | 01    |
| --10    | $s_6$        | $s_6$          | 01    |
| --00    | $s_6$        | $s_6$          | 01    |
| 0011    | $s_6$        | $s_7$          | 00    |
| -111    | $s_6$        | $s_1$          | 00    |
| 1011    | $s_6$        | $s_6$          | 01    |
| --01    | $s_7$        | $s_7$          | 00    |
| --10    | $s_7$        | $s_7$          | 00    |
| --00    | $s_7$        | $s_7$          | 00    |
| 0011    | $s_7$        | $s_8$          | 00    |
| -111    | $s_7$        | $s_1$          | 00    |
| 1011    | $s_7$        | $s_4$          | 00    |
| --01    | $s_8$        | $s_8$          | 00    |
| --10    | $s_8$        | $s_8$          | 00    |
| --00    | $s_8$        | $s_8$          | 00    |
| 0011    | $s_8$        | $s_9$          | 00    |
| -111    | $s_8$        | $s_1$          | 00    |
| 1011    | $s_8$        | $s_4$          | 00    |
| --01    | $s_9$        | $s_9$          | 00    |
| --10    | $s_9$        | $s_9$          | 00    |
| --00    | $s_9$        | $s_9$          | 00    |
| 0011    | $s_9$        | $s_0$          | 00    |
| -111    | $s_9$        | $s_1$          | 00    |
| 1011    | $s_9$        | $s_4$          | 00    |

## A.2 Máquina de estados: *bsse*

Esta máquina de estados possui 16 estados e 56 transições de estado. A Tabela A.2 mostra a função de transição de estados da máquina.

Tabela A.2: Descrição da máquina de estados *bsse*

| Entrada | Estado atual | Próximo estado | Saída   |
|---------|--------------|----------------|---------|
| 0-----  | $s_0$        | $s_0$          | 0000000 |
| 10----0 | $s_0$        | $s_1$          | 00110-0 |
| 10----1 | $s_0$        | $s_1$          | 00010-0 |
| 11----0 | $s_0$        | $s_{11}$       | 0011010 |
| 11----1 | $s_0$        | $s_{11}$       | 0001010 |
| 100---- | $s_1$        | $s_1$          | 00000-0 |
| 101-1-- | $s_1$        | $s_4$          | 10000-0 |
| 101-0-- | $s_1$        | $s_2$          | 10000-0 |

| Entrada | Estado atual | Próximo estado | Saída   |
|---------|--------------|----------------|---------|
| 0-----  | $s_1$        | $s_{11}$       | 000--10 |
| 11----- | $s_1$        | $s_{11}$       | 0000010 |
| 10----- | $s_2$        | $s_3$          | 00000-0 |
| 0-----  | $s_2$        | $s_{11}$       | 000--10 |
| 11----- | $s_2$        | $s_{11}$       | 0000010 |
| 10--0-- | $s_3$        | $s_2$          | 10000-0 |
| 10--1-- | $s_3$        | $s_4$          | 10000-0 |
| 0-----  | $s_3$        | $s_{11}$       | 000--10 |
| 11----- | $s_3$        | $s_{11}$       | 0000010 |
| 10----- | $s_4$        | $s_5$          | 00000-0 |
| 0-----  | $s_4$        | $s_{11}$       | 000--10 |
| 11----- | $s_4$        | $s_{11}$       | 0000010 |
| 10-1--  | $s_5$        | $s_4$          | 10000-0 |
| 10--1-- | $s_5$        | $s_4$          | 10000-0 |
| 10-00-- | $s_5$        | $s_6$          | 0100010 |
| 0-----  | $s_5$        | $s_{11}$       | 000--10 |
| 11----- | $s_5$        | $s_{11}$       | 0000010 |
| 10--0-  | $s_6$        | $s_6$          | 0100000 |
| 10--1-  | $s_6$        | $s_7$          | 01000-0 |
| 0-----  | $s_6$        | $s_{11}$       | 000--10 |
| 11----- | $s_6$        | $s_{11}$       | 0000010 |
| 10----- | $s_7$        | $s_8$          | 0000010 |
| 0-----  | $s_7$        | $s_{11}$       | 000--10 |
| 11----- | $s_7$        | $s_{11}$       | 0000010 |
| 10--0-  | $s_8$        | $s_8$          | 0000000 |
| 10--1-  | $s_8$        | $s_9$          | 10000-0 |
| 0-----  | $s_8$        | $s_{11}$       | 000--10 |
| 11----- | $s_8$        | $s_{11}$       | 0000010 |
| 10----- | $s_9$        | $s_{10}$       | 00000-0 |
| 0-----  | $s_9$        | $s_{11}$       | 000--10 |
| 11----- | $s_9$        | $s_{11}$       | 0000010 |
| 1001--  | $s_{10}$     | $s_{10}$       | 00000-0 |
| 10-01-- | $s_{10}$     | $s_1$          | 00010-0 |
| 10-00-- | $s_{10}$     | $s_6$          | 0100010 |
| 1011--  | $s_{10}$     | $s_9$          | 10000-0 |
| 0-----  | $s_{10}$     | $s_{11}$       | 000--10 |
| 11----- | $s_{10}$     | $s_{11}$       | 0000010 |
| 0---0-  | $s_{11}$     | $s_{11}$       | 000--00 |
| 11--0-  | $s_{11}$     | $s_{11}$       | 0000000 |
| 0---1-  | $s_{11}$     | $s_0$          | 000---1 |
| 10----- | $s_{11}$     | $s_1$          | 00000-0 |
| 11--1-  | $s_{11}$     | $s_{12}$       | 00001-0 |
| 11----- | $s_{12}$     | $s_{12}$       | 00001-0 |
| 10----- | $s_{12}$     | $s_1$          | 00000-0 |
| 0-----  | $s_{12}$     | $s_{11}$       | 000--10 |
| 0-----  | $s_{13}$     | $s_{11}$       | 000--10 |
| 0-----  | $s_{14}$     | $s_{11}$       | 000--10 |
| 0-----  | $s_{15}$     | $s_{11}$       | 000--10 |

### A.3 Máquina de estados: *bbtas*

Esta máquina de estados possui 6 estados e 24 transições de estado. A Tabela A.3 mostra a função de transição de estados da máquina.

Tabela A.3: Descrição da máquina de estados *bbtas*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 00      | $s_0$        | $s_0$          | 00    |
| 01      | $s_0$        | $s_1$          | 00    |
| 10      | $s_0$        | $s_1$          | 00    |
| 11      | $s_0$        | $s_1$          | 00    |
| 00      | $s_1$        | $s_0$          | 00    |
| 01      | $s_1$        | $s_2$          | 00    |
| 10      | $s_1$        | $s_2$          | 00    |
| 11      | $s_1$        | $s_2$          | 00    |
| 00      | $s_2$        | $s_1$          | 00    |
| 01      | $s_2$        | $s_3$          | 00    |
| 10      | $s_2$        | $s_3$          | 00    |
| 11      | $s_2$        | $s_3$          | 00    |
| 00      | $s_3$        | $s_4$          | 00    |
| 01      | $s_3$        | $s_3$          | 01    |
| 10      | $s_3$        | $s_3$          | 10    |
| 11      | $s_3$        | $s_3$          | 11    |
| 00      | $s_4$        | $s_5$          | 00    |
| 01      | $s_4$        | $s_4$          | 00    |
| 10      | $s_4$        | $s_4$          | 00    |
| 11      | $s_4$        | $s_4$          | 00    |
| 00      | $s_5$        | $s_0$          | 00    |
| 01      | $s_5$        | $s_5$          | 00    |
| 10      | $s_5$        | $s_5$          | 00    |
| 11      | $s_5$        | $s_5$          | 00    |

### A.4 Máquina de estados: *dk14*

Esta máquina de estados possui 7 estados e 56 transições de estado. A Tabela A.4 mostra a função de transição de estados da máquina.

Tabela A.4: Descrição da máquina de estados *dk14*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 000     | $s_1$        | $s_3$          | 00010 |
| 000     | $s_2$        | $s_1$          | 01001 |
| 000     | $s_3$        | $s_3$          | 10010 |
| 000     | $s_4$        | $s_3$          | 00010 |
| 000     | $s_5$        | $s_1$          | 01001 |
| 000     | $s_6$        | $s_1$          | 01001 |
| 000     | $s_7$        | $s_3$          | 10010 |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 100     | $s_2$        | $s_2$          | 01001 |
| 100     | $s_5$        | $s_2$          | 01001 |
| 100     | $s_6$        | $s_2$          | 01001 |
| 100     | $s_1$        | $s_4$          | 00010 |
| 100     | $s_3$        | $s_4$          | 10010 |
| 100     | $s_4$        | $s_4$          | 00010 |
| 100     | $s_7$        | $s_4$          | 10010 |
| 111     | $s_5$        | $s_1$          | 10001 |
| 111     | $s_6$        | $s_1$          | 10001 |
| 111     | $s_7$        | $s_1$          | 10001 |
| 111     | $s_1$        | $s_3$          | 01010 |
| 111     | $s_2$        | $s_3$          | 00100 |
| 111     | $s_3$        | $s_3$          | 01010 |
| 111     | $s_4$        | $s_3$          | 00100 |
| 110     | $s_5$        | $s_1$          | 10101 |
| 110     | $s_6$        | $s_1$          | 10101 |
| 110     | $s_7$        | $s_1$          | 10101 |
| 110     | $s_1$        | $s_4$          | 01010 |
| 110     | $s_3$        | $s_4$          | 01010 |
| 110     | $s_2$        | $s_5$          | 00100 |
| 110     | $s_4$        | $s_5$          | 00100 |
| 011     | $s_2$        | $s_2$          | 00101 |
| 011     | $s_5$        | $s_2$          | 00101 |
| 011     | $s_1$        | $s_3$          | 01000 |
| 011     | $s_3$        | $s_3$          | 01000 |
| 011     | $s_4$        | $s_3$          | 10100 |
| 011     | $s_6$        | $s_3$          | 10100 |
| 011     | $s_7$        | $s_3$          | 10100 |
| 001     | $s_2$        | $s_1$          | 00101 |
| 001     | $s_5$        | $s_1$          | 00101 |
| 001     | $s_1$        | $s_5$          | 00010 |
| 001     | $s_3$        | $s_5$          | 10010 |
| 001     | $s_4$        | $s_5$          | 00010 |
| 001     | $s_6$        | $s_5$          | 10100 |
| 001     | $s_7$        | $s_5$          | 10010 |
| 101     | $s_2$        | $s_1$          | 00001 |
| 101     | $s_5$        | $s_2$          | 10001 |
| 101     | $s_6$        | $s_2$          | 10001 |
| 101     | $s_7$        | $s_2$          | 10001 |
| 101     | $s_1$        | $s_5$          | 01010 |
| 101     | $s_3$        | $s_5$          | 01010 |
| 101     | $s_4$        | $s_5$          | 10100 |
| 010     | $s_2$        | $s_2$          | 00001 |
| 010     | $s_5$        | $s_2$          | 10101 |
| 010     | $s_6$        | $s_2$          | 10101 |
| 010     | $s_7$        | $s_2$          | 10101 |
| 010     | $s_1$        | $s_6$          | 01000 |
| 010     | $s_3$        | $s_6$          | 01000 |
| 010     | $s_4$        | $s_7$          | 10000 |

## A.5 Máquina de estados: *dk15*

Esta máquina de estados possui 4 estados e 32 transições de estado. A Tabela A.5 mostra a função de transição de estados da máquina.

Tabela A.5: Descrição da máquina de estados *dk15*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 000     | $s_1$        | $s_1$          | 00101 |
| 000     | $s_2$        | $s_2$          | 10010 |
| 000     | $s_3$        | $s_1$          | 00101 |
| 000     | $s_4$        | $s_2$          | 10010 |
| 001     | $s_1$        | $s_2$          | 00010 |
| 001     | $s_2$        | $s_2$          | 10100 |
| 001     | $s_3$        | $s_2$          | 00010 |
| 001     | $s_4$        | $s_2$          | 10100 |
| 010     | $s_1$        | $s_3$          | 00010 |
| 010     | $s_2$        | $s_3$          | 10010 |
| 010     | $s_3$        | $s_3$          | 00010 |
| 010     | $s_4$        | $s_3$          | 10010 |
| 011     | $s_3$        | $s_1$          | 00100 |
| 011     | $s_4$        | $s_1$          | 00100 |
| 011     | $s_1$        | $s_2$          | 10001 |
| 011     | $s_2$        | $s_2$          | 10001 |
| 111     | $s_3$        | $s_1$          | 00100 |
| 111     | $s_4$        | $s_1$          | 00100 |
| 111     | $s_1$        | $s_3$          | 10101 |
| 111     | $s_2$        | $s_3$          | 10101 |
| 100     | $s_1$        | $s_1$          | 01001 |
| 100     | $s_3$        | $s_1$          | 10100 |
| 100     | $s_4$        | $s_1$          | 01001 |
| 100     | $s_2$        | $s_3$          | 01001 |
| 101     | $s_1$        | $s_2$          | 01010 |
| 101     | $s_2$        | $s_2$          | 01010 |
| 101     | $s_3$        | $s_2$          | 01000 |
| 101     | $s_4$        | $s_2$          | 01010 |
| 110     | $s_1$        | $s_3$          | 01010 |
| 110     | $s_2$        | $s_3$          | 01010 |
| 110     | $s_4$        | $s_3$          | 10000 |
| 110     | $s_3$        | $s_4$          | 01010 |

## A.6 Máquina de estados: *dk16*

Esta máquina de estados possui 27 estados e 108 transições de estado. A Tabela A.6 mostra a função de transição da máquina.

Tabela A.6: Descrição da máquina de estados *dk16*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 00      | $s_1$        | $s_3$          | 001   |
| 00      | $s_2$        | $s_1$          | 001   |
| 00      | $s_3$        | $s_4$          | 001   |
| 00      | $s_4$        | $s_4$          | 010   |
| 00      | $s_5$        | $s_1$          | 010   |
| 00      | $s_6$        | $s_3$          | 010   |
| 00      | $s_7$        | $s_9$          | 010   |
| 00      | $s_8$        | $s_{15}$       | 010   |
| 00      | $s_9$        | $s_1$          | 000   |
| 00      | $s_{10}$     | $s_{14}$       | 000   |
| 00      | $s_{11}$     | $s_3$          | 000   |
| 00      | $s_{12}$     | $s_{20}$       | 000   |
| 00      | $s_{13}$     | $s_3$          | 101   |
| 00      | $s_{14}$     | $s_1$          | 101   |
| 00      | $s_{15}$     | $s_4$          | 101   |
| 00      | $s_{16}$     | $s_{20}$       | 000   |
| 00      | $s_{17}$     | $s_{15}$       | 010   |
| 00      | $s_{18}$     | $s_4$          | 100   |
| 00      | $s_{19}$     | $s_{18}$       | 100   |
| 00      | $s_{20}$     | $s_{19}$       | 100   |
| 00      | $s_{21}$     | $s_2$          | 100   |
| 00      | $s_{22}$     | $s_3$          | 000   |
| 00      | $s_{23}$     | $s_2$          | 100   |
| 00      | $s_{24}$     | $s_{14}$       | 000   |
| 00      | $s_{25}$     | $s_{15}$       | 010   |
| 00      | $s_{26}$     | $s_{20}$       | 000   |
| 00      | $s_{27}$     | $s_{15}$       | 010   |
| 01      | $s_1$        | $s_{10}$       | 001   |
| 01      | $s_2$        | $s_2$          | 001   |
| 01      | $s_3$        | $s_5$          | 001   |
| 01      | $s_4$        | $s_5$          | 010   |
| 01      | $s_5$        | $s_2$          | 010   |
| 01      | $s_6$        | $s_{21}$       | 010   |
| 01      | $s_7$        | $s_{18}$       | 010   |
| 01      | $s_8$        | $s_{26}$       | 000   |
| 01      | $s_9$        | $s_5$          | 000   |
| 01      | $s_{10}$     | $s_{13}$       | 000   |
| 01      | $s_{11}$     | $s_{23}$       | 000   |
| 01      | $s_{12}$     | $s_{19}$       | 000   |
| 01      | $s_{13}$     | $s_{10}$       | 101   |
| 01      | $s_{14}$     | $s_2$          | 101   |
| 01      | $s_{15}$     | $s_5$          | 101   |
| 01      | $s_{16}$     | $s_{19}$       | 000   |
| 01      | $s_{17}$     | $s_{23}$       | 000   |
| 01      | $s_{18}$     | $s_5$          | 010   |
| 01      | $s_{19}$     | $s_{23}$       | 010   |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 01      | $s_{20}$     | $s_{20}$       | 010   |
| 01      | $s_{21}$     | $s_1$          | 010   |
| 01      | $s_{22}$     | $s_3$          | 010   |
| 01      | $s_{23}$     | $s_1$          | 010   |
| 01      | $s_{24}$     | $s_{13}$       | 000   |
| 01      | $s_{25}$     | $s_3$          | 010   |
| 01      | $s_{26}$     | $s_{19}$       | 000   |
| 01      | $s_{27}$     | $s_3$          | 010   |
| 10      | $s_1$        | $s_{11}$       | 001   |
| 10      | $s_2$        | $s_8$          | 001   |
| 10      | $s_3$        | $s_6$          | 001   |
| 10      | $s_4$        | $s_6$          | 010   |
| 10      | $s_5$        | $s_{16}$       | 010   |
| 10      | $s_6$        | $s_{10}$       | 010   |
| 10      | $s_7$        | $s_{19}$       | 010   |
| 10      | $s_8$        | $s_{13}$       | 010   |
| 10      | $s_9$        | $s_6$          | 000   |
| 10      | $s_{10}$     | $s_1$          | 000   |
| 10      | $s_{11}$     | $s_{24}$       | 000   |
| 10      | $s_{12}$     | $s_{18}$       | 000   |
| 10      | $s_{13}$     | $s_{11}$       | 101   |
| 10      | $s_{14}$     | $s_8$          | 101   |
| 10      | $s_{15}$     | $s_6$          | 101   |
| 10      | $s_{16}$     | $s_{13}$       | 010   |
| 10      | $s_{17}$     | $s_{18}$       | 000   |
| 10      | $s_{18}$     | $s_6$          | 100   |
| 10      | $s_{19}$     | $s_{24}$       | 100   |
| 10      | $s_{20}$     | $s_9$          | 100   |
| 10      | $s_{21}$     | $s_{13}$       | 100   |
| 10      | $s_{22}$     | $s_{15}$       | 100   |
| 10      | $s_{23}$     | $s_{13}$       | 010   |
| 10      | $s_{24}$     | $s_{13}$       | 100   |
| 10      | $s_{25}$     | $s_{15}$       | 000   |
| 10      | $s_{26}$     | $s_{18}$       | 000   |
| 10      | $s_{27}$     | $s_{13}$       | 100   |
| 11      | $s_1$        | $s_{12}$       | 001   |
| 11      | $s_2$        | $s_9$          | 001   |
| 11      | $s_3$        | $s_7$          | 001   |
| 11      | $s_4$        | $s_7$          | 010   |
| 11      | $s_5$        | $s_{17}$       | 010   |
| 11      | $s_6$        | $s_{22}$       | 010   |
| 11      | $s_7$        | $s_{20}$       | 010   |
| 11      | $s_8$        | $s_{14}$       | 010   |
| 11      | $s_9$        | $s_7$          | 000   |
| 11      | $s_{10}$     | $s_2$          | 000   |
| 11      | $s_{11}$     | $s_{25}$       | 000   |
| 11      | $s_{12}$     | $s_{15}$       | 000   |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 11      | $s_{13}$     | $s_{12}$       | 101   |
| 11      | $s_{14}$     | $s_9$          | 101   |
| 11      | $s_{15}$     | $s_7$          | 101   |
| 11      | $s_{16}$     | $s_{14}$       | 010   |
| 11      | $s_{17}$     | $s_{27}$       | 000   |
| 11      | $s_{18}$     | $s_7$          | 100   |
| 11      | $s_{19}$     | $s_{25}$       | 100   |
| 11      | $s_{20}$     | $s_{26}$       | 100   |
| 11      | $s_{21}$     | $s_{14}$       | 100   |
| 11      | $s_{22}$     | $s_{15}$       | 000   |
| 11      | $s_{23}$     | $s_{14}$       | 010   |
| 11      | $s_{24}$     | $s_{14}$       | 100   |
| 11      | $s_{25}$     | $s_{15}$       | 000   |
| 11      | $s_{26}$     | $s_{21}$       | 000   |
| 11      | $s_{27}$     | $s_{14}$       | 100   |

## A.7 Máquina de estados: $dk27$

Esta máquina de estados possui 7 estados e 14 transições de estado. A Tabela A.7 mostra a função de transição de estados da máquina.

Tabela A.7: Descrição da máquina de estados  $dk27$

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 0       | $s_0$        | $s_6$          | 00    |
| 0       | $s_2$        | $s_5$          | 00    |
| 0       | $s_3$        | $s_5$          | 00    |
| 0       | $s_4$        | $s_6$          | 00    |
| 0       | $s_5$        | $s_0$          | 10    |
| 0       | $s_6$        | $s_0$          | 01    |
| 0       | $s_7$        | $s_5$          | 00    |
| 1       | $s_6$        | $s_2$          | 01    |
| 1       | $s_5$        | $s_2$          | 10    |
| 1       | $s_4$        | $s_6$          | 10    |
| 1       | $s_7$        | $s_6$          | 10    |
| 1       | $s_0$        | $s_4$          | 00    |
| 1       | $s_2$        | $s_3$          | 00    |
| 1       | $s_3$        | $s_7$          | 00    |

## A.8 Máquina de estados: $dk512$

Esta máquina de estados possui 15 estados e 30 transições de estado. A Tabela A.8 mostra a função de transição de estados da máquina.

Tabela A.8: Descrição da máquina de estados *dk512*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 0       | $s_1$        | $s_8$          | 000   |
| 0       | $s_2$        | $s_4$          | 000   |
| 0       | $s_3$        | $s_5$          | 000   |
| 0       | $s_4$        | $s_8$          | 000   |
| 0       | $s_5$        | $s_8$          | 000   |
| 0       | $s_6$        | $s_{13}$       | 000   |
| 0       | $s_7$        | $s_4$          | 000   |
| 0       | $s_8$        | $s_1$          | 001   |
| 0       | $s_9$        | $s_4$          | 000   |
| 0       | $s_{10}$     | $s_1$          | 010   |
| 0       | $s_{11}$     | $s_3$          | 010   |
| 0       | $s_{12}$     | $s_4$          | 100   |
| 0       | $s_{13}$     | $s_5$          | 100   |
| 0       | $s_{14}$     | $s_3$          | 100   |
| 0       | $s_{15}$     | $s_4$          | 000   |
| 1       | $s_1$        | $s_9$          | 000   |
| 1       | $s_2$        | $s_3$          | 000   |
| 1       | $s_3$        | $s_6$          | 000   |
| 1       | $s_4$        | $s_{11}$       | 000   |
| 1       | $s_5$        | $s_{12}$       | 000   |
| 1       | $s_6$        | $s_{14}$       | 000   |
| 1       | $s_7$        | $s_{15}$       | 000   |
| 1       | $s_8$        | $s_2$          | 001   |
| 1       | $s_9$        | $s_3$          | 001   |
| 1       | $s_{10}$     | $s_2$          | 010   |
| 1       | $s_{11}$     | $s_4$          | 010   |
| 1       | $s_{12}$     | $s_3$          | 001   |
| 1       | $s_{13}$     | $s_6$          | 100   |
| 1       | $s_{14}$     | $s_7$          | 100   |
| 1       | $s_{15}$     | $s_6$          | 000   |

## A.9 Máquina de estados: *donfile*

Esta máquina de estados possui 24 estados e 96 transições de estado. A Tabela A.9 mostra a função de transição de estados da máquina.

Tabela A.9: Descrição da máquina de estados *donfile*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 00      | $s_0$        | $s_0$          | 1     |
| 01      | $s_0$        | $s_6$          | 1     |
| 10      | $s_0$        | $s_{12}$       | 1     |
| 11      | $s_0$        | $s_{18}$       | 1     |
| 00      | $s_1$        | $s_1$          | 1     |
| 01      | $s_1$        | $s_7$          | 1     |
| 10      | $s_1$        | $s_{12}$       | 1     |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 11      | $s_1$        | $s_{18}$       | 1     |
| 00      | $s_2$        | $s_2$          | 1     |
| 01      | $s_2$        | $s_6$          | 1     |
| 10      | $s_2$        | $s_{12}$       | 1     |
| 11      | $s_2$        | $s_{19}$       | 1     |
| 00      | $s_3$        | $s_3$          | 1     |
| 01      | $s_3$        | $s_6$          | 1     |
| 10      | $s_3$        | $s_{13}$       | 1     |
| 11      | $s_3$        | $s_{19}$       | 1     |
| 00      | $s_4$        | $s_4$          | 1     |
| 01      | $s_4$        | $s_7$          | 1     |
| 10      | $s_4$        | $s_{13}$       | 1     |
| 11      | $s_4$        | $s_{18}$       | 1     |
| 00      | $s_5$        | $s_5$          | 1     |
| 01      | $s_5$        | $s_7$          | 1     |
| 10      | $s_5$        | $s_{13}$       | 1     |
| 11      | $s_5$        | $s_{19}$       | 1     |
| 00      | $s_6$        | $s_0$          | 1     |
| 01      | $s_6$        | $s_6$          | 1     |
| 10      | $s_6$        | $s_{14}$       | 1     |
| 11      | $s_6$        | $s_{20}$       | 1     |
| 00      | $s_7$        | $s_1$          | 1     |
| 01      | $s_7$        | $s_7$          | 1     |
| 10      | $s_7$        | $s_{14}$       | 1     |
| 11      | $s_7$        | $s_{20}$       | 1     |
| 00      | $s_8$        | $s_0$          | 1     |
| 01      | $s_8$        | $s_8$          | 1     |
| 10      | $s_8$        | $s_{14}$       | 1     |
| 11      | $s_8$        | $s_{21}$       | 1     |
| 00      | $s_9$        | $s_0$          | 1     |
| 01      | $s_9$        | $s_9$          | 1     |
| 10      | $s_9$        | $s_{15}$       | 1     |
| 11      | $s_9$        | $s_{21}$       | 1     |
| 00      | $s_{10}$     | $s_1$          | 1     |
| 01      | $s_{10}$     | $s_{10}$       | 1     |
| 10      | $s_{10}$     | $s_{15}$       | 1     |
| 11      | $s_{10}$     | $s_{20}$       | 1     |
| 00      | $s_{11}$     | $s_1$          | 1     |
| 01      | $s_{11}$     | $s_{11}$       | 1     |
| 10      | $s_{11}$     | $s_{15}$       | 1     |
| 11      | $s_{11}$     | $s_{21}$       | 1     |
| 00      | $s_{12}$     | $s_2$          | 1     |
| 01      | $s_{12}$     | $s_8$          | 1     |
| 10      | $s_{12}$     | $s_{12}$       | 1     |
| 11      | $s_{12}$     | $s_{22}$       | 1     |
| 00      | $s_{13}$     | $s_3$          | 1     |
| 01      | $s_{13}$     | $s_8$          | 1     |
| 10      | $s_{13}$     | $s_{13}$       | 1     |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 11      | $s_{13}$     | $s_{22}$       | 1     |
| 00      | $s_{14}$     | $s_2$          | 1     |
| 01      | $s_{14}$     | $s_8$          | 1     |
| 10      | $s_{14}$     | $s_{14}$       | 1     |
| 11      | $s_{14}$     | $s_{23}$       | 1     |
| 00      | $s_{15}$     | $s_2$          | 1     |
| 01      | $s_{15}$     | $s_9$          | 1     |
| 10      | $s_{15}$     | $s_{15}$       | 1     |
| 11      | $s_{15}$     | $s_{23}$       | 1     |
| 00      | $s_{16}$     | $s_3$          | 1     |
| 01      | $s_{16}$     | $s_9$          | 1     |
| 10      | $s_{16}$     | $s_{16}$       | 1     |
| 11      | $s_{16}$     | $s_{22}$       | 1     |
| 00      | $s_{17}$     | $s_3$          | 1     |
| 01      | $s_{17}$     | $s_9$          | 1     |
| 10      | $s_{17}$     | $s_{17}$       | 1     |
| 11      | $s_{17}$     | $s_{23}$       | 1     |
| 00      | $s_{18}$     | $s_4$          | 1     |
| 01      | $s_{18}$     | $s_{10}$       | 1     |
| 10      | $s_{18}$     | $s_{16}$       | 1     |
| 11      | $s_{18}$     | $s_{18}$       | 1     |
| 00      | $s_{19}$     | $s_5$          | 1     |
| 01      | $s_{19}$     | $s_{10}$       | 1     |
| 10      | $s_{19}$     | $s_{16}$       | 1     |
| 11      | $s_{19}$     | $s_{19}$       | 1     |
| 00      | $s_{20}$     | $s_4$          | 1     |
| 01      | $s_{20}$     | $s_{10}$       | 1     |
| 10      | $s_{20}$     | $s_{17}$       | 1     |
| 11      | $s_{20}$     | $s_{20}$       | 1     |
| 00      | $s_{21}$     | $s_4$          | 1     |
| 01      | $s_{21}$     | $s_{11}$       | 1     |
| 10      | $s_{21}$     | $s_{17}$       | 1     |
| 11      | $s_{21}$     | $s_{21}$       | 1     |
| 00      | $s_{22}$     | $s_5$          | 1     |
| 01      | $s_{22}$     | $s_{11}$       | 1     |
| 10      | $s_{22}$     | $s_{16}$       | 1     |
| 11      | $s_{22}$     | $s_{22}$       | 1     |
| 00      | $s_{23}$     | $s_5$          | 1     |
| 01      | $s_{23}$     | $s_{11}$       | 1     |
| 10      | $s_{23}$     | $s_{17}$       | 1     |
| 11      | $s_{23}$     | $s_{23}$       | 1     |

## A.10 Máquina de estados: *lion9*

Esta máquina de estados possui 9 estados e 25 transições de estado. A Tabela A.10 mostra a função de transição de estados da máquina.

Tabela A.10: Descrição da máquina de estados *lion9*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 10      | $s_0$        | $s_1$          | 0     |
| 00      | $s_0$        | $s_0$          | 0     |
| 00      | $s_1$        | $s_0$          | 0     |
| 10      | $s_1$        | $s_1$          | 0     |
| 11      | $s_1$        | $s_2$          | 0     |
| 10      | $s_2$        | $s_1$          | 0     |
| 11      | $s_2$        | $s_2$          | 0     |
| 01      | $s_2$        | $s_3$          | 0     |
| 11      | $s_3$        | $s_2$          | 1     |
| 01      | $s_3$        | $s_3$          | 1     |
| 00      | $s_3$        | $s_4$          | 1     |
| 01      | $s_4$        | $s_3$          | 1     |
| 00      | $s_4$        | $s_4$          | 1     |
| 10      | $s_4$        | $s_5$          | 1     |
| 00      | $s_5$        | $s_4$          | 1     |
| 10      | $s_5$        | $s_5$          | 1     |
| 11      | $s_5$        | $s_6$          | 1     |
| 10      | $s_6$        | $s_5$          | 1     |
| 11      | $s_6$        | $s_6$          | 1     |
| 01      | $s_6$        | $s_7$          | 1     |
| 11      | $s_7$        | $s_6$          | 1     |
| 01      | $s_7$        | $s_7$          | 1     |
| 00      | $s_7$        | $s_8$          | 1     |
| 01      | $s_8$        | $s_7$          | 1     |
| 00      | $s_8$        | $s_8$          | 1     |

### A.11 Máquina de estados: *modulo12*

Esta máquina de estados 12 estados e 24 transições de estado. A Tabela A.11 mostra a função de transição de estados da máquina.

Tabela A.11: Descrição da máquina de estados *modulo12*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 0       | $s_0$        | $s_0$          | 0     |
| 1       | $s_0$        | $s_1$          | 0     |
| 0       | $s_1$        | $s_1$          | 0     |
| 1       | $s_1$        | $s_2$          | 0     |
| 0       | $s_2$        | $s_2$          | 0     |
| 1       | $s_2$        | $s_3$          | 0     |
| 0       | $s_3$        | $s_3$          | 0     |
| 1       | $s_3$        | $s_4$          | 0     |
| 0       | $s_4$        | $s_4$          | 0     |
| 1       | $s_4$        | $s_5$          | 0     |
| 0       | $s_5$        | $s_5$          | 0     |
| 1       | $s_5$        | $s_6$          | 0     |
| 0       | $s_6$        | $s_6$          | 0     |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 1       | $s_6$        | $s_7$          | 0     |
| 0       | $s_7$        | $s_7$          | 0     |
| 1       | $s_7$        | $s_8$          | 0     |
| 0       | $s_8$        | $s_8$          | 0     |
| 1       | $s_8$        | $s_9$          | 0     |
| 0       | $s_9$        | $s_9$          | 0     |
| 1       | $s_9$        | $s_{10}$       | 0     |
| 0       | $s_{10}$     | $s_{10}$       | 0     |
| 1       | $s_{10}$     | $s_{11}$       | 0     |
| 0       | $s_{11}$     | $s_{11}$       | 0     |
| 1       | $s_{11}$     | $s_0$          | 0     |

### A.12 Máquina de estados: *shiftreg*

Esta máquina de estados possui 8 estados e 16 transições de estado. A Tabela A.12 mostra a função de transição de estados da máquina.

Tabela A.12: Descrição da máquina de estados *shiftreg*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 0       | $s_0$        | $s_0$          | 0     |
| 1       | $s_0$        | $s_4$          | 0     |
| 0       | $s_1$        | $s_0$          | 1     |
| 1       | $s_1$        | $s_4$          | 1     |
| 0       | $s_2$        | $s_1$          | 0     |
| 1       | $s_2$        | $s_5$          | 0     |
| 0       | $s_3$        | $s_1$          | 1     |
| 1       | $s_3$        | $s_5$          | 1     |
| 0       | $s_4$        | $s_2$          | 0     |
| 1       | $s_4$        | $s_6$          | 0     |
| 0       | $s_5$        | $s_2$          | 1     |
| 1       | $s_5$        | $s_6$          | 1     |
| 0       | $s_6$        | $s_3$          | 0     |
| 1       | $s_6$        | $s_7$          | 0     |
| 0       | $s_7$        | $s_3$          | 1     |
| 1       | $s_7$        | $s_7$          | 1     |

### A.13 Máquina de estados: *tav*

Esta máquina de estados possui 4 estados e 49 transições de estado. A Tabela A.13 mostra a função de transição de estados da máquina.

Tabela A.13: Descrição da máquina de estados *tav*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 1000    | $s_0$        | $s_1$          | 1000  |
| 0100    | $s_0$        | $s_1$          | 0100  |
| 0010    | $s_0$        | $s_1$          | 0010  |

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 0001    | $s_0$        | $s_1$          | 0001  |
| 0000    | $s_0$        | $s_1$          | 0000  |
| 11--    | $s_0$        | $s_1$          | 0000  |
| 1-1-    | $s_0$        | $s_1$          | 0000  |
| 1--1    | $s_0$        | $s_1$          | 0000  |
| -11-    | $s_0$        | $s_1$          | 0000  |
| -1-1    | $s_0$        | $s_1$          | 0000  |
| --11    | $s_0$        | $s_1$          | 0000  |
| 1000    | $s_1$        | $s_2$          | 1000  |
| 0100    | $s_1$        | $s_2$          | 0100  |
| 0010    | $s_1$        | $s_2$          | 0010  |
| 0001    | $s_1$        | $s_2$          | 0001  |
| 1100    | $s_1$        | $s_2$          | 1100  |
| 1010    | $s_1$        | $s_2$          | 1010  |
| 1001    | $s_1$        | $s_2$          | 1001  |
| 0110    | $s_1$        | $s_2$          | 0000  |
| 0000    | $s_1$        | $s_2$          | 0000  |
| 0011    | $s_1$        | $s_2$          | 0011  |
| 0101    | $s_1$        | $s_2$          | 0101  |
| 0111    | $s_1$        | $s_2$          | 0001  |
| 1011    | $s_1$        | $s_2$          | 1011  |
| 1101    | $s_1$        | $s_2$          | 1101  |
| 1110    | $s_1$        | $s_2$          | 1000  |
| 1111    | $s_1$        | $s_2$          | 1001  |
| 1000    | $s_2$        | $s_3$          | 1000  |
| 0100    | $s_2$        | $s_3$          | 0100  |
| 0010    | $s_2$        | $s_3$          | 0010  |
| 0001    | $s_2$        | $s_3$          | 0001  |
| 0000    | $s_2$        | $s_3$          | 0000  |
| 11--    | $s_2$        | $s_3$          | 0000  |
| 1-1-    | $s_2$        | $s_3$          | 0000  |
| 1--1    | $s_2$        | $s_3$          | 0000  |
| -11-    | $s_2$        | $s_3$          | 0000  |
| -1-1    | $s_2$        | $s_3$          | 0000  |
| --11    | $s_2$        | $s_3$          | 0000  |
| 1000    | $s_3$        | $s_0$          | 1000  |
| 0100    | $s_3$        | $s_0$          | 0100  |
| 0010    | $s_3$        | $s_0$          | 0010  |
| 0001    | $s_3$        | $s_0$          | 0001  |
| 0000    | $s_3$        | $s_0$          | 0000  |
| 11--    | $s_3$        | $s_0$          | 0000  |
| 1-1-    | $s_3$        | $s_0$          | 0000  |
| 1--1    | $s_3$        | $s_0$          | 0000  |
| -11-    | $s_3$        | $s_0$          | 0000  |
| -1-1    | $s_3$        | $s_0$          | 0000  |
| --11    | $s_3$        | $s_0$          | 0000  |

## A.14 Máquina de estados: *train11*

Esta máquina de estados possui 11 estados e 25 transições de estado. A Tabela A.14 mostra a função de transição de estados da máquina.

Tabela A.14: Descrição da máquina de estados *train11*

| Entrada | Estado atual | Próximo estado | Saída |
|---------|--------------|----------------|-------|
| 00      | $s_0$        | $s_0$          | 0     |
| 10      | $s_0$        | $s_1$          | -     |
| 01      | $s_0$        | $s_2$          | -     |
| 10      | $s_1$        | $s_1$          | 1     |
| 00      | $s_1$        | $s_3$          | 1     |
| 11      | $s_1$        | $s_5$          | 1     |
| 01      | $s_2$        | $s_2$          | 1     |
| 00      | $s_2$        | $s_7$          | 1     |
| 11      | $s_2$        | $s_9$          | 1     |
| 00      | $s_3$        | $s_3$          | 1     |
| 01      | $s_3$        | $s_4$          | 1     |
| 01      | $s_4$        | $s_4$          | 1     |
| 00      | $s_4$        | $s_0$          | -     |
| 11      | $s_5$        | $s_5$          | 1     |
| 01      | $s_5$        | $s_6$          | 1     |
| 01      | $s_6$        | $s_6$          | 1     |
| 00      | $s_6$        | $s_0$          | -     |
| 00      | $s_7$        | $s_7$          | 1     |
| 10      | $s_7$        | $s_8$          | 1     |
| 10      | $s_8$        | $s_8$          | 1     |
| 00      | $s_8$        | $s_0$          | -     |
| 11      | $s_9$        | $s_9$          | 1     |
| 10      | $s_9$        | $s_{10}$       | 1     |
| 10      | $s_{10}$     | $s_{10}$       | 1     |
| 00      | $s_{10}$     | $s_0$          | -     |

# APÊNDICE B – Lógicas Evoluídas para as Máquinas de Referência

## B.1 Máquina de estados: *bbara*

Esta máquina de estado tem 4 sinais da entrada primária  $i_3 \dots i_0$ , 2 sinais da saída primária  $s_1 \dots s_0$ , 10 estados (i.e. 4 sinais de estado, sendo 4 sinais da entrada secundária  $e_3 \dots e_0$  e 4 sinais da saída secundária  $e_3^+ \dots e_0^+$ ) e 60 transições de estado. A máquina foi evoluída com uma geometria  $32 \times 5$ . A atribuição de estados [4, 5, 1, 9, 13, 12, 14, 15, 7, 6] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 1.22005 ns considerando o fator de carga das portas e de 0.885 ns sem este fator. As equações de controle dos sinais de próximo estado e dos sinais da saída primária são mostradas a seguir:

$$e_3^+ \Leftarrow \text{OR}(\text{NOR}(\text{AND}(\text{XNOR}(i_3, i_2), \overline{e_3}), \text{XOR}(\text{AND}(i_0, i_1), \text{NOR}(e_0, i_3))), \text{AND}(\text{AND}(\text{OR}(i_3, e_3), \text{AND}(i_0, i_1)), \text{NAND}(\text{XNOR}(i_3, i_2), e_2))))$$

$$e_2^+ \Leftarrow \text{NAND}(\text{NOR}(\text{NAND}(i_1, i_0), \text{XOR}(\text{NAND}(e_0, e_3), \text{NAND}(e_1, i_0))), \text{OR}(\text{AND}(i_1, i_0), \overline{i_0}))$$

$$e_1^+ \Leftarrow \text{XNOR}(\text{AND}(\text{XOR}(\overline{i_2}, \text{NAND}(e_0, i_3)), \text{NOR}(\text{XNOR}(\text{NAND}(e_0, i_3), \text{NOR}(e_3, e_0)), i_2)), i_2)$$

$$e_0^+ \Leftarrow \text{NOR}(\text{NOR}(\text{OR}(\text{AND}(e_0, e_1), \text{NAND}(e_2, e_3)), \text{XNOR}(\text{OR}(i_3, e_0), i_2)), \text{XNOR}(\text{NOR}(\overline{e_2}, \text{OR}(i_3, e_3)), \text{NAND}(\overline{e_0}, \text{NAND}(i_1, i_0))))))$$

$$s_1 \Leftarrow \text{NOR}(e_2, \text{NAND}(i_2, e_3))$$

$$s_0 \Leftarrow \text{NOT}(\text{OR}(\text{OR}(e_0, \text{NAND}(e_1, e_3)), \text{NOR}(\text{NAND}(i_1, i_0), \text{XOR}(i_3, i_2))))$$

## B.2 Máquina de estados: *bbtas*

Esta máquina de estado tem 2 sinais da entrada primária  $i_1 \dots i_0$ , 2 sinais da saída primária  $s_1 \dots s_0$ , 6 estados (i.e. 3 sinais de estado, sendo 3 sinais da entrada secundária  $e_2 \dots e_0$  e 3 sinais da saída secundária  $e_2^+ \dots e_0^+$ ) e 24 transições de estado. A máquina foi evoluída com uma geometria  $16 \times 4$ . A atribuição de estados [3, 7, 1, 5, 4, 6] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 1.1367 ns considerando o fator de carga das

portas e de 0.73 ns sem este fator. As equações de controle dos sinais de próximo estado e dos sinais da saída primária são mostradas a seguir:

$$\begin{aligned}
 e_2^+ &\Leftarrow \text{NAND}(\text{NAND}(\text{NAND}(e_0, e_2), \text{OR}(i_0, i_1)), e_1) \\
 e_1^+ &\Leftarrow \text{NOR}(\text{AND}(\text{NAND}(\text{NOR}(i_1, i_0), \text{NAND}(e_2, e_0)), \overline{e_1}), \text{NOR}(\text{NAND}(e_2, e_0), \text{NOR}(i_1, i_0))) \\
 e_0^+ &\Leftarrow \text{XNOR}(\overline{e_0}, \text{NOR}(\text{NOT}(\text{NOR}(i_1, i_0)), \text{NAND}(e_2, \text{XOR}(e_0, e_1)))) \\
 s_1 &\Leftarrow \text{NOR}(\text{NAND}(i_1, \overline{e_1}), \text{NAND}(e_0, e_2)) \\
 s_0 &\Leftarrow \text{NOR}(\text{NAND}(e_0, e_2), \text{NAND}(i_0, \overline{e_1}))
 \end{aligned}$$

### B.3 Máquina de estados: *dk15*

Esta máquina de estado tem 3 sinais da entrada primária  $i_2 \dots i_0$ , 5 sinais da saída primária  $s_4 \dots s_0$ , 4 estados (i.e. 2 sinais de estado, sendo 2 sinais da entrada secundária  $e_1 \dots e_0$  e 2 sinais da saída secundária  $e_1^+ \dots e_0^+$ ) e 32 transições de estado. A máquina foi evoluída com uma geometria  $32 \times 4$ . A atribuição de estados  $[0, 2, 3, 1]$  foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 1.2977 ns considerando o fator de carga das portas e de 0.92 ns sem este fator. As equações de controle dos sinais de próximo estado e dos sinais da saída primária são mostradas a seguir:

$$\begin{aligned}
 e_1^+ &\Leftarrow \text{NAND}(\text{OR}(\text{NOT}(\text{NAND}(e_0, i_2)), \text{XNOR}(\overline{e_0}, \text{NOR}(i_0, e_1))), \text{OR}(\text{NOR}(\text{NAND}(e_1, i_2), i_0), \\
 &\quad \text{XNOR}(i_0, i_1))) \\
 e_0^+ &\Leftarrow \text{XOR}(\text{NAND}(\text{NOT}(\text{NOR}(i_1, i_0)), \overline{i_0}), \text{OR}(\text{OR}(\text{XNOR}(e_0, i_2), \text{NOR}(i_1, e_1)), \text{XNOR}(\text{OR}(i_1, e_0), \\
 &\quad \text{NAND}(i_2, i_0)))) \\
 s_4 &\Leftarrow \text{XNOR}(\text{NOR}(\text{NAND}(\overline{i_2}, \text{OR}(e_0, e_1)), \text{NOR}(\overline{i_1}, \text{NOR}(e_0, i_0))), \text{NAND}(\text{NAND}(\text{XNOR}(i_0, e_0), \\
 &\quad \text{OR}(i_1, i_2)), \text{XOR}(\text{NAND}(e_1, e_0), \overline{i_1}))) \\
 s_3 &\Leftarrow \text{NOR}(\text{NOT}(\text{AND}(\text{NAND}(i_1, i_0), i_2)), \text{NOR}(\text{NAND}(\text{XOR}(i_1, e_1), e_0), i_0)) \\
 s_2 &\Leftarrow \text{NOR}(\text{XNOR}(\text{NAND}(\overline{i_1}, \text{XNOR}(e_1, e_0)), \overline{i_0}), \text{NOR}(\text{NOR}(\overline{e_0}, \text{NOR}(e_1, i_1)), \text{XOR}(i_2, \overline{i_1}))) \\
 s_1 &\Leftarrow \text{XNOR}(\text{NOR}(\text{XNOR}(i_1, i_0), \text{NOR}(\text{XNOR}(i_1, e_1), \text{NAND}(i_2, e_0))), \text{OR}(\text{XNOR}(e_1, e_0), \\
 &\quad \text{OR}(i_1, i_2))) \\
 s_0 &\Leftarrow \text{NOR}(\text{NOT}(\text{XNOR}(i_0, i_1)), \text{XOR}(\text{NOR}(\text{NOR}(e_0, e_1), \text{OR}(i_2, i_0)), \text{NOR}(\text{NOR}(e_1, i_0), \overline{e_0})))
 \end{aligned}$$

### B.4 Máquina de estados: *dk27*

Esta máquina de estado tem 1 sinal da entrada primária  $i$ , 2 sinais da saída primária  $s_1 \dots s_0$ , 7 estados (i.e. 3 sinais de estado, sendo 3 sinais da entrada secundária  $e_2 \dots e_0$  e 3 sinais da saída

secundária  $e_2^+ \dots e_0^+$ ) e 14 transições de estado. A máquina foi evoluída com uma geometria  $16 \times 4$ . A atribuição de estados [2, 0, 1, 6, 5, 7, 4] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 0.55825 ns considerando o fator de carga das portas e de 0.43 ns sem este fator. As equações de controle dos sinais de próximo estado e dos sinais da saída primária são mostradas a seguir:

$$e_2^+ \Leftarrow \text{NOR}(\text{NOR}(\text{NAND}(\text{NOR}(e_0, e_2), i), e_1), \text{AND}(e_0, e_2))$$

$$e_1^+ \Leftarrow \text{NAND}(\text{NAND}(e_2, \text{XOR}(e_0, i)), \text{NAND}(e_1, \bar{e}_0))$$

$$e_0^+ \Leftarrow \text{AND}(\text{NAND}(\text{OR}(e_2, i), e_0), \text{NAND}(\text{NOR}(e_2, \bar{e}_1), i))$$

$$s_1 \Leftarrow \text{NOR}(\text{NAND}(\text{NAND}(e_0, e_1), e_2), \text{NOR}(i, e_0))$$

$$s_0 \Leftarrow \text{NOT}(\text{NAND}(e_0, e_1))$$

## B.5 Máquina de estados: *dk512*

Esta máquina de estado tem 1 sinal da entrada primária  $i$ , 3 sinais da saída primária  $s_2 \dots s_0$ , 15 estados (i.e. 4 sinais de estado, sendo 4 sinais da entrada secundária  $e_3 \dots e_0$  e 4 sinais da saída secundária  $e_3^+ \dots e_0^+$ ) e 30 transições de estado. A máquina foi evoluída com uma geometria  $32 \times 5$ . A atribuição de estados [6, 4, 15, 14, 2, 10, 8, 3, 5, 11, 1, 0, 13, 9, 12] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 1.2924 ns considerando o fator de carga das portas e de 0.84 ns sem este fator. As equações de controle dos sinais de próximo estado e dos sinais da saída primária são mostradas a seguir:

$$e_3^+ \Leftarrow \bar{e}_1$$

$$e_2^+ \Leftarrow \text{XOR}(\text{OR}(\text{NOR}(\text{NOR}(e_2, \text{NOR}(e_1, e_3))), \text{NAND}(\text{AND}(i, e_1), \text{OR}(e_0, e_3))), \text{NOR}(\text{NOR}(e_1, e_3), \text{AND}(i, e_1))), \text{AND}(\text{NAND}(\text{XOR}(e_0, i), \text{NOR}(e_2, \text{NOR}(e_1, e_3))), \text{nand}(\text{NOR}(\text{OR}(i, e_0), \text{NOR}(e_1, e_3)), \text{AND}(\text{NAND}(e_2, e_1), \text{OR}(e_0, e_3)))))$$

$$e_1^+ \Leftarrow \text{OR}(\text{AND}(\text{NAND}(e_0, e_2), \text{NOR}(e_3, e_2)), \text{NOR}(\text{NAND}(e_0, e_2), \text{NOR}(e_3, e_2)))$$

$$e_0^+ \Leftarrow \text{NAND}(\text{XNOR}(\text{NOT}(\text{NOR}(e_0, i)), \text{XOR}(\text{NOR}(e_0, e_1), \text{NOR}(e_0, i))), \text{OR}(\text{NAND}(\bar{i}, \text{NAND}(e_3, i)), \bar{i}))$$

$$s_2 \Leftarrow \text{NOR}(\text{NOR}(\text{AND}(e_3, e_0), \text{NOR}(e_0, i)), \text{XOR}(\text{NAND}(\text{AND}(e_3, e_0), \bar{e}_1), \text{AND}(\bar{e}_1, \text{NOR}(e_3, e_2))))$$

$$s_1 \Leftarrow \text{NOR}(e_2, \text{NAND}(e_0, \text{XNOR}(e_1, e_3)))$$

$$s_0 \Leftarrow \text{NOR}(\text{NAND}(\bar{e}_3, \text{OR}(i, e_1)), (\text{XNOR}(e_0, \text{NOR}(e_1, e_2))))$$

## B.6 Máquina de estados: *lion9*

Esta máquina de estado tem 2 sinais da entrada primária  $i_1 \dots i_0$ , 1 sinal da saída primária  $s$ , 9 estados (i.e. 4 sinais de estado, sendo 4 sinais da entrada secundária  $e_3 \dots e_0$  e 4 sinais da saída secundária  $e_3^+ \dots e_0^+$ ) e 25 transições de estado. A máquina foi evoluída com uma geometria  $32 \times 4$ . A atribuição de estados [11, 9, 3, 1, 2, 0, 8, 10, 14] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 0.807 ns considerando o fator de carga das portas e de 0.52 ns sem este fator. As equações de controle dos sinais de próximo estado e do sinal da saída primária são mostradas a seguir:

$$\begin{aligned}
 e_3^+ &\Leftarrow \text{NOR}(\text{AND}(i_1, \text{XNOR}(e_0, i_0)), \text{NOR}(i_1, e_3)) \\
 e_2^+ &\Leftarrow \text{NOR}(\text{NAND}(\text{NOR}(i_0, e_0), e_3), i_1) \\
 e_1^+ &\Leftarrow \text{NOR}(\text{NOR}(\text{AND}(i_0, e_0), \overline{i_1}), \text{AND}(\text{NOR}(e_3, i_1), i_0)) \\
 e_0^+ &\Leftarrow \text{OR}(\text{NOR}(\text{OR}(i_1, e_3), \text{NOR}(i_0, \text{OR}(i_1, e_3))), \text{NOR}(\text{NOR}(i_0, \text{OR}(i_1, e_3)), \overline{e_0})) \\
 s &\Leftarrow \text{NAND}(e_0, \text{OR}(e_1, e_3))
 \end{aligned}$$

## B.7 Máquina de estados: *modulo12*

Esta máquina de estado tem 1 sinal da entrada primária  $i$ , 1 sinal da saída primária  $s$ , 12 estados (i.e. 4 sinais de estado, sendo 4 sinais da entrada secundária  $e_3 \dots e_0$  e 4 sinais da saída secundária  $e_3^+ \dots e_0^+$ ) e 24 transições de estado. A máquina foi evoluída com uma geometria  $16 \times 4$ . A atribuição de estados [15, 7, 6, 14, 10, 2, 3, 1, 5, 13, 9, 11] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 0.82435 ns considerando o fator de carga das portas e de 0.56 ns sem este fator. As equações de controle dos sinais de próximo estado e do sinal da saída primária são mostradas a seguir:

$$\begin{aligned}
 e_3^+ &\Leftarrow \text{NOT}(\text{NAND}(\text{OR}(\text{NAND}(e_1, i), \text{XOR}(e_2, e_0)), \text{NOR}(\text{NOR}(e_3, e_2), \text{NOR}(e_3, i)))) \\
 e_2^+ &\Leftarrow \text{NAND}(\text{NAND}(\text{XNOR}(\overline{e_3}), \text{NAND}(e_0, e_3)), i), \text{NOT}(\text{AND}(\text{NAND}(e_3, i), i))) \\
 e_1^+ &\Leftarrow \text{AND}(\text{OR}(\text{NOR}(\text{NAND}(e_2, i), e_2), e_3), \text{NAND}(\overline{e_3}), \text{NOR}(\text{NAND}(e_2, i), e_2))) \\
 e_0^+ &\Leftarrow \text{XOR}(\text{NOR}(e_0, \text{NOR}(i, e_3)), \text{NOR}(i, e_3)) \\
 s &\Leftarrow \text{NOR}(e_0, e_1)
 \end{aligned}$$

## B.8 Máquina de estados: *shiftrreg*

Esta máquina de estado tem 1 sinal da entrada primária  $i$ , 1 sinal da saída primária  $s$ , 8 estados (i.e. 3 sinais de estado, sendo 3 sinais da entrada secundária  $e_2 \dots e_0$  e 3 sinais da saída secundária  $e_2^+ \dots e_0^+$ ) e 16 transições de estado. A máquina foi evoluída com uma geometria  $4 \times 3$ . A atribuição de estados  $[6, 7, 4, 5, 2, 3, 0, 1]$  foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 0.035 ns considerando ou não o fator de carga das portas. As equações de controle dos sinais de próximo estado e do sinal da saída primária são mostradas a seguir:

$$e_2^+ \Leftarrow \bar{i}$$

$$e_1^+ \Leftarrow \bar{e}_0$$

$$e_0^+ \Leftarrow e_2$$

$$s \Leftarrow e_1$$

## B.9 Máquina de estados: *tav*

Esta máquina de estado tem 4 sinais da entrada primária  $i_3 \dots i_0$ , 4 sinais da saída primária  $s_3 \dots s_0$ , 4 estados (i.e. 2 sinais de estado, sendo 2 sinais da entrada secundária  $e_1 \dots e_0$  e 2 sinais da saída secundária  $e_1^+ \dots e_0^+$ ) e 49 transições de estado. A máquina foi evoluída com uma geometria  $32 \times 4$ . A atribuição de estados  $[1, 0, 3, 2]$  foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 0.63625 ns considerando o fator de carga das portas e de 0.325 ns sem este fator. As equações de controle dos sinais de próximo estado e dos sinais da saída primária são mostradas a seguir:

$$e_1^+ \Leftarrow \text{XNOR}(e_0, e_1)$$

$$e_0^+ \Leftarrow \bar{e}_0$$

$$s_3 \Leftarrow \text{NAND}(\text{NAND}(\bar{e}_1, \text{AND}(\bar{e}_0, i_3)), \text{NAND}(\text{NOR}(i_2, i_1), \text{AND}(\bar{i}_0, i_3)))$$

$$s_2 \Leftarrow \text{NOR}(\text{NOR}(\text{NOR}(i_3, i_0), \text{NOR}(e_1, e_0)), \text{NAND}(i_2, \bar{i}_1))$$

$$s_1 \Leftarrow \text{NOR}(\text{NOR}(\text{NOR}(i_0, i_3), \text{NOR}(e_1, e_0)), \text{NAND}(i_1, \bar{i}_2))$$

$$s_0 \Leftarrow \text{NAND}(\text{NAND}(i_0, \text{NOR}(e_0, e_1)), \text{OR}(\text{NAND}(i_0, \bar{i}_2), \text{NOT}(\text{NOR}(i_1, i_3))))$$

## B.10 Máquina de estados: *train11*

Esta máquina de estado tem 2 sinais da entrada primária  $i_1 \dots i_0$ , 1 sinal da saída primária  $s$ , 11 estados (i.e. 4 sinais de estado, sendo 4 sinais da entrada secundária  $e_3 \dots e_0$  e 4 sinais da saída secundária  $e_3^+ \dots e_0^+$ ) e 25 transições de estado. A máquina foi evoluída com uma geometria  $32 \times 4$ . A atribuição de estados [9, 11, 13, 3, 1, 2, 0, 12, 8, 5, 4] foi utilizada para a evolução do circuito. O atraso da lógica de controle foi de 0.7894 ns considerando o fator de carga das portas e de 0.52 ns sem este fator. As equações de controle dos sinais de próximo estado e do sinal da saída primária são mostradas a seguir:

$$e_3^+ \Leftarrow \text{NAND}(\text{OR}(\text{OR}(i_1, e_1), i_0), \text{XOR}(\text{NAND}(i_1, e_3), \text{AND}(e_3, i_0)))$$

$$e_2^+ \Leftarrow \text{NOR}(\text{OR}(\text{NOR}(e_2, i_0), e_1), \text{AND}(\text{NAND}(e_2, e_0), \text{XNOR}(i_1, e_3)))$$

$$e_1^+ \Leftarrow \text{NOR}(\overline{e_1}, \text{NOR}(i_1, \overline{i_0}))$$

$$e_0^+ \Leftarrow \text{NAND}(\text{NAND}(\text{NOR}(i_1, e_2), e_0), \text{NAND}(\text{OR}(e_0, \text{NOR}(i_1, e_2)), \text{XNOR}(e_2, i_0)))$$

$$s \Leftarrow \text{NAND}(\text{NOR}(e_1, e_2), \text{NOR}(i_1, i_0))$$

# Livros Grátis

( <http://www.livrosgratis.com.br> )

Milhares de Livros para Download:

[Baixar livros de Administração](#)

[Baixar livros de Agronomia](#)

[Baixar livros de Arquitetura](#)

[Baixar livros de Artes](#)

[Baixar livros de Astronomia](#)

[Baixar livros de Biologia Geral](#)

[Baixar livros de Ciência da Computação](#)

[Baixar livros de Ciência da Informação](#)

[Baixar livros de Ciência Política](#)

[Baixar livros de Ciências da Saúde](#)

[Baixar livros de Comunicação](#)

[Baixar livros do Conselho Nacional de Educação - CNE](#)

[Baixar livros de Defesa civil](#)

[Baixar livros de Direito](#)

[Baixar livros de Direitos humanos](#)

[Baixar livros de Economia](#)

[Baixar livros de Economia Doméstica](#)

[Baixar livros de Educação](#)

[Baixar livros de Educação - Trânsito](#)

[Baixar livros de Educação Física](#)

[Baixar livros de Engenharia Aeroespacial](#)

[Baixar livros de Farmácia](#)

[Baixar livros de Filosofia](#)

[Baixar livros de Física](#)

[Baixar livros de Geociências](#)

[Baixar livros de Geografia](#)

[Baixar livros de História](#)

[Baixar livros de Línguas](#)

[Baixar livros de Literatura](#)  
[Baixar livros de Literatura de Cordel](#)  
[Baixar livros de Literatura Infantil](#)  
[Baixar livros de Matemática](#)  
[Baixar livros de Medicina](#)  
[Baixar livros de Medicina Veterinária](#)  
[Baixar livros de Meio Ambiente](#)  
[Baixar livros de Meteorologia](#)  
[Baixar Monografias e TCC](#)  
[Baixar livros Multidisciplinar](#)  
[Baixar livros de Música](#)  
[Baixar livros de Psicologia](#)  
[Baixar livros de Química](#)  
[Baixar livros de Saúde Coletiva](#)  
[Baixar livros de Serviço Social](#)  
[Baixar livros de Sociologia](#)  
[Baixar livros de Teologia](#)  
[Baixar livros de Trabalho](#)  
[Baixar livros de Turismo](#)